

Magma Classic

Version 6.0.1
Dec 23rd, 2014

Data Structure Index

Data Structures

Here are the data structures with brief descriptions:

__attribute__	13
attachment_t	25
basic_message_t	26
cache_keys_t	28
cache_t	30
client_t	31
command_t	33
composition_t	34
connection_t	35
credential_t	40
hashed_bucket_t	42
http_content_t	43
http_data_t	44
http_page_t	45
imap_fetch_dataitems_t	47
imap_fetch_response_t	50
imap_folder_status_t	51
inx_t	53
ip_t	56
linked_node_t	58
linked_record_t	59
magma_keys_t	60
magma_t	62
mail_cache_t	77
mail_message_t	78
mail_mime_t	80
mappings_t	82
media_type_t	84
meta_folder_t	85
meta_message_t	87
meta_stats_tag_t	90
meta_user_t	91
multi_t	96
neue_t	100
nvp_t	102
object_cache_t	104
pool_t	105
queue_t	107
register_session_t	108
relay_keys_t	110

relay_t	112
serialization_t	113
server_config_t	114
server_keys_t	117
server_t	119
session_t	122
smtp_inbound_filter_t	126
smtp_inbound_prefs_t	128
smtp_message_t	135
smtp_outbound_prefs_t	137
smtp_recipients_t	139
smtp_session_t	140
stacker_node_t	144
stacker_t	145
statistics_vp_t	147
subnet_t	148
symbol_t	149
teacher_data_t	150
tok_state_t	152

File Index

File List

Here is a list of all files with brief descriptions:

magma/magma.c (The main executable entry point)	683
magma/magma.h (The global include file. This header includes both system headers and Magma module headers)	684
magma/queries.h (Assorted SQL queries used throughout Magma)	1354
magma/core/core.h (A collection of types, declarations and includes needed when accessing the core module and the type definitions needed to parse the header files that follow)	205
magma/core/type.c (A function for translating the M_TYPE enum into an equivalent string, typically so that a type code can be recorded as a string in error messages)	521
magma/core/buckets/arrays.c (A collection of functions used to create, maintain and safely utilize arrays of pointers)	154
magma/core/buckets/buckets.h (The function declarations and types for the thread-safe object pool interface)	159
magma/core/buckets/pool.c (A collection of functions used to create, maintain and safely utilize collections of object pointers that are accessed by multiple threads)	172
magma/core/buckets/stacked.c (An interface for handling FIFO stacks)	178
magma/core/classify/ascii.c (Functions used to classify ASCII characters)	180
magma/core/classify/classify.h (Functions used to classify characters into specific classes)	184
magma/core/compare/compare.h (Function declarations for various data and string comparison functions)	188
magma/core/compare/ends.c (Functions used to compare the ends of strings with other strings)	193
magma/core/compare/equal.c (Functions to check for string equality)	195
magma/core/compare/search.c (String search functions)	198
magma/core/compare/starts.c (Functions used to compare the starts of strings with other strings)	203
magma/core/encodings/base64.c (Functions for base64 encoding/decoding of data)	208
magma/core/encodings/encodings.h (Functions used to encode and decode data in various formats)	211
magma/core/encodings/hex.c (Functions for encoding/decoding hexadecimal data)	221
magma/core/encodings/mappings.c (Character-to-value mappings employed by the various base64 encoders)	225
magma/core/encodings/qp.c (Functions for encoding/decoding quoted printable data, as described by RFC 2045, section 6.7)	229
magma/core/encodings/url.c (Functions used to encode and decode website URLs)	231
magma/core/encodings/zbase32.c (A modified base32 encoding routine (zbase32) which selects characters to enhance readability. zbase32 strings may be used in URLs without any further encoding)	233
magma/core/hash/adler.c (An x86 implementation of the Adler hash algorithm)	235
magma/core/hash/crc.c (An x86 implementation of the 32-bit and 64-bit CRC algorithms)	236
magma/core/hash/fletcher.c (An implementation of the Fletcher hash algorithm)	240
magma/core/hash/hash.h (Declarations for hash functions that have been implemented internally)	241
magma/core/hash/murmur.c (An x64 implementation of the Murmur hash function)	244
magma/core/host/files.c (Generic system file I/O operations)	245

magma/core/host/folder.c (Functions for folder operations)	248
magma/core/host/host.c (Functions to retrieve information about the operating system)	249
magma/core/host/host.h (Provide access to system interfaces)	250
magma/core/host/mappings.c (Map numeric system values to their string equivalents)	227
magma/core/host/process.c (Functions for managing processes)	258
magma/core/host/spool.c (Functions for checking, creating, maintaining and using the spool)	262
magma/core/indexes/cursors.c (The generic index interface for handling cursors)	266
magma/core/indexes/hashed.c (Function declarations and types for the hashed list)	270
magma/core/indexes/indexes.h (Function declarations and types for the generic index interface)	275
magma/core/indexes/inx.c (The generic index interface used to abstract away the underlying data structure used for storage)	286
magma/core/indexes/linked.c (The linked list implementation functions utilized by the generic index interface)	292
magma/core/log/log.c (Internal logging functions. This function should be accessed using the appropriate macro)	300
magma/core/log/log.h (The function declarations and macros needed to access the error log subsystem)	303
magma/core/memory/align.c (Functions for memory alignment operations)	310
magma/core/memory/bits.c (A collection of functions used handle common bit manipulation tasks)	311
magma/core/memory/memory.c (The functions used to handle Magma memory buffers)	312
magma/core/memory/memory.h (The functions used to allocate and manipulate blocks of memory off the heap and inside the secured address space)	317
magma/core/memory/secure.c (Functions for allocating secure memory. Secure buffers should always be used to hold sensitive information)	326
magma/core/parsers/case.c (A collection of functions used for manipulating the capitalization of characters)	332
magma/core/parsers/line.c (Functions used to extract lines from within a larger block of data)	338
magma/core/parsers/parsers.h (The function declarations and types needed by the generic parsing functions)	369
magma/core/parsers/time.c (Functions used to parse time)	396
magma/core/parsers/token.c (Functions for tokenizing strings)	398
magma/core/parsers/trim.c (Functions used to trim whitespace from strings)	405
magma/core/parsers/formats/formats.h (Function declarations and types used by the structured format parsers)	334
magma/core/parsers/formats/nvp.c (Interface to the name/value pair parser)	336
magma/core/parsers/numbers/digits.c (Functions for counting the digit places in a number, including the sign character for signed numbers)	340
magma/core/parsers/numbers/numbers.c (Functions for converting different string types into binary numbers)	343
magma/core/parsers/numbers/numbers.h (Function declarations for the number conversion functions)	355
magma/core/parsers/special/bracket.c (Functions for extracting a value inside a pair of brackets)	394
magma/core/parsers/special/special.h (Declarations for functions involved in extracting data from specialized formats)	395
magma/core/strings/allocation.c (Functions used to allocate stringers)	407
magma/core/strings/data.c (Functions used to inspect the data of managed strings)	415

magma/core/strings/info.c (A collection of functions used to extract information from stringer options)	425
magma/core/strings/length.c (Fncions used to find stringer lengths)	428
magma/core/strings/multi.c (Functions for handling the multi-type structure)	432
magma/core/strings/nuller.c (Functions for handling null terminated strings)	437
magma/core/strings/opts.c (Functions for handling managed string options)	442
magma/core/strings/print.c (Functions for printing formatted data to managed strings)	444
magma/core/strings/replace.c (Functions used for string replacement)	447
magma/core/strings/shortcuts.c (A collection of shortcuts used to call various string functions using sensible default values)	448
magma/core/strings/strings.h (Function declarations and types used by the different modules involved with handling stringers and null terminated strings)	453
magma/core/strings/validate.c (A collection of functions used to validate stringer allocation option combinations)	493
magma/core/thread/keys.c (Functions for handling thread local storage keys)	497
magma/core/thread/mutex.c (Functions for thread coordination via a mutex)	499
magma/core/thread/rwlock.c (Functions for thread coordination via a read/write lock)	502
magma/core/thread/thread.c (Functions for spawning new threads, and retrieving their exit statuses)	506
magma/core/thread/thread.h (Interface for providing pthread functionality)	511
magma/engine/engine.h (The engine module that contains all of the logic to control the overall execution of the system)	657
magma/engine/config/config.h (The entry point for all configuration modules used by magma)	543
magma/engine/config/cache/cache.c (Functions for handling the cache instance configurations)	522
magma/engine/config/cache/cache.h (Types and functions for creating and accessing the cache structures)	535
magma/engine/config/cache/keys.h (A collection of keys that define the configuration of cache instances)	539
magma/engine/config/global/datatier.c (Database interface for the engine module)	553
magma/engine/config/global/global.c (Functions for handling the global configuration)	592
magma/engine/config/global/global.h (The global configuration structure used for overall system settings, and functions to initialize it at startup and free it at shutdown)	599
magma/engine/config/global/keys.h (A collection of keys that define access rules, default values and other parameters needed when loading the global configuration)	540
magma/engine/config/relay/keys.h (A collection of keys that define access rules, default values and other parameters needed to configure relay instances)	541
magma/engine/config/relay/relay.c (Functions for handling relay server instance configurations)	606
magma/engine/config/relay/relay.h (The types and functions involved with creating and accessing the relay structures)	613
magma/engine/config/servers/keys.h (A collection of keys that define access rules, default values and other parameters needed when configuring server instances)	542
magma/engine/config/servers/servers.c (Functions for handling the server instance configurations)	617
magma/engine/config/servers/servers.h (The types and functions involved in creating and accessing the server structure)	623
magma/engine/context/args.c (Functions for parsing command line arguments)	630
magma/engine/context/context.h (Functions involved in initializing, manipulating and verifying the operating context for the system)	632

magma/engine/context/process.c (Functions used to start and stop the daemon, including the execution of the module init and module clean up functions)	259
magma/engine/context/sanity.c (A collection of checks performed at launch to make sure the system will operate as expected)	640
magma/engine/context/signal.c (A collection of functions used to register and handle signals)	641
magma/engine/context/system.c (Functions used to interface with and configure the operating system)	644
magma/engine/context/thread.c (The thread start and stop functions)	510
magma/engine/controller/controller.h (Functions involved with assigning tasks to worker threads and routing network connections to the proper protocol subsystem)	647
magma/engine/controller/protocol.c (Functions used to route incoming network connections to their designated protocol modules. This layer also tracks overall protocol statistics and establishes SSL connections for connections that arrive on secure ports)	651
magma/engine/controller/queue.c (Functions used to distribute tasks to available worker threads)	653
magma/engine/status/build.c (Functions for retrieving the version and build information)	658
magma/engine/status/build.h	659
magma/engine/status/performance.c (A collection of functions used to measure and track system performance)	660
magma/engine/status/statistics.c (A collection of functions used to track and access system statistics)	661
magma/engine/status/status.c (Functions used to coordinate system state and worker thread operation and shutdown.)	671
magma/engine/status/status.h (Functions involved with tracking system status and performance)	674
magma/network/addresses.c (A collection of functions used to allocate, configure and destroy connection structures)	688
magma/network/clients.c (Functions for handling network client connections)	696
magma/network/connections.c (A collection of functions to allocate, configure and destroy connection structures)	698
magma/network/http.h (The HTTP server control structures)	701
magma/network/imap.h (The IMAP server control structures)	723
magma/network/listeners.c (Functions used to listen for incoming connections on specific ports. Successful socket connections are then passed up to the controller for routing)	756
magma/network/meta.h (Meta information structures/types for users, folders, messages, etc)	758
magma/network/network.h (The types and functions for abstracting access to network functionality)	760
magma/network/options.c (Functions used to set network socket options)	783
magma/network/pop.h (The POP control structures)	786
magma/network/read.c (Functions used to read in data via the network)	798
magma/network/reverse.c (Function to perform reverse DNS lookups)	800
magma/network/sessions.h (Structures for handling web session data)	803
magma/network/smtp.h (Structures used to control SMTP connections/sessions)	812
magma/network/write.c (Functions used to write data out via the network)	839
magma/objects/locks.c (Functions for managing locks synchronized via memcached)	910
magma/objects/objects.c (Functions used for managing objects)	970
magma/objects/objects.h (Functions used to interface with objects)	1005
magma/objects/serials.c (Functions used for track and update the object checkpoints)	1010

magma/objects/config/config.c (The user configuration interface)	843
magma/objects/config/config.h (The user configuration interface)	548
magma/objects/config/datatier.c (The database routines for the user configuration interface)	554
magma/objects/contacts/contacts.c (The interface to managing user address book contacts)	848
magma/objects/contacts/contacts.h (Interface for managing user address book contacts)	859
magma/objects/contacts/datatier.c (Functions for handling user contacts in the database)	556
magma/objects/contacts/find.c (Functions to search a collection of contacts)	869
magma/objects/folders/contacts.c (Interface for user contact folders)	853
magma/objects/folders/datatier.c (Folder data functions)	560
magma/objects/folders/find.c (Functions to search for specified folders)	871
magma/objects/folders/folders.c (Interface with user folders)	873
magma/objects/folders/folders.h (Interface to managing user folders)	888
magma/objects/folders/messages.c (Interface with user message folders)	899
magma/objects/mail/cache.c (Functions used to cache a mail message in its parsed form)	526
magma/objects/mail/cleanup.c (Functions for cleaning up mail messages)	913
magma/objects/mail/counters.c (Functions used to calculate mail message metrics needed to implement business rules)	915
magma/objects/mail/datatier.c (Functions used to interface with and manage message data)	562
magma/objects/mail/headers.c (Functions used to handle mail message header information)	918
magma/objects/mail/load_message.c (Functions used to load mail messages)	924
magma/objects/mail/mail.h (Functions used to interface with and manage message data)	927
magma/objects/mail/mime.c (Functions used to parse and manipulate MIME messages)	959
magma/objects/mail/objects.c (Functions used to interface with and manage message data)	968
magma/objects/mail/parsing.c (Functions used to parse mail messages and extract information from them)	972
magma/objects/mail/paths.c (Functions for mapping mail messages to their persistent file storage paths)	974
magma/objects/mail/remove_message.c (Functions used to delete messages)	976
magma/objects/mail/signatures.c (Functions used to insert signatures into mail messages)	977
magma/objects/mail/store_message.c (Functions used to store and copy mail message data)	983
magma/objects/messages/datatier.c (Message data functions)	565
magma/objects/messages/messages.c (Mail message interface functions)	901
magma/objects/messages/messages.h (Mail message operations)	986
magma/objects/messages/meta.c (Functions to interface with the deprecated meta_message_t structure)	993
magma/objects/neue/credentials.c	998
magma/objects/neue/neue.c (Neue entry points)	1001
magma/objects/neue/neue.h	1002
magma/objects/sessions/sessions.c (Routines to handle web sessions)	1013
magma/objects/sessions/sessions.h (Functions for handling web sessions)	804
magma/objects/users/alerts.c (Functions for handling user alerts and warnings)	1029
magma/objects/users/aliases.c (Functions for handling user aliases)	1030
magma/objects/users/contacts.c (The user context interface for contacts)	856
magma/objects/users/datatier.c (The database interface for the user objects)	566
magma/objects/users/folders.c (Functions used for handling message folders)	876
magma/objects/users/messages.c (The user context interface for message folders)	903

magma/objects/users/users.c (Functions for handling the user context)	1031
magma/objects/users/users.h (Functions to interface with and manage user data)	1039
magma/objects/warehouse/datatier.c (Functions used by the warehouse objects to access the database)	578
magma/objects/warehouse/domains.c (Functions for managing the list of system domain names)	1065
magma/objects/warehouse/patterns.c (Functions used to manage the list of spam patterns that are scanned for detection in outbound messages)	1069
magma/objects/warehouse/warehouse.c (The warehouse management functions)	1072
magma/objects/warehouse/warehouse.h (Functions to provide access to warehoused reference data needed to make intelligent decisions)	1074
magma/providers/providers.h (The entry point for the provider modules)	1285
magma/providers/symbols.c (Functions used to load the external library symbols)	1303
magma/providers/symbols.h (External function pointers/definitions)	1305
magma/providers/checkers/allocations.h (Functions used to scan, analyze, check, and validate data)	1080
magma/providers/checkers/checkers.h (Functions used to scan, analyze, check, and validate data)	1082
magma/providers/checkers/clamav.c (Interface to the ClamAV library)	1093
magma/providers/checkers/dkim.c (Functions used to generate and verify Domain Keys Identified Mail (DKIM))	1097
magma/providers/checkers/dspam.c (DSPAM interface functions)	1101
magma/providers/checkers/spf.c (The functions used to validate SPF information)	1104
magma/providers/compress/bzip.c (The interface for the BZIP compression functions)	1107
magma/providers/compress/compress.c (Interface to the compression functions)	1109
magma/providers/compress/compress.h (Compression interface functions/handlers)	1113
magma/providers/compress/engine.c (Generic engine independent compression interfaces)	1122
magma/providers/compress/lzo.c (The interface for the LZO compression functions)	1123
magma/providers/compress/zlib.c (The interface for the ZLIB compression functions)	1125
magma/providers/consumers/cache.c (Distributed cache interface)	529
magma/providers/consumers/consumers.h (Interfaces for clients that consume/load/retrieve data across various network protocols)	1127
magma/providers/consumers/counters.c	917
magma/providers/consumers/deserialization.c (Distributed cache interface for de-serializing various data types)	1140
magma/providers/consumers/serialization.c (Distributed cache interface for serializing various data types)	1145
magma/providers/cryptography/ciphers.c (Functions used to handle cryptographic ciphers)	1150
magma/providers/cryptography/cryptex.c (Functions for handling the secure data type)	1153
magma/providers/cryptography/cryptography.h (Functions used to perform cryptographic operations and provide truly random data)	1157
magma/providers/cryptography/digest.c (Functions used to create a secure one-way hash of an arbitrary input buffer)	1187
magma/providers/cryptography/ecies.c (ECIES encryption/decryption functions)	1190
magma/providers/cryptography/openssl.c (The interface to OpenSSL routines)	1196
magma/providers/cryptography/random.c (A collection of functions for generating random data)	1204
magma/providers/cryptography/scramble.c (Functions used to handle symmetric encryption)	1209

magma/providers/cryptography/symmetric.c (Functions used to encrypt/decrypt data using symmetric ciphers)	1214
magma/providers/database/database.h (Functions used to interface with the database)	1216
magma/providers/database/mysql.c (MYSQL Symbols)	1238
magma/providers/database/query.c (Traditional SQL queries in string form)	1244
magma/providers/database/results.c (A collection of functions for handling MySQL result sets)	1246
magma/providers/database/stmts.c (A collection of functions for working with the MySQL prepared statement interface)	1255
magma/providers/database/transaction.c (MySQL transaction interface)	1262
magma/providers/images/freetype.c (Functions used to handle fonts)	1264
magma/providers/images/gd.c (The functions used to create and images using the GD library)	1265
magma/providers/images/images.h (The functions used to create and modify image files. For our purposes that include fonts)	1266
magma/providers/images/jpeg.c (The functions used to create and modify JPEG image files)	1269
magma/providers/images/png.c (The functions used to create and modify PNG image files)	1270
magma/providers/parsers/json.c (JSON serialization)	1271
magma/providers/parsers/parsers.h (The entry point for modules involved with accessing functionality provided by alien code)	381
magma/providers/parsers/xml.c (The interface to the xml parser)	1272
magma/providers/storage/data.c (The meta functions needed by the engine module)	420
magma/providers/storage/storage.h (The Tokyo Cabinet interface, which is primarily used for memory and disk based storage)	1287
magma/providers/storage/tank.c (The storage system interface. Uses Tokyo Cabinet to store the underlying files)	1293
magma/providers/storage/tokyo.c (Tokyo Cabinet symbols)	1298
magma/providers/storage/tree.c (Use Tokyo Cabinet to provide a tree based index implementation for the generic index interface)	1299
magma/servers/servers.h (The inclusion point for the different server implementations)	629
magma/servers/http/commands.h (The data structure involved with parsing and routing POP commands)	1369
magma/servers/http/content.c (Functions for handling the management of web server content)	1374
magma/servers/http/data.c (Assorted functions for handling HTTP data)	422
magma/servers/http/errors.c (Error page templates)	1380
magma/servers/http/http.c (Functions used to handle HTTP commands and actions)	1384
magma/servers/http/http.h (The entry point for the HTTP server module)	703
magma/servers/http/parse.c (Functions used to parse an HTTP request)	1387
magma/servers/http/response.c (Functions for fulfilling responses to http client requests)	1403
magma/servers/http/sessions.c (HTTP session handlers)	1021
magma/servers/imap/commands.c (The functions involved with parsing and routing POP commands)	1407
magma/servers/imap/commands.h (The data structure involved with parsing and routing POP commands)	1370
magma/servers/imap/fetch.c (Functions used to handle IMAP commands/actions)	1415
magma/servers/imap/fetch_response.c (Functions used to handle IMAP commands/actions)	1420
magma/servers/imap/flags.c (Functions to handle IMAP commands/actions)	1421
magma/servers/imap/folders.c (Functions used to handle IMAP commands/actions)	880
magma/servers/imap/imap.c (Functions used to handle IMAP commands/actions)	1425

magma/servers/imap/imap.h (The entry point for the IMAP server module)	724
magma/servers/imap/messages.c (Functions used to handle IMAP commands and actions)	904
magma/servers/imap/output.c (Functions used to handle IMAP commands/actions)	1433
magma/servers/imap/parse.c (Functions used to handle IMAP commands and actions)	1391
magma/servers/imap/parse_address.c (Functions used to handle IMAP commands/actions)	1434
magma/servers/imap/range.c (Functions used to handle IMAP commands/actions)	1436
magma/servers/imap/search.c (Functions used to handle IMAP commands/actions)	200
magma/servers/imap/sessions.c (IMAP session handlers)	1023
magma/servers/molten/commands.c (Functions used to parse protocol specific data out of the inbound network stream)	1409
magma/servers/molten/commands.h (The data structure involved with parsing and routing Molten commands)	1371
magma/servers/molten/molten.c (Functions used to handle Molten commands/actions)	1437
magma/servers/molten/molten.h (The entry point for the Molten server module)	1439
magma/servers/molten/sessions.c (Molten session handlers)	1024
magma/servers/pop/commands.c (Functions involved with parsing and dispatching POP commands)	1411
magma/servers/pop/commands.h (The data structure involved with parsing and routing POP commands)	1372
magma/servers/pop/mailbox.c (Utility functions used to retrieve POP3 message-related statistics)	1441
magma/servers/pop/parse.c (The POP protocol parsers)	1397
magma/servers/pop/pop.c (Functions used to handle POP commands/actions)	1443
magma/servers/pop/pop.h (The entry point for the POP server module)	787
magma/servers/pop/sessions.c (Functions for managing POP3 sessions)	1025
magma/servers/smtp/accept.c (Functions used to handle SMTP commands/actions)	1449
magma/servers/smtp/checkers.c (Functions used by the SMTP protocol to check and if necessary validate data using external information)	1452
magma/servers/smtp/commands.c (The functions involved with parsing and routing SMTP commands)	1413
magma/servers/smtp/commands.h (The data structure for parsing and routing SMTP commands)	1373
magma/servers/smtp/datatier.c (Functions used to interface with and manage data needed by the SMTP protocol)	579
magma/servers/smtp/messages.c (Handle the SMTP message structure)	906
magma/servers/smtp/parse.c (Functions used to parse command parameters from SMTP clients)	1399
magma/servers/smtp/relay.c (Functions to relay messages via SMTP to the outbound mail server)	610
magma/servers/smtp/session.c (Functions used to handle SMTP sessions)	1455
magma/servers/smtp/smtp.c (Functions used to handle SMTP commands/actions)	1459
magma/servers/smtp/smtp.h (The entry point for the SMTP server module)	814
magma/servers/smtp/transmit.c (Handle replies)	1466
magma/web/web.h (The web application modules)	1573
magma/web/contact/abuse.c (Functions for detecting abuse of the contact form)	1469
magma/web/contact/business.c (Functions for handling the logic of the contact form)	1474
magma/web/contact/contact.c (Handle the contact form)	1479

magma/web/contact/contact.h (Definitions for handling the web contact form)	1481
magma/web/portal/config.c (Utilities functions for handling user config entries)	846
magma/web/portal/contacts.c (Functions for handling user contact list entries)	857
magma/web/portal/endpoint.c (The control logic for the Portal JSON endpoint)	1485
magma/web/portal/flags.c (Functions for handling message flags and tags)	1423
magma/web/portal/folders.c (Folder manipulation functions for use by the portal)	885
magma/web/portal/mail.c (Functions for various smtp-level functionality for use in conjunction with the portal)	1506
magma/web/portal/messages.c (Functions to help assemble mail message for output)	907
magma/web/portal/methods.h (Definitions for all supported web portal methods and their calling interfaces)	1509
magma/web/portal/parse.c (Json-rpc request parameter parsers for the portal)	1401
magma/web/portal/portal.c (The portal web application)	1510
magma/web/portal/portal.h (The portal web application)	1512
magma/web/register/abuse.c (Functions for handling potential abuse of the new user registration process)	1471
magma/web/register/business.c (Functions for handling validation for the registration process)	1476
magma/web/register/captcha.c (The captcha interface for the registration process)	1546
magma/web/register/datatier.c (Functions for handling the new user registration process)	585
magma/web/register/register.c (Functions for handling the registration process)	1548
magma/web/register/register.h (Functions for handling the registration process)	1551
magma/web/register/sessions.c (Functions for handling the internal session structure used by new user registration attempts)	1027
magma/web/statistics/datatier.c (Interface for harvesting statistics from the database for the portal statistics app)	587
magma/web/statistics/statistics.c (Generate a dynamic web page with statistics on server performance)	669
magma/web/statistics/statistics.h (Code for dynamically generating the portal statistics page)	1562
magma/web/teacher/datatier.c (Allow users to train their statistical mail filter)	589
magma/web/teacher/teacher.c (Functions to allow users to train the statistical mail filter)	1565
magma/web/teacher/teacher.h (A facility for allowing users to train the statistical mail filter)	1568

Data Structure Documentation

__attribute__ Struct Reference

```
#include <indexes.h>
```

Data Fields

- **inx_t * inx**
- **hashed_bucket_t * bucket**
- **uint64_t serial**
- **uint64_t slot**
- **uint64_t count**
- **uint32_t buckets**
- **linked_node_t * node**
- **uint64_t position**
- **uint32_t flags**
- **size_t length**
- **struct {**
 - **int_t cookie**
 - **int_t connection**
 - **} response**
- **session_t * session**
- **http_method_t method**
- **inx_t * pairs**
- **inx_t * headers**
- **int_t mode**
- **int_t merged**
- **int_t port**
- **stringer_t * host**
- **stringer_t * location**
- **stringer_t * cookie**
- **stringer_t * agent**
- **stringer_t * body**
- **union {**
 - **struct {**
 - **uint64_t id**
 - **json_t * request**
 - **json_t * params**
 - **} portal**
- **};**
- **meta_user_t * user**
- **imap_arguments_t * arguments**
- **stringer_t * tag**
- **stringer_t * command**
- **stringer_t * username**
- **int_t read_only**
- **int_t uid**
- **int_t session_state**
- **uint64_t selected**

- uint64_t **user_checkpoint**
- uint64_t **messages_checkpoint**
- uint64_t **folders_checkpoint**
- uint64_t **messages_recent**
- uint64_t **messages_total**
- uint64_t **alertnum**
- uint64_t **created**
- **stringer_t** * **type**
- **stringer_t** * **message**
- **bool_t** **selected**
- uint64_t **aliasnum**
- **stringer_t** * **display**
- **stringer_t** * **address**
- **bool_t** **expunge**
- **int_t** **spf**
- **int_t** **dkim**
- **int_t** **wildcard**
- **int_t** **mailboxes**
- **int_t** **restricted**
- **stringer_t** * **domain**
- uint8_t **engine**
- struct {
 - uint64_t **original**
 - uint64_t **compressed**
 - } **length**
- struct {
 - uint64_t **original**
 - uint64_t **compressed**
 - } **hash**
- struct {
 - uint64_t **envelope**
 - uint64_t **hmac**
 - uint64_t **original**
 - uint64_t **body**
 - } **length**
- uint32_t **engine**
- struct {
 - uint64_t **vector**
 - uint64_t **original**
 - uint64_t **scrambled**
 - } **length**
- struct {
 - uint32_t **scrambled**
 - } **hash**
- uint8_t **ver**
- uint8_t **rec**
- uint64_t **flags**
- struct {
 - uint64_t **tnum**
 - uint64_t **unum**
 - uint64_t **onum**
 - uint64_t **snum**
 - uint64_t **created**

- } **meta**
- struct {
- uint64_t **length**
- uint64_t **compressed**
- uint64_t **encrypted**
- } **data**
- struct {
- uint64_t **tnum**
- uint64_t **unum**
- uint64_t **onum**
- uint64_t **snum**
- uint64_t **stamp**
- } **meta**
- struct {
- uint64_t **created**
- uint64_t **updated**
- uint64_t **deleted**
- uint64_t **expiration**
- } **stamps**
- void * **tree**
- **multi_t** **key**
- void * **value**

Detailed Description

HIGH: Create an interface for loading SSL/TLS and DKIM certificates from the database. HIGH: Create an interface for checking whether an address or IP is trusted.

Definition at line 24 of file hashed.c.

Field Documentation

union { ... }

stringer_t * __attribute__ ::address

Definition at line 70 of file meta.h.

stringer_t * __attribute__ ::agent

Definition at line 69 of file http.h.

uint64_t __attribute__ ::alertnum

Definition at line 58 of file meta.h.

uint64_t __attribute__ ::aliasnum

Definition at line 69 of file meta.h.

imap_arguments_t* __attribute__::arguments

Definition at line 35 of file imap.h.

uint64_t __attribute__::body

Definition at line 46 of file cryptography.h.

stringer_t * __attribute__::body

Definition at line 69 of file http.h.

Referenced by __attribute__().

hashed_bucket_t* __attribute__::bucket

Definition at line 26 of file hashed.c.

uint32_t __attribute__::buckets

Definition at line 31 of file hashed.c.

stringer_t * __attribute__::command

Definition at line 36 of file imap.h.

uint64_t __attribute__::compressed

The compressed length of the data, if applicable.

Definition at line 28 of file compress.h.

int_t __attribute__::connection

Definition at line 62 of file http.h.

stringer_t * __attribute__::cookie

Definition at line 69 of file http.h.

int_t __attribute__::cookie

Definition at line 61 of file http.h.

uint64_t __attribute__::count

Definition at line 27 of file hashed.c.

uint64_t __attribute__::created

Time stamp taken when the object is stored.

When the object was first created.

Definition at line 58 of file meta.h.

Referenced by __attribute__().

struct { ... } __attribute__::data

Referenced by __attribute__().

uint64_t __attribute__::deleted

When the object was flagged for deletion.

Definition at line 69 of file storage.h.

stringer_t* __attribute__::display

Definition at line 70 of file meta.h.

int_t __attribute__::dkim

Definition at line 21 of file warehouse.h.

stringer_t* __attribute__::domain

Definition at line 25 of file warehouse.h.

uint64_t __attribute__::encrypted

The length of the encrypted data block, if applicable.

Definition at line 49 of file storage.h.

uint32_t __attribute__::engine

Definition at line 52 of file cryptography.h.

uint8_t __attribute__::engine

Definition at line 24 of file compress.h.

uint64_t __attribute__::envelope

Definition at line 43 of file cryptography.h.

uint64_t __attribute__::expiration

When archived/deleted objects can be permanently purged.

Definition at line 70 of file storage.h.

bool_t __attribute__::expunge

Definition at line 17 of file pop.h.

uint64_t __attribute__::flags

A collection of bit mask flags to indicate whether the object was compressed, encrypted, or replicated.

Definition at line 36 of file storage.h.

uint32_t __attribute__::flags

Definition at line 21 of file secure.c.

Referenced by __attribute__().

uint64_t __attribute__::folders_checkpoint

Definition at line 38 of file imap.h.

struct { ... } __attribute__::hash

struct { ... } __attribute__::hash

inx_t * __attribute__::headers

Definition at line 67 of file http.h.

uint64_t __attribute__::hmac

Definition at line 44 of file cryptography.h.

stringer_t* __attribute__::host

Definition at line 69 of file http.h.

uint64_t __attribute__::id

Definition at line 73 of file http.h.

inx_t * __attribute__::inx

Definition at line 25 of file hashed.c.

multi_t __attribute__::key

Definition at line 18 of file tree.c.

Referenced by __attribute__().

uint64_t __attribute__::length

The full length of the original data.

Definition at line 47 of file storage.h.

struct { ... } __attribute__::length

struct { ... } __attribute__::length

struct { ... } __attribute__::length

size_t __attribute__::length

Definition at line 22 of file secure.c.

Referenced by __attribute__().

stringer_t * __attribute__::location

Definition at line 69 of file http.h.

int_t __attribute__::mailboxes

Definition at line 23 of file warehouse.h.

int_t __attribute__::merged

Definition at line 68 of file http.h.

stringer_t * __attribute__::message

Definition at line 59 of file meta.h.

Referenced by __attribute__().

uint64_t __attribute__::messages_checkpoint

Definition at line 38 of file imap.h.

uint64_t __attribute__::messages_recent

Definition at line 38 of file imap.h.

uint64_t __attribute__::messages_total

Definition at line 38 of file imap.h.

struct { ... } __attribute__::meta

struct { ... } __attribute__::meta

http_method_t __attribute__::method

Definition at line 66 of file http.h.

int_t __attribute__::mode

Definition at line 68 of file http.h.

linked_node_t* __attribute__::node

Definition at line 29 of file linked.c.

uint64_t __attribute__::onum

The object number.

Definition at line 41 of file storage.h.

uint64_t __attribute__::original

Definition at line 27 of file compress.h.

inx_t* __attribute__::pairs

Definition at line 67 of file http.h.

json_t * __attribute__::params

Definition at line 74 of file http.h.

int_t __attribute__::port

Definition at line 68 of file http.h.

struct { ... } __attribute__::portal

uint64_t __attribute__::position

Definition at line 30 of file linked.c.

int_t __attribute__::read_only

Definition at line 37 of file imap.h.

uint8_t __attribute__::rec

The length of the record data.

Definition at line 35 of file storage.h.

json_t* __attribute__::request

Definition at line 74 of file http.h.

struct { ... } __attribute__::response

int_t __attribute__::restricted

Definition at line 24 of file warehouse.h.

uint32_t __attribute__::scrambled

Definition at line 62 of file cryptography.h.

uint64_t __attribute__::scrambled

Definition at line 57 of file cryptography.h.

bool_t __attribute__::selected

Definition at line 68 of file meta.h.

uint64_t __attribute__::selected

Definition at line 38 of file imap.h.

uint64_t __attribute__::serial

Definition at line 27 of file hashed.c.

Referenced by __attribute__().

session_t* __attribute__::session

Definition at line 65 of file http.h.

int_t __attribute__::session_state

Definition at line 37 of file imap.h.

uint64_t __attribute__::slot

Definition at line 27 of file hashed.c.

uint64_t __attribute__::snum

The serial number. Starts at zero, and increments for each update.

Definition at line 42 of file storage.h.

int_t __attribute__::spf

Definition at line 20 of file warehouse.h.

uint64_t __attribute__::stamp

Time stamp taken when the object is stored.

Definition at line 63 of file storage.h.

struct { ... } __attribute__::stamps

stringer_t* __attribute__::tag

Definition at line 36 of file imap.h.

uint64_t __attribute__::tnum

Which local storage tank was used to store the object.

Definition at line 39 of file storage.h.

void* __attribute__::tree

Definition at line 17 of file tree.c.

stringer_t* __attribute__::type

Definition at line 59 of file meta.h.

int_t __attribute__::uid

Definition at line 37 of file imap.h.

uint64_t __attribute__::unum

The user number of the object owner.

Definition at line 40 of file storage.h.

uint64_t __attribute__::updated

When the object was last updated.

Definition at line 68 of file storage.h.

meta_user_t * __attribute__::user

Definition at line 34 of file imap.h.

uint64_t __attribute__::user_checkpoint

Definition at line 38 of file imap.h.

stringer_t * __attribute__::username

Definition at line 36 of file imap.h.

void* __attribute__::value

Definition at line 19 of file tree.c.

uint64_t __attribute__::vector

Definition at line 55 of file cryptography.h.

uint8_t __attribute__::ver

Number indicating the entry version, which also tells us the layout of the data.

Definition at line 34 of file storage.h.

int_t __attribute__::wildcard

Definition at line 22 of file warehouse.h.

The documentation for this struct was generated from the following files:

- magma/core/indexes/**hashed.c**
- magma/core/indexes/**indexes.h**
- magma/core/indexes/**linked.c**
- magma/core/memory/**secure.c**
- magma/network/**http.h**
- magma/network/**imap.h**
- magma/network/**meta.h**
- magma/network/**pop.h**
- magma/objects/warehouse/**warehouse.h**
- magma/providers/compress/**compress.h**
- magma/providers/cryptography/**cryptography.h**
- magma/providers/storage/**storage.h**
- magma/providers/storage/**tree.c**

attachment_t Struct Reference

```
#include <sessions.h>
```

Data Fields

- `uint64_t attach_id`
 - `stringer_t * filename`
 - `stringer_t * filedata`
-

Detailed Description

Definition at line 22 of file sessions.h.

Field Documentation

`uint64_t attachment_t::attach_id`

Definition at line 23 of file sessions.h.

Referenced by `portal_endpoint_attachments_add()`.

`stringer_t* attachment_t::filedata`

Definition at line 25 of file sessions.h.

Referenced by `portal_get_upload_attachment()`, `portal_smtp_create_data()`, `portal_upload()`, and `sess_release_attachment()`.

`stringer_t* attachment_t::filename`

Definition at line 24 of file sessions.h.

Referenced by `portal_endpoint_attachments_add()`, `portal_smtp_create_data()`, and `sess_release_attachment()`.

The documentation for this struct was generated from the following file:

- `magma/network/sessions.h`

basic_message_t Struct Reference

```
#include <mail.h>
```

Data Fields

- `placer_t to`
- `placer_t from`
- `placer_t date`
- `placer_t subject`
- `stringer_t * text`

Detailed Description

Definition at line 24 of file mail.h.

Field Documentation

`placer_t basic_message_t::date`

Definition at line 27 of file mail.h.

Referenced by mail_setup_basic().

`placer_t basic_message_t::from`

Definition at line 26 of file mail.h.

Referenced by mail_setup_basic().

`placer_t basic_message_t::subject`

Definition at line 28 of file mail.h.

Referenced by mail_setup_basic().

`stringer_t* basic_message_t::text`

Definition at line 29 of file mail.h.

Referenced by mail_setup_basic().

`placer_t basic_message_t::to`

Definition at line 25 of file mail.h.

Referenced by mail_setup_basic().

The documentation for this struct was generated from the following file:

- magma/objects/mail/**mail.h**

cache_keys_t Struct Reference

```
#include <cache.h>
```

Data Fields

- `size_t offset`
 - `multi_t norm`
 - `char * name`
 - `char * description`
 - `bool_t required`
-

Detailed Description

Definition at line 16 of file `cache.h`.

Field Documentation

`char* cache_keys_t::description`

Definition at line 20 of file `cache.h`.

`char* cache_keys_t::name`

Definition at line 19 of file `cache.h`.

Referenced by `cache_set_value()`.

`multi_t cache_keys_t::norm`

Definition at line 18 of file `cache.h`.

Referenced by `cache_free()`, `cache_output_help()`, `cache_output_settings()`, `cache_set_value()`, and `cache_validate()`.

`size_t cache_keys_t::offset`

Definition at line 17 of file `cache.h`.

Referenced by `cache_free()`, `cache_output_settings()`, `cache_set_value()`, and `cache_validate()`.

`bool_t cache_keys_t::required`

Definition at line 21 of file `cache.h`.

Referenced by `cache_output_help()`.

The documentation for this struct was generated from the following file:

- `magma/engine/config/cache/cache.h`

cache_t Struct Reference

```
#include <cache.h>
```

Data Fields

- char * **name**
 - uint32_t **port**
 - uint32_t **weight**
-

Detailed Description

Definition at line 24 of file cache.h.

Field Documentation

char* cache_t::name

Definition at line 25 of file cache.h.

uint32_t cache_t::port

Definition at line 26 of file cache.h.

uint32_t cache_t::weight

Definition at line 27 of file cache.h.

The documentation for this struct was generated from the following file:

- magma/engine/config/cache/**cache.h**

client_t Struct Reference

```
#include <network.h>
```

Data Fields

- void * **ssl**
 - int **sockd**
 - int **status**
 - **placer_t** **line**
 - **stringer_t** * **buffer**
-

Detailed Description

Definition at line 62 of file network.h.

Field Documentation

stringer_t* client_t::buffer

Definition at line 67 of file network.h.

Referenced by client_close(), client_connect(), client_read(), client_read_line(), smtp_client_connect(), smtp_client_send_data(), smtp_client_send_helo(), smtp_client_send_mailfrom(), smtp_client_send_nullfrom(), and smtp_client_send_rcptto().

placer_t client_t::line

Definition at line 66 of file network.h.

Referenced by client_connect(), client_read(), client_read_line(), smtp_client_connect(), smtp_client_send_data(), smtp_client_send_helo(), smtp_client_send_mailfrom(), smtp_client_send_nullfrom(), smtp_client_send_rcptto(), and smtp_relay_message().

int client_t::sockd

Definition at line 64 of file network.h.

Referenced by client_close(), client_connect(), client_print(), client_read(), client_read_line(), client_secure(), client_status(), client_write(), and smtp_client_connect().

void* client_t::ssl

Definition at line 63 of file network.h.

Referenced by client_close(), client_read(), client_read_line(), client_secure(), and client_write().

int client_t::status

Definition at line 65 of file network.h.

Referenced by `client_connect()`, `client_read()`, `client_read_line()`, `client_secure()`, `client_status()`, and `client_write()`.

The documentation for this struct was generated from the following file:

- magma/network/**network.h**

command_t Struct Reference

```
#include <network.h>
```

Data Fields

- char * **string**
 - size_t **length**
 - void * **function**
-

Detailed Description

Definition at line 55 of file network.h.

Field Documentation

void* command_t::function

Definition at line 58 of file network.h.

Referenced by imap_process(), molten_parse(), pop_process(), portal_endpoint(), and smtp_process().

size_t command_t::length

Definition at line 57 of file network.h.

Referenced by imap_compare(), imap_process(), molten_compare(), molten_parse(), pop_compare(), pop_process(), portal_endpoint(), portal_endpoint_compare(), smtp_compare(), smtp_disabled(), and smtp_process().

char* command_t::string

Definition at line 56 of file network.h.

Referenced by imap_compare(), imap_process(), molten_compare(), molten_parse(), pop_compare(), pop_process(), portal_endpoint(), portal_endpoint_compare(), smtp_compare(), smtp_disabled(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(), smtp_parse_rcpt_to(), and smtp_process().

The documentation for this struct was generated from the following file:

- magma/network/**network.h**

composition_t Struct Reference

```
#include <sessions.h>
```

Data Fields

- uint64_t **comp_id**
 - uint64_t **attached**
 - inx_t * **attachments**
-

Detailed Description

Definition at line 28 of file sessions.h.

Field Documentation

uint64_t composition_t::attached

Definition at line 30 of file sessions.h.

Referenced by portal_endpoint_attachments_add().

inx_t* composition_t::attachments

Definition at line 31 of file sessions.h.

Referenced by portal_endpoint_attachments_add(), portal_endpoint_attachments_remove(), portal_endpoint_messages_compose(), portal_endpoint_messages_send(), portal_get_upload_attachment(), and sess_release_composition().

uint64_t composition_t::comp_id

Definition at line 29 of file sessions.h.

Referenced by portal_endpoint_messages_compose().

The documentation for this struct was generated from the following file:

- magma/network/sessions.h

connection_t Struct Reference

```
#include <network.h>
```

Data Fields

- union {
- pop_session_t **pop**
- imap_session_t **imap**
- **smtp_session_t smtp**
- http_session_t **http**
- };
- struct {
- uint32_t **spins**
- uint32_t **violations**
- } **protocol**
- struct {
- void * **ssl**
- int **sockd**
- int **status**
- **placer_t** **line**
- **stringer_t** * **buffer**
- struct {
- **int_t** **status**
- **stringer_t** * **domain**
- } **reverse**
- } **network**
- uint64_t **refs**
- pthread_mutex_t **lock**
- **server_t** * **server**
- **command_t** * **command**

Detailed Description

Definition at line 70 of file network.h.

Field Documentation

union { ... }

stringer_t* connection_t::buffer

Definition at line 88 of file network.h.

Referenced by `con_destroy()`, `con_init_network_buffer()`, `con_read()`, `con_read_line()`, `http_body()`, `imap_parse_literal()`, `imap_starttls()`, `pop_starttls()`, `protocol_secure()`, `smtp_data_finish()`, `smtp_data_read()`, and `smtp_starttls()`.

command_t* connection_t::command

Definition at line 99 of file network.h.

Referenced by `imap_process()`, `molten_parse()`, `pop_process()`, `smtp_disabled()`, `smtp_parse_helo_domain()`, `smtp_parse_mail_from_path()`, `smtp_parse_rcpt_to()`, and `smtp_process()`.

stringer_t* connection_t::domain

Definition at line 92 of file network.h.

http_session_t connection_t::http

Definition at line 75 of file network.h.

Referenced by `contact_process()`, `http_body()`, `http_data_get()`, `http_data_value_parse()`, `http_parse_context()`, `http_parse_header()`, `http_parse_method()`, `http_parse_pairs()`, `http_print_301()`, `http_print_400()`, `http_print_403()`, `http_print_404()`, `http_print_405()`, `http_print_500()`, `http_print_500_log()`, `http_print_501()`, `http_process()`, `http_requeue()`, `http_response()`, `http_response_connection()`, `http_response_cookie()`, `http_response_header()`, `http_response_options()`, `http_session_reset()`, `portal_endpoint()`, `portal_endpoint_alert_acknowledge()`, `portal_endpoint_alert_list()`, `portal_endpoint_aliases()`, `portal_endpoint_attachments_add()`, `portal_endpoint_attachments_progress()`, `portal_endpoint_attachments_remove()`, `portal_endpoint_auth()`, `portal_endpoint_config_edit()`, `portal_endpoint_config_load()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_contacts_edit()`, `portal_endpoint_contacts_list()`, `portal_endpoint_contacts_load()`, `portal_endpoint_contacts_move()`, `portal_endpoint_contacts_remove()`, `portal_endpoint_cookies()`, `portal_endpoint_error()`, `portal_endpoint_folders_add()`, `portal_endpoint_folders_list()`, `portal_endpoint_folders_remove()`, `portal_endpoint_folders_rename()`, `portal_endpoint_folders_tags()`, `portal_endpoint_logout()`, `portal_endpoint_messages_compose()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_load()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_send()`, `portal_endpoint_messages_tag()`, `portal_endpoint_messages_tags()`, `portal_endpoint_response()`, `portal_endpoint_search()`, `portal_folder_contacts_add()`, `portal_folder_contacts_remove()`, `portal_folder_mail_add()`, `portal_folder_mail_remove()`, `portal_get_upload_attachment()`, `portal_meta()`, `portal_settings_changepass()`, `portal_settings_identity()`, `portal_upload()`, `portal_validate_request()`, `register_process()`, `sess_create()`, `sess_get()`, `teacher_add_cookie()`, `teacher_print_message()`, and `teacher_process()`.

imap_session_t connection_t::imap

Definition at line 73 of file network.h.

Referenced by `imap_append()`, `imap_append_message()`, `imap_capability()`, `imap_check()`, `imap_close()`, `imap_command_parser()`, `imap_copy()`, `imap_create()`, `imap_delete()`, `imap_examine()`, `imap_expunge()`, `imap_fetch()`, `imap_fetch_message()`, `imap_fetch_return_header()`, `imap_fetch_return_message()`, `imap_fetch_return_mime()`, `imap_fetch_return_text()`, `imap_id()`, `imap_idle()`, `imap_invalid()`, `imap_list()`, `imap_login()`, `imap_logout()`, `imap_lsub()`, `imap_message_copier()`, `imap_message_expunge()`, `imap_noop()`, `imap_parse_arguments()`, `imap_process()`, `imap_rename()`, `imap_search()`, `imap_search_messages()`, `imap_select()`, `imap_session_destroy()`, `imap_session_update()`, `imap_starttls()`, `imap_status()`, `imap_store()`, `imap_subscribe()`, and `imap_unsubscribe()`.

placer_t connection_t::line

Definition at line 87 of file network.h.

Referenced by con_init_network_buffer(), con_read(), con_read_line(), http_data_header_parse(), http_parse_method(), http_process(), imap_command_parser(), imap_parse_literal(), imap_process(), imap_starttls(), molten_parse(), pop_num_parse(), pop_pass_parse(), pop_process(), pop_starttls(), pop_top_parse(), pop_user_parse(), smtp_auth_login(), smtp_auth_plain(), smtp_data_finish(), smtp_data_read(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(), smtp_parse_rcpt_to(), smtp_process(), and smtp_starttls().

pthread_mutex_t connection_t::lock

Definition at line 97 of file network.h.

Referenced by con_decrement_refs(), con_destroy(), con_increment_refs(), con_init(), con_reverse_check(), con_reverse_domain(), con_reverse_enqueue(), con_reverse_status(), and protocol_secure().

struct { ... } connection_t::network

Referenced by con_addr(), con_destroy(), con_init(), con_init_network_buffer(), con_print(), con_read(), con_read_line(), con_reverse_check(), con_reverse_domain(), con_reverse_enqueue(), con_reverse_lookup(), con_reverse_status(), con_secure(), con_status(), con_write_bl(), http_body(), http_data_header_parse(), http_parse_method(), http_process(), imap_command_parser(), imap_parse_literal(), imap_process(), imap_starttls(), molten_parse(), pop_num_parse(), pop_pass_parse(), pop_process(), pop_starttls(), pop_top_parse(), pop_user_parse(), protocol_secure(), smtp_auth_login(), smtp_auth_plain(), smtp_data_finish(), smtp_data_read(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(), smtp_parse_rcpt_to(), smtp_process(), and smtp_starttls().

pop_session_t connection_t::pop

Definition at line 72 of file network.h.

Referenced by pop_capa(), pop_dele(), pop_last(), pop_list(), pop_pass(), pop_process(), pop_retr(), pop_rset(), pop_session_destroy(), pop_session_reset(), pop_starttls(), pop_stat(), pop_top(), pop_uidl(), and pop_user().

struct { ... } connection_t::protocol

Referenced by http_process(), http_requeue(), imap_invalid(), imap_process(), imap_requeue(), pop_invalid(), pop_process(), pop_requeue(), smtp_disabled(), smtp_invalid(), smtp_process(), and smtp_requeue().

uint64_t connection_t::refs

Definition at line 96 of file network.h.

Referenced by con_decrement_refs(), con_increment_refs(), and con_reverse_enqueue().

struct { ... } connection_t::reverse

Referenced by con_destroy(), con_reverse_check(), con_reverse_domain(), con_reverse_enqueue(), and con_reverse_status().

server_t* connection_t::server

Definition at line 98 of file network.h.

Referenced by con_destroy(), con_init(), con_secure(), http_process(), http_requeue(), imap_fetch_message(), imap_fetch_return_message(), imap_fetch_return_mime(), imap_fetch_return_text(), imap_init(), imap_invalid(), imap_process(), imap_requeue(), imap_starttls(), mail_add_inbound_headers(), mail_add_outbound_headers(), pop_invalid(), pop_process(), pop_requeue(), pop_retr(), pop_starttls(), pop_top(), portal_endpoint_messages_load(), protocol_enqueue(), protocol_secure(), smtp_accept_message(), smtp_disabled(), smtp_ehlo(), smtp_helo(), smtp_init(), smtp_invalid(), smtp_process(), smtp_requeue(), and smtp_starttls().

smtp_session_t connection_t::smtp

Definition at line 74 of file network.h.

Referenced by mail_add_inbound_headers(), mail_add_outbound_headers(), mail_add_required_headers(), smtp_accept_message(), smtp_add_inbound(), smtp_add_outbound(), smtp_add_recipient(), smtp_auth_login(), smtp_auth_plain(), smtp_bounce(), smtp_check_duplicate_recipient(), smtp_check_greylist(), smtp_data(), smtp_data_inbound(), smtp_data_outbound(), smtp_data_read(), smtp_ehlo(), smtp_helo(), smtp_init(), smtp_mail_from(), smtp_parse_mail_from_path(), smtp_rcpt_to(), smtp_relay_message(), smtp_session_destroy(), smtp_session_reset(), smtp_update_receive_stats(), smtp_update_transmission_stats(), and submission_init().

int connection_t::sockd

Definition at line 85 of file network.h.

Referenced by con_addr(), con_destroy(), con_init(), con_print(), con_read(), con_read_line(), con_reverse_lookup(), con_status(), con_write_bl(), imap_starttls(), pop_starttls(), protocol_secure(), and smtp_starttls().

uint32_t connection_t::spins

Definition at line 79 of file network.h.

Referenced by http_process(), imap_process(), pop_process(), and smtp_process().

void* connection_t::ssl

Definition at line 84 of file network.h.

Referenced by con_destroy(), con_read(), con_read_line(), con_secure(), con_write_bl(), imap_starttls(), pop_starttls(), protocol_secure(), and smtp_starttls().

int_t connection_t::status

Definition at line 91 of file network.h.

int connection_t::status

Definition at line 86 of file network.h.

Referenced by `con_print()`, `con_read()`, `con_read_line()`, `con_status()`, `con_write_bl()`, `imap_starttls()`, `pop_starttls()`, and `smtp_starttls()`.

uint32_t connection_t::violations

Definition at line 80 of file `network.h`.

Referenced by `http_process()`, `http_requeue()`, `imap_invalid()`, `imap_process()`, `imap_requeue()`, `pop_invalid()`, `pop_process()`, `pop_requeue()`, `smtp_disabled()`, `smtp_invalid()`, `smtp_process()`, and `smtp_requeue()`.

The documentation for this struct was generated from the following file:

- `magma/network/network.h`

credential_t Struct Reference

```
#include <neue.h>
```

Data Fields

- **int_t** type
- union {
- struct {
- **stringer_t** * **address**
- **stringer_t** * **domain**
- } **mail**
- struct {
- **stringer_t** * **username**
- **stringer_t** * **domain**
- **stringer_t** * **password**
- **stringer_t** * **key**
- } **auth**
- };

TODO: Add usernum to structure and call the appropriate fetch function to lookup its value.

Detailed Description

Definition at line 21 of file neue.h.

Field Documentation

union { ... }

TODO: Add usernum to structure and call the appropriate fetch function to lookup its value.

stringer_t* credential_t::address

Definition at line 30 of file neue.h.

struct { ... } credential_t::auth

Referenced by `credential_alloc_auth()`, `credential_alloc_mail()`, `credential_free()`, `imap_login()`, `pop_pass()`, `portal_endpoint_auth()`, `register_data_insert_user()`, `smtp_auth_login()`, `smtp_auth_plain()`, `smtp_fetch_authorization()`, and `teacher_process()`.

stringer_t* credential_t::domain

Definition at line 31 of file neue.h.

stringer_t* credential_t::key

Definition at line 38 of file neue.h.

struct { ... } credential_t::mail

Referenced by credential_alloc_mail(), credential_free(), smtp_fetch_inbound(), and smtp_rcpt_to().

stringer_t* credential_t::password

Definition at line 37 of file neue.h.

int_t credential_t::type

Definition at line 23 of file neue.h.

Referenced by credential_alloc_auth(), credential_alloc_mail(), credential_free(), smtp_fetch_authorization(), and smtp_fetch_inbound().

stringer_t* credential_t::username

Definition at line 35 of file neue.h.

The documentation for this struct was generated from the following file:

- magma/objects/neue/**neue.h**

hashed_bucket_t Struct Reference

Data Fields

- void * **data**
 - multi_t **key**
 - struct hashed_bucket_t * **next**
-

Detailed Description

Definition at line 18 of file hashed.c.

Field Documentation

void* hashed_bucket_t::data

Definition at line 19 of file hashed.c.

Referenced by hashed_bucket_alloc(), hashed_bucket_find_key(), hashed_cursor_value_active(), hashed_cursor_value_next(), hashed_delete(), hashed_free(), and hashed_truncate().

multi_t hashed_bucket_t::key

Definition at line 20 of file hashed.c.

Referenced by hashed_bucket_alloc(), hashed_bucket_find_key(), hashed_cursor_key_active(), hashed_cursor_key_next(), hashed_delete(), hashed_free(), and hashed_truncate().

struct hashed_bucket_t* hashed_bucket_t::next [read]

Definition at line 21 of file hashed.c.

Referenced by hashed_bucket_add(), hashed_bucket_find_key(), hashed_cursor_active(), hashed_delete(), hashed_free(), and hashed_truncate().

The documentation for this struct was generated from the following file:

- magma/core/indexes/**hashed.c**

http_content_t Struct Reference

```
#include <http.h>
```

Data Fields

- **stringer_t * location**
 - **stringer_t * resource**
 - **stringer_t * type**
 - **struct http_content_t * next**
-

Detailed Description

Definition at line 46 of file http.h.

Field Documentation

stringer_t* http_content_t::location

Definition at line 47 of file http.h.

Referenced by http_free_content().

struct http_content_t* http_content_t::next [read]

Definition at line 48 of file http.h.

stringer_t * http_content_t::resource

Definition at line 47 of file http.h.

Referenced by http_free_content(), http_page_get(), http_response(), register_print_message(), register_print_step1(), register_print_step2(), and register_print_step3().

stringer_t * http_content_t::type

Definition at line 47 of file http.h.

Referenced by contact_print_form(), contact_print_message(), http_free_content(), http_get_static(), http_get_template(), http_response(), portal_print_login(), register_print_message(), register_print_step1(), register_print_step2(), register_print_step3(), teacher_print_form(), and teacher_print_message().

The documentation for this struct was generated from the following file:

- magma/network/http.h

http_data_t Struct Reference

```
#include <http.h>
```

Data Fields

- **HTTP_DATA** source
 - **stringer_t** * name
 - **stringer_t** * value
-

Detailed Description

Definition at line 41 of file http.h.

Field Documentation

stringer_t* http_data_t::name

Definition at line 43 of file http.h.

Referenced by http_data_free(), http_data_get(), http_data_header_parse_line(), http_data_value_parse(), http_parse_header(), and portal_upload().

HTTP_DATA http_data_t::source

Definition at line 42 of file http.h.

Referenced by http_data_get(), http_data_header_parse_line(), and http_data_value_parse().

stringer_t * http_data_t::value

Definition at line 43 of file http.h.

Referenced by contact_business(), contact_print_form(), http_body(), http_data_free(), http_data_header_parse_line(), http_data_value_parse(), http_parse_header(), http_response_allow_cross(), multipart_get_boundary(), portal_upload(), register_business_step1(), register_business_step2(), register_process(), and teacher_process().

The documentation for this struct was generated from the following file:

- magma/network/http.h

http_page_t Struct Reference

```
#include <http.h>
```

Data Fields

- xmlDocPtr **doc_obj**
 - **http_content_t** * **content**
 - xmlParserCtxtPtr **doc_ctx**
 - xmlXPathContextPtr **xpath_ctx**
-

Detailed Description

Definition at line 51 of file http.h.

Field Documentation

http_content_t* http_page_t::content

Definition at line 53 of file http.h.

Referenced by `contact_print_form()`, `contact_print_message()`, `http_page_get()`, `portal_print_login()`, `teacher_print_form()`, and `teacher_print_message()`.

xmlParserCtxtPtr http_page_t::doc_ctx

Definition at line 54 of file http.h.

Referenced by `http_page_free()`, and `http_page_get()`.

xmlDocPtr http_page_t::doc_obj

Definition at line 52 of file http.h.

Referenced by `contact_print_form()`, `contact_print_message()`, `http_page_free()`, `http_page_get()`, `portal_print_login()`, `teacher_print_form()`, and `teacher_print_message()`.

xmlXPathContextPtr http_page_t::xpath_ctx

Definition at line 55 of file http.h.

Referenced by `contact_business_add_error()`, `contact_print_form()`, `contact_print_message()`, `http_page_free()`, `http_page_get()`, `portal_print_login()`, `teacher_add_error()`, `teacher_print_form()`, and `teacher_print_message()`.

The documentation for this struct was generated from the following file:

- `magma/network/http.h`

imap_fetch_dataitems_t Struct Reference

```
#include <imap.h>
```

Data Fields

- `int_t uid`
- `int_t flags`
- `int_t internaldate`
- `int_t envelope`
- `int_t bodystructure`
- `int_t rfc822`
- `int_t rfc822_header`
- `int_t rfc822_size`
- `int_t rfc822_text`
- `int_t body`
- `array_t * peek`
- `array_t * peek_partial`
- `array_t * normal`
- `array_t * normal_partial`

Detailed Description

Definition at line 23 of file `imap.h`.

Field Documentation

`int_t imap_fetch_dataitems_t::body`

Definition at line 24 of file `imap.h`.

Referenced by `imap_fetch_message()`, and `imap_parse_dataitems()`.

`int_t imap_fetch_dataitems_t::bodystructure`

Definition at line 24 of file `imap.h`.

Referenced by `imap_fetch_message()`, and `imap_parse_dataitems()`.

`int_t imap_fetch_dataitems_t::envelope`

Definition at line 24 of file `imap.h`.

Referenced by `imap_fetch_message()`, and `imap_parse_dataitems()`.

`int_t imap_fetch_dataitems_t::flags`

Definition at line 24 of file `imap.h`.

Referenced by `imap_fetch_message()`, and `imap_parse_dataitems()`.

`int_t imap_fetch_dataitems_t::internaldate`

Definition at line 24 of file `imap.h`.

Referenced by `imap_fetch_message()`, and `imap_parse_dataitems()`.

`array_t * imap_fetch_dataitems_t::normal`

Definition at line 25 of file `imap.h`.

Referenced by `imap_fetch()`, `imap_fetch_free_items()`, `imap_fetch_message()`, and `imap_parse_dataitems()`.

`array_t * imap_fetch_dataitems_t::normal_partial`

Definition at line 25 of file `imap.h`.

Referenced by `imap_fetch_free_items()`, `imap_fetch_message()`, and `imap_parse_dataitems()`.

`array_t* imap_fetch_dataitems_t::peek`

Definition at line 25 of file `imap.h`.

Referenced by `imap_fetch_free_items()`, `imap_fetch_message()`, and `imap_parse_dataitems()`.

`array_t * imap_fetch_dataitems_t::peek_partial`

Definition at line 25 of file `imap.h`.

Referenced by `imap_fetch_free_items()`, `imap_fetch_message()`, and `imap_parse_dataitems()`.

`int_t imap_fetch_dataitems_t::rfc822`

Definition at line 24 of file `imap.h`.

Referenced by `imap_fetch()`, `imap_fetch_message()`, and `imap_parse_dataitems()`.

`int_t imap_fetch_dataitems_t::rfc822_header`

Definition at line 24 of file `imap.h`.

Referenced by `imap_fetch()`, `imap_fetch_message()`, and `imap_parse_dataitems()`.

`int_t imap_fetch_dataitems_t::rfc822_size`

Definition at line 24 of file `imap.h`.

Referenced by `imap_fetch_message()`, and `imap_parse_dataitems()`.

int_t imap_fetch_dataitems_t::rfc822_text

Definition at line 24 of file imap.h.

Referenced by `imap_fetch()`, `imap_fetch_message()`, and `imap_parse_dataitems()`.

int_t imap_fetch_dataitems_t::uid

Definition at line 24 of file imap.h.

Referenced by `imap_fetch_message()`, and `imap_parse_dataitems()`.

The documentation for this struct was generated from the following file:

- `magma/network/imap.h`

imap_fetch_response_t Struct Reference

```
#include <imap.h>
```

Data Fields

- `stringer_t * key`
 - `stringer_t * value`
 - `struct imap_fetch_response_t * next`
-

Detailed Description

Definition at line 28 of file `imap.h`.

Field Documentation

`stringer_t* imap_fetch_response_t::key`

Definition at line 29 of file `imap.h`.

Referenced by `imap_fetch()`, `imap_fetch_response_add()`, `imap_fetch_response_free()`, `smtp_check_greylist()`, and `smtp_store_spamsig()`.

`struct imap_fetch_response_t* imap_fetch_response_t::next [read]`

Definition at line 30 of file `imap.h`.

Referenced by `imap_fetch()`, `imap_fetch_response_add()`, and `imap_fetch_response_free()`.

`stringer_t * imap_fetch_response_t::value`

Definition at line 29 of file `imap.h`.

Referenced by `imap_fetch()`, `imap_fetch_response_add()`, `imap_fetch_response_free()`, `imap_parse_address()`, and `smtp_check_greylist()`.

The documentation for this struct was generated from the following file:

- `magma/network/imap.h`

imap_folder_status_t Struct Reference

```
#include <imap.h>
```

Data Fields

- uint64_t **foldernum**
 - uint64_t **recent**
 - uint64_t **unseen**
 - uint64_t **uidnext**
 - uint64_t **messages**
 - uint64_t **first**
-

Detailed Description

Definition at line 19 of file imap.h.

Field Documentation

uint64_t imap_folder_status_t::first

Definition at line 20 of file imap.h.

Referenced by `imap_examine()`, `imap_folder_status()`, and `imap_select()`.

uint64_t imap_folder_status_t::foldernum

Definition at line 20 of file imap.h.

Referenced by `imap_examine()`, `imap_folder_status()`, `imap_select()`, and `imap_status()`.

uint64_t imap_folder_status_t::messages

Definition at line 20 of file imap.h.

Referenced by `imap_examine()`, `imap_folder_status()`, `imap_select()`, and `imap_status()`.

uint64_t imap_folder_status_t::recent

Definition at line 20 of file imap.h.

Referenced by `imap_examine()`, `imap_folder_status()`, `imap_select()`, and `imap_status()`.

uint64_t imap_folder_status_t::uidnext

Definition at line 20 of file imap.h.

Referenced by `imap_examine()`, `imap_folder_status()`, `imap_select()`, and `imap_status()`.

uint64_t imap_folder_status_t::unseen

Definition at line 20 of file `imap.h`.

Referenced by `imap_folder_status()`, and `imap_status()`.

The documentation for this struct was generated from the following file:

- `magma/network/imap.h`

inx_t Struct Reference

```
#include <indexes.h>
```

Data Fields

- void * **index**
- pthread_rwlock_t **lock**
- uint64_t **count**
- uint64_t **serial**
- uint64_t **automatic**
- uint64_t **options**
- uint64_t **references**
- void(* **data_free**)(void ***data**)
- void(* **index_free**)(void ***index**)
- void(* **index_truncate**)(void ***index**)
- bool_t(* **delete**)(void ***index**, multi_t envelope)
- bool_t(* **insert**)(void ***index**, multi_t envelope, void ***data**)
- void(* **find**)(void ***index**, multi_t envelope)
- void(* **cursor_free**)(void *cursor)
- void(* **cursor_reset**)(void *cursor)
- void(* **cursor_alloc**)(void ***index**)
- void(* **cursor_value_next**)(void *cursor)
- void(* **cursor_value_active**)(void *cursor)
- multi_t(* **cursor_key_next**)(void *cursor)
- multi_t(* **cursor_key_active**)(void *cursor)

Detailed Description

Definition at line 39 of file indexes.h.

Field Documentation

uint64_t inx_t::automatic

Definition at line 44 of file indexes.h.

Referenced by inx_alloc(), inx_auto_read(), inx_auto_unlock(), inx_auto_write(), inx_lock_read(), inx_lock_write(), and inx_unlock().

uint64_t inx_t::count

Definition at line 44 of file indexes.h.

Referenced by hashed_delete(), hashed_insert(), inx_count(), inx_serial(), linked_delete(), linked_find(), linked_insert(), tree_delete(), and tree_insert().

void(* inx_t::cursor_alloc)(void *index)

Referenced by hashed_alloc(), inx_cursor_alloc(), linked_alloc(), and tree_alloc().

void(* inx_t::cursor_free)(void *cursor)

Referenced by hashed_alloc(), linked_alloc(), and tree_alloc().

multi_t(* inx_t::cursor_key_active)(void *cursor)

Referenced by hashed_alloc(), linked_alloc(), and tree_alloc().

multi_t(* inx_t::cursor_key_next)(void *cursor)

Referenced by hashed_alloc(), linked_alloc(), and tree_alloc().

void(* inx_t::cursor_reset)(void *cursor)

Referenced by hashed_alloc(), linked_alloc(), and tree_alloc().

void*(* inx_t::cursor_value_active)(void *cursor)

Referenced by hashed_alloc(), linked_alloc(), and tree_alloc().

void*(* inx_t::cursor_value_next)(void *cursor)

Referenced by hashed_alloc(), linked_alloc(), and tree_alloc().

void(* inx_t::data_free)(void *data)

Referenced by hashed_alloc(), hashed_delete(), hashed_free(), hashed_truncate(), linked_alloc(), linked_record_free(), tree_alloc(), tree_delete(), and tree_truncate().

bool_t(* inx_t::delete)(void *index, multi_t envelope)

Referenced by hashed_alloc(), inx_delete(), inx_replace(), linked_alloc(), and tree_alloc().

void*(* inx_t::find)(void *index, multi_t envelope)

Referenced by hashed_alloc(), inx_find(), linked_alloc(), and tree_alloc().

void* inx_t::index

Definition at line 42 of file indexes.h.

Referenced by hashed_alloc(), hashed_delete(), hashed_find(), hashed_free(), hashed_insert(), hashed_truncate(), linked_delete(), linked_find(), linked_free(), linked_insert(), linked_truncate(), tree_alloc(), tree_count(), tree_cursor_alloc(), tree_delete(), tree_free(), tree_insert(), and tree_truncate().

void(* inx_t::index_free)(void *index)

Referenced by hashed_alloc(), inx_free(), linked_alloc(), and tree_alloc().

void(* inx_t::index_truncate)(void *index)

Referenced by hashed_alloc(), inx_truncate(), linked_alloc(), and tree_alloc().

bool_t(* inx_t::insert)(void *index, multi_t envelope, void *data)

Referenced by hashed_alloc(), inx_insert(), inx_replace(), linked_alloc(), and tree_alloc().

pthread_rwlock_t inx_t::lock

Definition at line 43 of file indexes.h.

Referenced by inx_alloc(), inx_auto_read(), inx_auto_unlock(), inx_auto_write(), inx_free(), inx_lock_read(), inx_lock_write(), and inx_unlock().

uint64_t inx_t::options

Definition at line 44 of file indexes.h.

Referenced by hashed_alloc(), inx_options(), linked_alloc(), and tree_alloc().

uint64_t inx_t::references

Definition at line 44 of file indexes.h.

Referenced by inx_alloc(), inx_cursor_alloc(), and inx_free().

uint64_t inx_t::serial

Definition at line 44 of file indexes.h.

Referenced by hashed_delete(), hashed_insert(), linked_delete(), linked_insert(), tree_delete(), and tree_insert().

The documentation for this struct was generated from the following file:

- magma/core/indexes/indexes.h

ip_t Struct Reference

```
#include <network.h>
```

Data Fields

- `sa_family_t family`
- `union {`
- `struct in_addr ip4`
- `struct in6_addr ip6`
- `void * ip`
- `};`

Detailed Description

Definition at line 15 of file network.h.

Field Documentation

union { ... }

sa_family_t ip_t::family

Definition at line 16 of file network.h.

Referenced by `con_addr()`, `ip_address_equal()`, `ip_matches_subnet()`, `ip_octet()`, `ip_presentation()`, `ip_reversed()`, `ip_segment()`, `ip_standard()`, `ip_str_addr()`, `ip_str_subnet()`, `ip_subnet()`, `ip_word()`, `signal_shutdown()`, and `spf_check()`.

void* ip_t::ip

Definition at line 20 of file network.h.

Referenced by `ip_matches_subnet()`.

struct in_addr ip_t::ip4 [read]

Definition at line 18 of file network.h.

Referenced by `con_addr()`, `ip_address_equal()`, `ip_octet()`, `ip_presentation()`, `ip_reversed()`, `ip_segment()`, `ip_standard()`, `ip_str_addr()`, `ip_subnet()`, `ip_word()`, `signal_shutdown()`, and `spf_check()`.

struct in6_addr ip_t::ip6 [read]

Definition at line 19 of file network.h.

Referenced by `con_addr()`, `ip_address_equal()`, `ip_octet()`, `ip_presentation()`, `ip_reversed()`, `ip_segment()`, `ip_standard()`, `ip_str_addr()`, `ip_subnet()`, `ip_word()`, `signal_shutdown()`, and `spf_check()`.

The documentation for this struct was generated from the following file:

- magma/network/**network.h**

linked_node_t Struct Reference

Data Fields

- `linked_record_t * record`
 - `struct linked_node_t * next`
 - `struct linked_node_t * prev`
-

Detailed Description

Definition at line 22 of file `linked.c`.

Field Documentation

`struct linked_node_t * linked_node_t::next` `[read]`

Definition at line 24 of file `linked.c`.

Referenced by `linked_cursor_active()`, `linked_cursor_next()`, `linked_delete()`, `linked_find()`, `linked_insert()`, and `linked_truncate()`.

`struct linked_node_t * linked_node_t::prev` `[read]`

Definition at line 24 of file `linked.c`.

Referenced by `linked_delete()`, and `linked_insert()`.

`linked_record_t * linked_node_t::record`

Definition at line 23 of file `linked.c`.

Referenced by `linked_cursor_key_active()`, `linked_cursor_key_next()`, `linked_cursor_value_active()`, `linked_cursor_value_next()`, `linked_delete()`, `linked_find()`, `linked_insert()`, and `linked_truncate()`.

The documentation for this struct was generated from the following file:

- `magma/core/indexes/linked.c`

linked_record_t Struct Reference

Data Fields

- **multi_t** **key**
 - **void **** **data**
 - **uint64_t** **size**
 - **uint64_t** **count**
-

Detailed Description

Definition at line 16 of file linked.c.

Field Documentation

uint64_t linked_record_t::count

Definition at line 19 of file linked.c.

Referenced by linked_record_alloc().

void** linked_record_t::data

Definition at line 18 of file linked.c.

Referenced by linked_record_alloc(), and linked_record_free().

multi_t linked_record_t::key

Definition at line 17 of file linked.c.

Referenced by linked_delete(), linked_find(), linked_record_alloc(), linked_record_free(), and linked_record_get_key().

uint64_t linked_record_t::size

Definition at line 19 of file linked.c.

The documentation for this struct was generated from the following file:

- magma/core/indexes/**linked.c**

magma_keys_t Struct Reference

```
#include <global.h>
```

Data Fields

- void * **store**
- multi_t **norm**
- chr_t * **name**
- chr_t * **description**
- bool_t **file**
- bool_t **database**
- bool_t **overwrite**
- bool_t **set**
- bool_t **required**

Detailed Description

Definition at line 16 of file global.h.

Field Documentation

bool_t magma_keys_t::database

Definition at line 22 of file global.h.

Referenced by config_load_database_settings().

chr_t* magma_keys_t::description

Definition at line 20 of file global.h.

bool_t magma_keys_t::file

Definition at line 21 of file global.h.

Referenced by config_load_cmdline_settings(), and config_load_file_settings().

chr_t* magma_keys_t::name

Definition at line 19 of file global.h.

Referenced by config_load_cmdline_settings(), config_load_database_settings(), config_load_file_settings(), config_output_value(), and config_value_set().

multi_t magma_keys_t::norm

Definition at line 18 of file global.h.

Referenced by config_free(), config_output_help(), config_output_value(), and config_value_set().

bool_t magma_keys_t::overwrite

Definition at line 23 of file global.h.

Referenced by config_load_database_settings().

bool_t magma_keys_t::required

Definition at line 25 of file global.h.

Referenced by config_load_cmdline_settings(), config_load_database_settings(), config_load_file_settings(), and config_output_help().

bool_t magma_keys_t::set

Definition at line 24 of file global.h.

Referenced by config_load_cmdline_settings(), config_load_database_settings(), config_load_file_settings(), and config_validate_settings().

void* magma_keys_t::store

Definition at line 17 of file global.h.

Referenced by config_output_value(), and config_value_set().

The documentation for this struct was generated from the following file:

- magma/engine/config/global/global.h

magma_t Struct Reference

```
#include <global.h>
```

Data Fields

- struct {
- **bool_t** output_config
- **bool_t** output_resource_limits
- **chr_t** file [MAGMA_FILEPATH_MAX+1]
- } config
- struct {
- **stringer_t** * contact
- **stringer_t** * abuse
- } admin
- struct {
- **uint64_t** number
- **chr_t** name [MAGMA_HOSTNAME_MAX+1]
- } host
- struct {
- **chr_t** * file
- **bool_t** unload
- } library
- struct {
- **bool_t** daemonize
- **char** * root_directory
- **char** * impersonate_user
- **bool_t** increase_resource_limits
- **uint32_t** thread_stack_size
- **uint32_t** worker_threads
- **uint32_t** network_buffer
- **bool_t** enable_core_dumps
- **uint64_t** core_dump_size_limit
- **stringer_t** * domain
- } system
- struct {
- struct {
- **bool_t** enable
- **uint64_t** length
- } memory
- **stringer_t** * salt
- **stringer_t** * links
- **stringer_t** * sessions
- } secure
- struct {
- **bool_t** file
- **chr_t** * path
- } output
- struct {
- **bool_t** imap
- **bool_t** http

- **bool_t** content
- **bool_t** file
- **bool_t** line
- **bool_t** time
- **bool_t** stack
- **bool_t** function
- **} log**
- **struct {**
- **chr_t * tank**
- **stringer_t * active**
- **stringer_t * root**
- **} storage**
- **struct {**
- **uint32_t relay_limit**
- **uint32_t recipient_limit**
- **uint32_t address_length_limit**
- **uint32_t helo_length_limit**
- **uint32_t wrap_line_length**
- **uint64_t message_length_limit**
- **struct {**
- **uint32_t count**
- **stringer_t * domain [MAGMA_BLACKLIST_INSTANCES]**
- **} blacklists**
- **stringer_t * bypass_addr**
- **inx_t * bypass_subnets**
- **} smtp**
- **struct {**
- **bool_t close**
- **bool_t allow_cross_domain**
- **chr_t * fonts**
- **chr_t * pages**
- **chr_t * templates**
- **uint32_t session_timeout**
- **} http**
- **struct {**
- **relay_t * host [MAGMA_RELAY_INSTANCES]**
- **struct {**
- **uint32_t premium**
- **uint32_t standard**
- **} count**
- **uint32_t timeout**
- **} relay**
- **struct {**
- **struct {**
- **bool_t indent**
- **bool_t safeguard**
- **} portal**
- **struct {**
- **stringer_t * sender**
- **} contact**
- **bool_t statistics**
- **bool_t registration**
- **stringer_t * ssl_redirect**

- } web
- struct {
- **bool_t enabled**
- **chr_t * domain**
- **chr_t * selector**
- **stringer_t * privkey**
- } **dkim**
- struct {
- struct {
- **uint32_t seed_length**
- } **cryptography**
- struct {
- **chr_t * host**
- **chr_t * user**
- **chr_t * password**
- **chr_t * schema**
- **uint32_t port**
- **chr_t * socket_path**
- struct {
- **uint32_t timeout**
- **uint32_t connections**
- } **pool**
- } **database**
- struct {
- **bool_t available**
- **char * signatures**
- } **virus**
- struct {
- **chr_t * path**
- **stringer_t * cache**
- } **location**
- struct {
- **cache_t * host** [MAGMA_CACHE_INSTANCES]
- struct {
- **uint32_t timeout**
- **uint32_t connections**
- } **pool**
- **uint32_t retry**
- **uint32_t timeout**
- } **cache**
- struct {
- struct {
- **uint32_t timeout**
- **uint32_t connections**
- } **pool**
- } **spf**
- } **iface**
- **chr_t * spool**
- **int_t page_length**
- **uint32_t init**
- **pthread_rwlock_t lock**
- **server_t * servers** [MAGMA_SERVER_INSTANCES]

Detailed Description

Definition at line 28 of file global.h.

Field Documentation

stringer_t* magma_t::abuse

Definition at line 40 of file global.h.

Referenced by config_validate_settings(), contact_business(), http_response(), smtp_accept_message(), and smtp_rcpt_to().

stringer_t* magma_t::active

Definition at line 99 of file global.h.

Referenced by imap_append_message(), mail_create_directory(), mail_db_insert_duplicate_message(), mail_db_insert_message(), and mail_message_path().

uint32_t magma_t::address_length_limit

Definition at line 106 of file global.h.

Referenced by smtp_parse_mail_from_path(), and smtp_parse_rcpt_to().

struct { ... } magma_t::admin

Referenced by config_validate_settings(), contact_business(), http_response(), register_business_step2(), smtp_accept_message(), and smtp_rcpt_to().

bool_t magma_t::allow_cross_domain

Definition at line 123 of file global.h.

bool_t magma_t::available

Definition at line 180 of file global.h.

struct { ... } magma_t::blacklists

Referenced by config_free(), config_output_value(), config_output_value_generic(), config_value_set(), and smtp_check_rbl().

stringer_t* magma_t::bypass_addr

Definition at line 117 of file global.h.

inx_t* magma_t::bypass_subnets

Definition at line 118 of file global.h.

Referenced by smtp_add_bypass_entry(), and smtp_bypass_check().

struct { ... } magma_t::cache

stringer_t* magma_t::cache

Definition at line 186 of file global.h.

Referenced by cache_alloc(), cache_free(), cache_output_settings(), cache_start(), cache_stop(), cache_validate(), and config_validate_settings().

bool_t magma_t::close

Definition at line 122 of file global.h.

struct { ... } magma_t::config

Referenced by args_parse(), config_load_file_settings(), config_output_settings(), config_validate_settings(), lib_load(), and system_init_resource_limits().

uint32_t magma_t::connections

Definition at line 175 of file global.h.

struct { ... } magma_t::contact

stringer_t* magma_t::contact

Definition at line 39 of file global.h.

Referenced by config_validate_settings(), contact_business(), http_response(), register_business_step2(), smtp_accept_message(), and smtp_rcpt_to().

bool_t magma_t::content

Definition at line 88 of file global.h.

Referenced by http_content_load_fonts(), and http_load_file().

uint64_t magma_t::core_dump_size_limit

Definition at line 63 of file global.h.

Referenced by system_init_core_dumps().

struct { ... } magma_t::count

uint32_t magma_t::count

Definition at line 113 of file global.h.

Referenced by relay_counter(), and smtp_client_connect().

struct { ... } magma_t::cryptography

Referenced by rand_start(), and rand_thread_start().

bool_t magma_t::daemonize

Definition at line 54 of file global.h.

Referenced by config_validate_settings(), and system_fork_daemon().

struct { ... } magma_t::database

Referenced by sql_open(), sql_start(), sql_stop(), stmt_start(), and stmt_stop().

struct { ... } magma_t::dkim

Referenced by config_validate_settings(), dkim_create(), mail_add_forward_headers(), mail_add_outbound_headers(), and ssl_start().

chr_t* magma_t::domain

Definition at line 154 of file global.h.

stringer_t* magma_t::domain[MAGMA_BLACKLIST_INSTANCES]

Definition at line 65 of file global.h.

Referenced by config_validate_settings(), credential_alloc_auth(), credential_username(), dkim_create(), register_business_step2(), register_data_insert_user(), and smtp_bounce().

bool_t magma_t::enable

Definition at line 70 of file global.h.

bool_t magma_t::enable_core_dumps

Definition at line 62 of file global.h.

Referenced by system_init_core_dumps().

bool_t magma_t::enabled

Definition at line 153 of file global.h.

Referenced by config_validate_settings(), dkim_create(), mail_add_forward_headers(), mail_add_outbound_headers(), and ssl_start().

bool_t magma_t::file

Definition at line 80 of file global.h.

chr_t* magma_t::file

Definition at line 49 of file global.h.

chr_t magma_t::file[MAGMA_FILEPATH_MAX+1]

Definition at line 35 of file global.h.

Referenced by args_parse(), config_load_file_settings(), config_output_settings(), config_validate_settings(), lib_load(), log_internal(), log_rotate(), and log_start().

chr_t* magma_t::fonts

Definition at line 124 of file global.h.

bool_t magma_t::function

Definition at line 94 of file global.h.

Referenced by log_internal().

uint32_t magma_t::helo_length_limit

Definition at line 107 of file global.h.

Referenced by smtp_parse_helo_domain().

cache_t* magma_t::host[MAGMA_CACHE_INSTANCES]

Definition at line 190 of file global.h.

chr_t* magma_t::host

Definition at line 166 of file global.h.

relay_t* magma_t::host[MAGMA_RELAY_INSTANCES]

Definition at line 131 of file global.h.

struct { ... } magma_t::host

Referenced by config_fetch_host_number(), config_fetch_settings(), config_load_database_settings(), config_output_settings(), process_start(), relay_alloc(), relay_counter(), relay_free(), relay_output_settings(), relay_validate(), servers_network_start(), sess_create(), smtp_client_connect(), and smtp_client_send_helo().

struct { ... } magma_t::http

bool_t magma_t::http

Definition at line 87 of file global.h.

Referenced by config_validate_settings(), http_body(), http_content_load_fonts(), http_content_refresh(), http_content_start(), http_data_value_parse(), http_load_file(), http_parse_header(), http_parse_method(), http_response_connection(), http_response_header(), http_response_options(), register_captcha_random_font(), and sess_get().

struct { ... } magma_t::iface

Referenced by cache_alloc(), cache_free(), cache_output_settings(), cache_start(), cache_stop(), cache_validate(), config_validate_settings(), rand_start(), rand_thread_start(), spf_start(), spf_stop(), sql_open(), sql_start(), sql_stop(), stmt_start(), stmt_stop(), virus_check(), virus_engine_create(), virus_engine_refresh(), virus_sigs_total(), virus_start(), and virus_stop().

bool_t magma_t::imap

Definition at line 86 of file global.h.

Referenced by imap_command_parser().

char* magma_t::impersonate_user

Definition at line 56 of file global.h.

Referenced by system_init_impersonation().

bool_t magma_t::increase_resource_limits

Definition at line 57 of file global.h.

Referenced by system_init_resource_limits().

bool_t magma_t::indent

Definition at line 141 of file global.h.

uint32_t magma_t::init

Definition at line 213 of file global.h.

Referenced by process_start(), and process_stop().

uint64_t magma_t::length

Definition at line 71 of file global.h.

struct { ... } magma_t::library

Referenced by config_validate_settings(), lib_load(), and lib_unload().

bool_t magma_t::line

Definition at line 91 of file global.h.

Referenced by log_internal().

stringer_t* magma_t::links

Definition at line 75 of file global.h.

struct { ... } magma_t::location

pthread_rwlock_t magma_t::lock

Definition at line 214 of file global.h.

struct { ... } magma_t::log

Referenced by http_body(), http_content_load_fonts(), http_data_value_parse(), http_load_file(), http_parse_header(), http_parse_method(), imap_command_parser(), and log_internal().

struct { ... } magma_t::memory

Referenced by mm_sec_start(), and st_realloc().

uint64_t magma_t::message_length_limit

Definition at line 109 of file global.h.

Referenced by smtp_ehlo(), and smtp_mail_from().

chr_t magma_t::name[MAGMA_HOSTNAME_MAX+1]

Definition at line 45 of file global.h.

Referenced by config_fetch_host_number(), config_fetch_settings(), config_output_settings(), process_start(), servers_network_start(), and smtp_client_send_helo().

uint32_t magma_t::network_buffer

Definition at line 60 of file global.h.

Referenced by con_init_network_buffer(), and net_init().

uint64_t magma_t::number

Definition at line 44 of file global.h.

Referenced by config_load_database_settings(), config_output_settings(), and sess_create().

struct { ... } magma_t::output

Referenced by config_validate_settings(), log_rotate(), and log_start().

bool_t magma_t::output_config

Definition at line 31 of file global.h.

Referenced by config_validate_settings().

bool_t magma_t::output_resource_limits

Definition at line 32 of file global.h.

Referenced by system_init_resource_limits().

int_t magma_t::page_length

Definition at line 210 of file global.h.

Referenced by mm_sec_start(), process_start(), st_alloc_opts(), and st_realloc().

chr_t* magma_t::pages

Definition at line 125 of file global.h.

chr_t* magma_t::password

Definition at line 168 of file global.h.

chr_t* magma_t::path

Definition at line 81 of file global.h.

Referenced by config_validate_settings(), log_rotate(), and log_start().

struct { ... } ::@40 magma_t::pool

struct { ... } ::@39 magma_t::pool

struct { ... } ::@38 magma_t::pool

uint32_t magma_t::port

Definition at line 170 of file global.h.

struct { ... } magma_t::portal

Referenced by portal_endpoint(), portal_endpoint_error(), portal_endpoint_response(), portal_process(), and portal_upload().

uint32_t magma_t::premium

Definition at line 133 of file global.h.

Referenced by relay_counter(), and smtp_client_connect().

stringer_t* magma_t::privkey

Definition at line 156 of file global.h.

Referenced by config_validate_settings(), dkim_create(), and ssl_start().

uint32_t magma_t::recipient_limit

Definition at line 105 of file global.h.

Referenced by config_validate_settings(), and smtp_rcpt_to().

bool_t magma_t::registration

Definition at line 148 of file global.h.

Referenced by http_response().

struct { ... } magma_t::relay

Referenced by relay_alloc(), relay_counter(), relay_free(), relay_output_settings(), relay_validate(), and smtp_client_connect().

uint32_t magma_t::relay_limit

Definition at line 104 of file global.h.

Referenced by config_validate_settings(), and smtp_data().

uint32_t magma_t::retry

Definition at line 195 of file global.h.

stringer_t* magma_t::root

Definition at line 100 of file global.h.

Referenced by mail_create_directory(), and mail_message_path().

char* magma_t::root_directory

Definition at line 55 of file global.h.

Referenced by config_validate_settings(), and system_change_root_directory().

bool_t magma_t::safeguard

Definition at line 142 of file global.h.

stringer_t* magma_t::salt

Definition at line 74 of file global.h.

Referenced by credential_alloc_auth().

chr_t* magma_t::schema

Definition at line 169 of file global.h.

struct { ... } magma_t::secure

Referenced by credential_alloc_auth(), mm_sec_start(), sess_get(), sess_token(), and st_realloc().

uint32_t magma_t::seed_length

Definition at line 162 of file global.h.

chr_t* magma_t::selector

Definition at line 155 of file global.h.

Referenced by config_validate_settings(), and dkim_create().

stringer_t* magma_t::sender

Definition at line 145 of file global.h.

server_t* magma_t::servers[MAGMA_SERVER_INSTANCES]

Definition at line 215 of file global.h.

Referenced by net_listen(), servers_alloc(), servers_encryption_start(), servers_encryption_stop(), servers_free(), servers_get_by_socket(), servers_get_count_using_port(), servers_network_start(), servers_network_stop(), servers_output_settings(), and servers_validate().

uint32_t magma_t::session_timeout

Definition at line 127 of file global.h.

stringer_t* magma_t::sessions

Definition at line 76 of file global.h.

Referenced by sess_get(), and sess_token().

char* magma_t::signatures

Definition at line 181 of file global.h.

struct { ... } magma_t::smtp

Referenced by config_free(), config_output_value(), config_output_value_generic(), config_validate_settings(), config_value_set(), mail_message_cleanup(), smtp_add_bypass_entry(), smtp_bypass_check(), smtp_check_rbl(), smtp_data(), smtp_ehlo(), smtp_mail_from(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(), smtp_parse_rcpt_to(), and smtp_rcpt_to().

chr_t* magma_t::socket_path

Definition at line 171 of file global.h.

struct { ... } magma_t::spf

Referenced by spf_start(), and spf_stop().

chr_t* magma_t::spool

Definition at line 209 of file global.h.

Referenced by config_validate_settings(), spool_path(), and virus_start().

stringer_t* magma_t::ssl_redirect

Definition at line 149 of file global.h.

Referenced by http_print_301().

bool_t magma_t::stack

Definition at line 93 of file global.h.

Referenced by log_internal().

uint32_t magma_t::standard

Definition at line 134 of file global.h.

bool_t magma_t::statistics

Definition at line 147 of file global.h.

Referenced by http_response().

struct { ... } magma_t::storage

Referenced by imap_append_message(), mail_create_directory(), mail_db_insert_duplicate_message(), mail_db_insert_message(), mail_message_path(), and tank_start().

struct { ... } magma_t::system

Referenced by con_init_network_buffer(), config_validate_settings(), credential_alloc_auth(), credential_username(), net_init(), queue_init(), queue_shutdown(), queue_signal(), register_business_step2(), register_data_insert_user(), smtp_bounce(), system_change_root_directory(), system_fork_daemon(), system_init_core_dumps(), system_init_impersonation(), system_init_resource_limits(), and thread_launch().

chr_t* magma_t::tank

Definition at line 98 of file global.h.

Referenced by tank_start().

chr_t* magma_t::templates

Definition at line 126 of file global.h.

uint32_t magma_t::thread_stack_size

Definition at line 58 of file global.h.

Referenced by config_validate_settings(), and thread_launch().

bool_t magma_t::time

Definition at line 92 of file global.h.

Referenced by log_internal().

uint32_t magma_t::timeout

Definition at line 136 of file global.h.

Referenced by smtp_client_connect().

bool_t magma_t::unload

Definition at line 50 of file global.h.

Referenced by lib_unload().

chr_t* magma_t::user

Definition at line 167 of file global.h.

struct { ... } magma_t::virus

Referenced by config_validate_settings(), virus_check(), virus_engine_create(), virus_engine_refresh(), virus_sigs_total(), virus_start(), and virus_stop().

struct { ... } magma_t::web

Referenced by http_print_301(), http_response(), portal_endpoint(), portal_endpoint_error(), portal_endpoint_response(), portal_process(), and portal_upload().

uint32_t magma_t::worker_threads

Definition at line 59 of file global.h.

Referenced by queue_init(), queue_shutdown(), and queue_signal().

uint32_t magma_t::wrap_line_length

Definition at line 108 of file global.h.

Referenced by config_validate_settings(), and mail_message_cleanup().

The documentation for this struct was generated from the following file:

- magma/engine/config/global/global.h

mail_cache_t Struct Reference

```
#include <mail.h>
```

Data Fields

- `uint64_t messagenum`
 - `stringer_t * text`
-

Detailed Description

Definition at line 19 of file mail.h.

Field Documentation

`uint64_t mail_cache_t::messagenum`

Definition at line 20 of file mail.h.

Referenced by `mail_cache_get()`, `mail_cache_reset()`, and `mail_cache_set()`.

`stringer_t* mail_cache_t::text`

Definition at line 21 of file mail.h.

Referenced by `mail_cache_destroy()`, `mail_cache_get()`, `mail_cache_reset()`, and `mail_cache_set()`.

The documentation for this struct was generated from the following file:

- `magma/objects/mail/mail.h`

mail_message_t Struct Reference

```
#include <mail.h>
```

Data Fields

- **mail_mime_t** * mime
 - **size_t** header_length
 - **placer_t** to
 - **placer_t** from
 - **placer_t** date
 - **placer_t** subject
 - **stringer_t** * text
-

Detailed Description

Definition at line 39 of file mail.h.

Field Documentation

placer_t mail_message_t::date

Definition at line 42 of file mail.h.

Referenced by mail_message().

placer_t mail_message_t::from

Definition at line 42 of file mail.h.

Referenced by mail_message().

size_t mail_message_t::header_length

Definition at line 41 of file mail.h.

Referenced by mail_load_header(), mail_load_message(), mail_message(), and portal_message_header().

mail_mime_t* mail_message_t::mime

Definition at line 40 of file mail.h.

Referenced by imap_fetch_body(), imap_fetch_body_part(), imap_fetch_message(), mail_destroy(), mail_mime_update(), portal_message_attachments(), and portal_message_body().

placer_t mail_message_t::subject

Definition at line 42 of file mail.h.

Referenced by mail_message().

stringer_t* mail_message_t::text

Definition at line 43 of file mail.h.

Referenced by imap_fetch_message(), mail_add_forward_headers(), mail_destroy(), mail_load_header(), mail_load_message(), mail_load_message_top(), mail_message(), mail_mime_update(), mail_modify_part(), mail_signature_add(), pop_retr(), pop_top(), and portal_message_header().

placer_t mail_message_t::to

Definition at line 42 of file mail.h.

Referenced by mail_message().

The documentation for this struct was generated from the following file:

- magma/objects/mail/**mail.h**

mail_mime_t Struct Reference

```
#include <mail.h>
```

Data Fields

- **array_t** * children
- **stringer_t** * boundary
- **uint32_t** type
- **uint32_t** encoding
- **placer_t** header
- **placer_t** body
- **placer_t** entire

Detailed Description

Definition at line 32 of file mail.h.

Field Documentation

placer_t mail_mime_t::body

Definition at line 36 of file mail.h.

Referenced by `imap_fetch_body()`, `imap_fetch_bodystructure()`, `imap_fetch_message()`, `mail_mime_part()`, `portal_message_attachments()`, and `portal_message_body()`.

stringer_t* mail_mime_t::boundary

Definition at line 34 of file mail.h.

Referenced by `mail_mime_free()`, and `mail_mime_part()`.

array_t* mail_mime_t::children

Definition at line 33 of file mail.h.

Referenced by `imap_fetch_body_part()`, `imap_fetch_bodystructure()`, `mail_mime_free()`, `mail_mime_part()`, `portal_message_attachments()`, and `portal_message_body()`.

uint32_t mail_mime_t::encoding

Definition at line 35 of file mail.h.

Referenced by `mail_mime_part()`.

placer_t mail_mime_t::entire

Definition at line 36 of file mail.h.

Referenced by mail_mime_part().

placer_t mail_mime_t::header

Definition at line 36 of file mail.h.

Referenced by imap_fetch_body(), imap_fetch_bodystructure(), mail_mime_part(), and portal_message_attachments().

uint32_t mail_mime_t::type

Definition at line 35 of file mail.h.

Referenced by mail_mime_part(), portal_message_attachments(), and portal_message_body().

The documentation for this struct was generated from the following file:

- magma/objects/mail/**mail.h**

mappings_t Struct Reference

```
#include <encodings.h>
```

Data Fields

- struct {
- **chr_t** characters [32]
- **chr_t** values [128]
- } **zbase32**
- struct {
- **chr_t** characters [64]
- **chr_t** values [128]
- } **base64**
- struct {
- **chr_t** characters [64]
- **chr_t** values [128]
- } **base64_mod**

Detailed Description

Definition at line 38 of file encodings.h.

Field Documentation

struct { ... } mappings_t::base64

Referenced by base64_decode(), and base64_encode().

struct { ... } mappings_t::base64_mod

Referenced by base64_decode_mod(), and base64_encode_mod().

chr_t mappings_t::characters[64]

Definition at line 40 of file encodings.h.

Referenced by base64_encode(), base64_encode_mod(), and zbase32_encode().

chr_t mappings_t::values[128] ()

Definition at line 40 of file encodings.h.

Referenced by base64_decode(), base64_decode_mod(), and zbase32_decode().

struct { ... } mappings_t::zbase32

Referenced by `zbase32_decode()`, and `zbase32_encode()`.

The documentation for this struct was generated from the following file:

- `magma/core/encodings/encodings.h`

media_type_t Struct Reference

```
#include <mail.h>
```

Data Fields

- **chr_t * extension**
 - **bool_t bin**
 - **chr_t * name**
-

Detailed Description

Definition at line 46 of file mail.h.

Field Documentation

bool_t media_type_t::bin

Definition at line 48 of file mail.h.

Referenced by mail_mime_encode_part().

chr_t* media_type_t::extension

Definition at line 47 of file mail.h.

chr_t* media_type_t::name

Definition at line 49 of file mail.h.

Referenced by mail_mime_encode_part().

The documentation for this struct was generated from the following file:

- magma/objects/mail/**mail.h**

meta_folder_t Struct Reference

```
#include <meta.h>
```

Data Fields

- **chr_t name** [128]
 - **uint32_t order**
 - **uint64_t parent**
 - **uint64_t foldernum**
-

Detailed Description

Definition at line 81 of file meta.h.

Field Documentation

uint64_t meta_folder_t::foldernum

Definition at line 84 of file meta.h.

Referenced by `imap_append()`, `imap_append_message()`, `imap_copy()`, `imap_folder_create()`, `imap_folder_remove()`, `imap_folder_rename()`, `imap_folder_status()`, `imap_narrow_folders()`, `meta_data_fetch_folders()`, `meta_messages_update_sequences()`, `portal_endpoint_folders_list()`, and `portal_folder_mail_add()`.

chr_t meta_folder_t::name[128]

Definition at line 82 of file meta.h.

Referenced by `imap_folder_create()`, `imap_folder_rename()`, `imap_list()`, `imap_lsub()`, `meta_data_fetch_folders()`, `meta_folders_name()`, and `portal_endpoint_folders_list()`.

uint32_t meta_folder_t::order

Definition at line 83 of file meta.h.

Referenced by `imap_folder_create()`, `imap_folder_rename()`, `imap_next_folder_order()`, and `meta_data_fetch_folders()`.

uint64_t meta_folder_t::parent

Definition at line 84 of file meta.h.

Referenced by `imap_folder_create()`, `imap_folder_rename()`, `imap_list()`, `imap_lsub()`, `imap_next_folder_order()`, `meta_data_fetch_folders()`, `meta_folders_children()`, `meta_folders_name()`, and `portal_endpoint_folders_list()`.

The documentation for this struct was generated from the following file:

- magma/network/**meta.h**

meta_message_t Struct Reference

```
#include <meta.h>
```

Data Fields

- `size_t` **size**
- `array_t *` **tags**
- `chr_t` **server** [33]
- `uint32_t` **status**
- `uint32_t` **updated**
- `uint64_t` **messagenum**
- `uint64_t` **foldernum**
- `uint64_t` **sequencenum**
- `uint64_t` **signum**
- `uint64_t` **sigkey**
- `uint64_t` **created**

Detailed Description

Definition at line 73 of file meta.h.

Field Documentation

`uint64_t meta_message_t::created`

Definition at line 78 of file meta.h.

Referenced by `imap_fetch_message()`, `imap_message_copier()`, `imap_search_messages_date()`, `meta_data_fetch_messages()`, `meta_messages_copier()`, `portal_endpoint_messages_list()`, and `portal_message_server()`.

`uint64_t meta_message_t::foldernum`

Definition at line 78 of file meta.h.

Referenced by `adjust_message_encryption()`, `imap_append()`, `imap_close()`, `imap_copy()`, `imap_examine()`, `imap_expunge()`, `imap_folder_remove()`, `imap_folder_status()`, `imap_narrow_messages()`, `imap_search_messages()`, `imap_select()`, `imap_session_destroy()`, `imap_session_update()`, `imap_update_flags()`, `meta_data_fetch_messages()`, `meta_data_flags_remove()`, `meta_data_flags_replace()`, `meta_folders_stats_tags()`, `meta_messages_mover()`, `meta_messages_update_sequences()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_tag()`, and `portal_message_meta()`.

`uint64_t meta_message_t::messagenum`

Definition at line 78 of file meta.h.

Referenced by `adjust_message_encryption()`, `imap_copy()`, `imap_duplicate_messages()`, `imap_fetch_message()`, `imap_folder_remove()`, `imap_folder_status()`, `imap_message_copier()`, `imap_message_expunge()`, `imap_narrow_messages()`, `imap_search()`, `imap_search_messages()`, `imap_search_messages_range()`, `imap_store()`, `mail_load_header()`, `mail_load_message()`, `meta_data_delete_tag()`, `meta_data_fetch_message_tags()`, `meta_data_fetch_messages()`, `meta_data_flags_add()`, `meta_data_flags_remove()`, `meta_data_flags_replace()`, `meta_data_insert_tag()`, `meta_data_truncate_tags()`, `meta_messages_copier()`, `meta_messages_mover()`, `pop_session_destroy()`, `pop_uidl()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_tag()`, and `portal_message_meta()`.

uint64_t meta_message_t::sequencenum

Definition at line 78 of file `meta.h`.

Referenced by `imap_copy()`, `imap_expunge()`, `imap_fetch()`, `imap_narrow_messages()`, `imap_search()`, `imap_search_messages()`, `imap_search_messages_range()`, `imap_store()`, and `meta_messages_update_sequences()`.

chr_t meta_message_t::server[33]

Definition at line 76 of file `meta.h`.

Referenced by `adjust_message_encryption()`, `imap_folder_remove()`, `imap_message_copier()`, `imap_message_expunge()`, `mail_load_header()`, `mail_load_message()`, `meta_data_fetch_messages()`, `meta_messages_copier()`, `pop_session_destroy()`, and `portal_endpoint_messages_remove()`.

uint64_t meta_message_t::sigkey

Definition at line 78 of file `meta.h`.

Referenced by `imap_message_copier()`, `mail_load_message()`, `meta_data_fetch_messages()`, and `meta_messages_copier()`.

uint64_t meta_message_t::signum

Definition at line 78 of file `meta.h`.

Referenced by `imap_message_copier()`, `mail_load_message()`, `meta_data_fetch_messages()`, and `meta_messages_copier()`.

size_t meta_message_t::size

Definition at line 74 of file `meta.h`.

Referenced by `imap_fetch_message()`, `imap_folder_remove()`, `imap_message_copier()`, `imap_message_expunge()`, `imap_search_messages_size()`, `meta_data_fetch_messages()`, `meta_messages_copier()`, `pop_list()`, `pop_session_destroy()`, `pop_total_size()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_remove()`, and `portal_message_header()`.

uint32_t meta_message_t::status

Definition at line 77 of file `meta.h`.

Referenced by `adjust_message_encryption()`, `decrypt_user_messages()`, `encrypt_user_messages()`, `imap_append()`, `imap_close()`, `imap_copy()`, `imap_examine()`, `imap_expunge()`, `imap_fetch()`, `imap_fetch_message()`, `imap_folder_status()`, `imap_message_copier()`, `imap_search_messages_inner()`, `imap_select()`, `imap_session_destroy()`, `imap_session_update()`, `imap_store()`, `imap_update_flags()`, `mail_load_header()`, `mail_load_message()`, `meta_check_message_encryption()`, `meta_data_fetch_messages()`, `meta_messages_copier()`, `meta_messages_mover()`, `pop_delete()`, `pop_get_last()`, `pop_get_message()`, `pop_list()`, `pop_retr()`, `pop_session_destroy()`, `pop_session_reset()`, `pop_top()`, `pop_total_messages()`, `pop_total_size()`, `pop_uidl()`, `portal_endpoint_messages_flag()`, and `portal_message_flags_array()`.

array_t* meta_message_t::tags

Definition at line 75 of file `meta.h`.

Referenced by `meta_data_fetch_message_tags()`, `meta_folders_stats_tags()`, `meta_message_dupe()`, `meta_message_free()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_tag()`, and `portal_message_tags_array()`.

uint32_t meta_message_t::updated

Definition at line 77 of file `meta.h`.

Referenced by `imap_fetch()`, and `imap_fetch_message()`.

The documentation for this struct was generated from the following file:

- `magma/network/meta.h`

meta_stats_tag_t Struct Reference

```
#include <users.h>
```

Data Fields

- `uint64_t count`
 - `stringer_t * tag`
-

Detailed Description

Definition at line 70 of file `users.h`.

Field Documentation

`uint64_t meta_stats_tag_t::count`

Definition at line 71 of file `users.h`.

Referenced by `meta_folders_stats_tags()`, and `portal_endpoint_folders_tags()`.

`stringer_t* meta_stats_tag_t::tag`

Definition at line 72 of file `users.h`.

Referenced by `meta_folder_stats_tag_alloc()`, and `portal_endpoint_folders_tags()`.

The documentation for this struct was generated from the following file:

- `magma/objects/users/users.h`

meta_user_t Struct Reference

```
#include <meta.h>
```

Data Fields

- **META_USER_FLAGS** flags
- pthread_mutex_t **lock**
- stringer_t * **username**
- stringer_t * **passhash**
- stringer_t * **storage_privkey**
- stringer_t * **storage_pubkey**
- inx_t * **aliases**
- inx_t * **messages**
- inx_t * **folders**
- inx_t * **message_folders**
- inx_t * **ads**
- inx_t * **contacts**
- struct {
- uint64_t **user**
- uint64_t **messages**
- uint64_t **folders**
- uint64_t **contacts**
- } **serials**
- struct {
- time_t **stamp**
- uint64_t **smtp**
- uint64_t **pop**
- uint64_t **imap**
- uint64_t **web**
- uint64_t **generic**
- pthread_mutex_t **lock**
- } **refs**
- uint64_t **usernum**
- uint64_t **lock_status**

Detailed Description

Definition at line 88 of file meta.h.

Field Documentation

inx_t * meta_user_t::ads

Definition at line 95 of file meta.h.

Referenced by meta_user_destroy().

inx_t* meta_user_t::aliases

Definition at line 95 of file meta.h.

Referenced by meta_data_fetch_mailbox_aliases(), and meta_user_destroy().

uint64_t meta_user_t::contacts

Definition at line 97 of file meta.h.

inx_t * meta_user_t::contacts

Definition at line 95 of file meta.h.

Referenced by meta_contacts_update(), meta_user_destroy(), meta_user_serial_get(), and meta_user_serial_set().

META_USER_FLAGS meta_user_t::flags

Definition at line 89 of file meta.h.

Referenced by adjust_message_encryption(), mail_load_message(), meta_check_message_encryption(), meta_data_fetch_user(), meta_data_user_build(), and portal_endpoint_auth().

uint64_t meta_user_t::folders

Definition at line 97 of file meta.h.

inx_t * meta_user_t::folders

Definition at line 95 of file meta.h.

Referenced by meta_data_fetch_folders(), meta_folders_update(), meta_message_folders_update(), meta_messages_copier(), meta_messages_login_update(), meta_messages_mover(), meta_messages_update(), meta_user_destroy(), meta_user_serial_get(), and meta_user_serial_set().

uint64_t meta_user_t::generic

Definition at line 101 of file meta.h.

Referenced by meta_user_ref_add(), meta_user_ref_dec(), and meta_user_ref_total().

uint64_t meta_user_t::imap

Definition at line 101 of file meta.h.

Referenced by meta_user_ref_add(), meta_user_ref_dec(), and meta_user_ref_total().

pthread_mutex_t meta_user_t::lock

Definition at line 93 of file meta.h.

Referenced by meta_user_create(), meta_user_destroy(), meta_user_ref_add(), meta_user_ref_dec(), meta_user_ref_stamp(), meta_user_ref_total(), meta_user_rlock(), meta_user_unlock(), and meta_user_wlock().

uint64_t meta_user_t::lock_status

Definition at line 104 of file meta.h.

Referenced by meta_data_fetch_user(), meta_data_user_build(), and portal_endpoint_auth().

inx_t * meta_user_t::message_folders

Definition at line 95 of file meta.h.

Referenced by meta_message_folders_update(), and meta_user_destroy().

uint64_t meta_user_t::messages

Definition at line 97 of file meta.h.

inx_t * meta_user_t::messages

Definition at line 95 of file meta.h.

Referenced by decrypt_user_messages(), encrypt_user_messages(), meta_check_message_encryption(), meta_data_fetch_messages(), meta_folders_update(), meta_messages_copier(), meta_messages_login_update(), meta_messages_mover(), meta_messages_update(), meta_user_destroy(), meta_user_serial_get(), and meta_user_serial_set().

stringer_t * meta_user_t::passhash

Definition at line 94 of file meta.h.

Referenced by meta_data_fetch_user(), meta_data_user_build(), meta_get(), meta_user_build(), and meta_user_destroy().

uint64_t meta_user_t::pop

Definition at line 101 of file meta.h.

Referenced by meta_folders_update(), meta_messages_login_update(), meta_messages_update(), meta_user_ref_add(), meta_user_ref_dec(), and meta_user_ref_total().

struct { ... } meta_user_t::refs

Referenced by meta_folders_update(), meta_messages_login_update(), meta_messages_update(), meta_user_create(), meta_user_destroy(), meta_user_ref_add(), meta_user_ref_dec(), meta_user_ref_stamp(), and meta_user_ref_total().

struct { ... } meta_user_t::serials

Referenced by meta_contacts_update(), meta_folders_update(), meta_message_folders_update(), meta_messages_login_update(), meta_messages_update(), meta_user_build(), meta_user_serial_get(), meta_user_serial_set(), and meta_user_update().

uint64_t meta_user_t::smtp

Definition at line 101 of file meta.h.

Referenced by meta_user_ref_add(), meta_user_ref_dec(), and meta_user_ref_total().

time_t meta_user_t::stamp

Definition at line 100 of file meta.h.

Referenced by meta_user_ref_add(), meta_user_ref_dec(), and meta_user_ref_stamp().

stringer_t * meta_user_t::storage_privkey

Definition at line 94 of file meta.h.

Referenced by adjust_message_encryption(), mail_load_header(), mail_load_message(), meta_check_message_encryption(), meta_data_user_build(), and meta_user_destroy().

stringer_t * meta_user_t::storage_pubkey

Definition at line 94 of file meta.h.

Referenced by adjust_message_encryption(), meta_check_message_encryption(), meta_data_user_build(), and meta_user_destroy().

uint64_t meta_user_t::user

Definition at line 97 of file meta.h.

Referenced by meta_user_build(), meta_user_serial_get(), meta_user_serial_set(), and meta_user_update().

stringer_t* meta_user_t::username

Definition at line 94 of file meta.h.

Referenced by decrypt_user_messages(), encrypt_user_messages(), meta_data_fetch_messages(), meta_data_user_build(), meta_user_build(), meta_user_destroy(), portal_endpoint_auth(), and sess_destroy().

uint64_t meta_user_t::usernum

Definition at line 104 of file meta.h.

Referenced by adjust_message_encryption(), imap_update_flags(), mail_load_header(), mail_load_message(), meta_contacts_update(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_messages(), meta_data_fetch_user(), meta_data_update_log(), and meta_data_user_build(),

meta_folders_update(), meta_message_folders_update(), meta_messages_copier(),
meta_messages_login_update(), meta_messages_mover(), meta_messages_update(), meta_user_build(),
meta_user_serial_check(), and meta_user_update().

uint64_t meta_user_t::web

Definition at line 101 of file meta.h.

Referenced by meta_user_ref_add(), meta_user_ref_dec(), and meta_user_ref_total().

The documentation for this struct was generated from the following file:

- magma/network/**meta.h**

multi_t Struct Reference

```
#include <strings.h>
```

Data Fields

- **M_TYPE** type
- union {
- **bool_t** **binary**
- void * **bl**
- char * **ns**
- **stringer_t** * **st**
- uint8_t **u8**
- uint16_t **u16**
- uint32_t **u32**
- uint64_t **u64**
- int8_t **i8**
- int16_t **i16**
- int32_t **i32**
- int64_t **i64**
- float **fl**
- double **dbl**
- } **val**

Detailed Description

Definition at line 205 of file strings.h.

Field Documentation

bool_t multi_t::binary

Definition at line 208 of file strings.h.

Referenced by cache_set_value(), cmp_mt_mt(), config_value_set(), ident_mt_mt(), mt_dupe(), mt_get_char(), relay_set_value(), and servers_set_value().

void* multi_t::bl

Definition at line 209 of file strings.h.

Referenced by cache_output_help(), config_output_help(), mt_get_char(), mt_is_empty(), relay_output_help(), and servers_output_help().

double multi_t::dbl

Definition at line 221 of file strings.h.

Referenced by cmp_mt_mt(), ident_mt_mt(), mt_dupe(), mt_get_char(), and mt_get_number().

float multi_t::fl

Definition at line 220 of file strings.h.

Referenced by cmp_mt_mt(), ident_mt_mt(), mt_dupe(), mt_get_char(), and mt_get_number().

int16_t multi_t::i16

Definition at line 217 of file strings.h.

Referenced by cache_validate(), cmp_mt_mt(), ident_mt_mt(), mt_dupe(), mt_get_char(), mt_get_number(), relay_validate(), and servers_validate().

int32_t multi_t::i32

Definition at line 218 of file strings.h.

Referenced by cache_set_value(), cache_validate(), cmp_mt_mt(), config_value_set(), ident_mt_mt(), mt_dupe(), mt_get_char(), mt_get_number(), relay_set_value(), relay_validate(), servers_set_value(), and servers_validate().

int64_t multi_t::i64

Definition at line 219 of file strings.h.

Referenced by cache_set_value(), cache_validate(), cmp_mt_mt(), config_value_set(), ident_mt_mt(), mt_dupe(), mt_get_char(), mt_get_number(), relay_set_value(), relay_validate(), servers_set_value(), and servers_validate().

int8_t multi_t::i8

Definition at line 216 of file strings.h.

Referenced by cache_set_value(), cache_validate(), cmp_mt_mt(), config_value_set(), ident_mt_mt(), mt_dupe(), mt_get_char(), mt_get_number(), relay_set_value(), relay_validate(), servers_set_value(), and servers_validate().

char* multi_t::ns

Definition at line 210 of file strings.h.

Referenced by cache_set_value(), cmp_mt_mt(), config_value_set(), ident_mt_mt(), mt_dupe(), mt_free(), mt_get_char(), mt_get_length(), mt_is_empty(), relay_set_value(), and servers_set_value().

stringer_t* multi_t::st

Definition at line 211 of file strings.h.

Referenced by cache_set_value(), cmp_mt_mt(), config_load_cmdline_settings(), config_load_file_settings(), config_value_set(), contact_details_fetch(), contact_edit(), http_data_value_parse(), http_load_file(), http_parse_header(), ident_mt_mt(), meta_folders_stats_tags(), mt_dupe(), mt_free(), mt_get_char(), mt_get_length(), mt_is_empty(), nvp_parse(), relay_set_value(), servers_set_value(), teacher_add_cookie(), user_config_edit(), user_config_fetch(), and warehouse_fetch_domains().

M_TYPE multi_t::type

Definition at line 206 of file strings.h.

Referenced by `adjust_message_encryption()`, `cache_free()`, `cache_output_settings()`, `cache_set_value()`, `cache_validate()`, `cmp_mt_mt()`, `config_free()`, `config_output_value()`, `config_value_set()`, `ident_mt_mt()`, `imap_message_copier()`, `meta_data_fetch_all_tags()`, `meta_data_fetch_folders()`, `meta_data_fetch_messages()`, `mt_dupe()`, `mt_free()`, `mt_get_char()`, `mt_get_length()`, `mt_get_null()`, `mt_get_number()`, `mt_get_type()`, `mt_is_empty()`, `mt_is_number()`, `mt_set_type()`, `relay_free()`, `relay_output_settings()`, `relay_set_value()`, `relay_validate()`, `servers_free()`, `servers_output_settings()`, `servers_set_value()`, `servers_validate()`, and `smtp_add_bypass_entry()`.

uint16_t multi_t::u16

Definition at line 213 of file strings.h.

Referenced by `cache_set_value()`, `cache_validate()`, `cmp_mt_mt()`, `config_value_set()`, `ident_mt_mt()`, `mt_dupe()`, `mt_get_char()`, `mt_get_number()`, `relay_set_value()`, `relay_validate()`, `servers_set_value()`, and `servers_validate()`.

uint32_t multi_t::u32

Definition at line 214 of file strings.h.

Referenced by `cache_set_value()`, `cache_validate()`, `cmp_mt_mt()`, `config_value_set()`, `ident_mt_mt()`, `mt_dupe()`, `mt_get_char()`, `mt_get_number()`, `relay_set_value()`, `relay_validate()`, `servers_set_value()`, and `servers_validate()`.

uint64_t multi_t::u64

Definition at line 215 of file strings.h.

Referenced by `adjust_message_encryption()`, `cache_set_value()`, `cache_validate()`, `cmp_mt_mt()`, `config_value_set()`, `contact_folder_create()`, `contact_move()`, `contacts_fetch()`, `http_content_load_fonts()`, `ident_mt_mt()`, `imap_append_message()`, `imap_duplicate_messages()`, `imap_folder_create()`, `imap_folder_remove()`, `imap_folder_rename()`, `imap_message_copier()`, `imap_narrow_folders()`, `imap_narrow_messages()`, `imap_search_messages()`, `magma_folder_fetch()`, `message_folder_create()`, `messages_fetch()`, `meta_data_fetch_alerts()`, `meta_data_fetch_all_tags()`, `meta_data_fetch_folders()`, `meta_data_fetch_mailbox_aliases()`, `meta_data_fetch_messages()`, `meta_messages_copier()`, `meta_messages_mover()`, `mt_dupe()`, `mt_get_char()`, `mt_get_number()`, `pop_session_destroy()`, `portal_endpoint_attachments_add()`, `portal_endpoint_attachments_remove()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_contacts_load()`, `portal_endpoint_contacts_remove()`, `portal_endpoint_messages_compose()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_load()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_send()`, `portal_endpoint_messages_tag()`, `portal_get_upload_attachment()`, `portal_parse_json_str_array()`, `register_data_fetch_blocklist()`, `relay_set_value()`, `relay_validate()`, `servers_set_value()`, `servers_validate()`, `sess_create()`, `sess_get()`, `smtp_add_bypass_entry()`, `smtp_fetch_inbound()`, and `warehouse_fetch_patterns()`.

uint8_t multi_t::u8

Definition at line 212 of file strings.h.

Referenced by `cache_set_value()`, `cache_validate()`, `cmp_mt_mt()`, `config_value_set()`, `ident_mt_mt()`, `mt_dupe()`, `mt_get_char()`, `mt_get_number()`, `relay_set_value()`, `relay_validate()`, `servers_set_value()`, and `servers_validate()`.

union { ... } multi_t::val

Referenced by `adjust_message_encryption()`, `cache_output_help()`, `cache_set_value()`, `cache_validate()`, `cmp_mt_mt()`, `config_load_cmdline_settings()`, `config_load_file_settings()`, `config_output_help()`, `config_value_set()`, `contact_details_fetch()`, `contact_edit()`, `contact_folder_create()`, `contact_move()`, `contacts_fetch()`, `http_content_load_fonts()`, `http_data_value_parse()`, `http_load_file()`, `http_parse_header()`, `ident_mt_mt()`, `imap_append_message()`, `imap_duplicate_messages()`, `imap_folder_create()`, `imap_folder_remove()`, `imap_folder_rename()`, `imap_message_copier()`, `imap_narrow_folders()`, `imap_narrow_messages()`, `imap_search_messages()`, `magma_folder_fetch()`, `message_folder_create()`, `messages_fetch()`, `meta_data_fetch_alerts()`, `meta_data_fetch_all_tags()`, `meta_data_fetch_folders()`, `meta_data_fetch_mailbox_aliases()`, `meta_data_fetch_messages()`, `meta_folders_stats_tags()`, `meta_messages_copier()`, `meta_messages_mover()`, `mt_dupe()`, `mt_free()`, `mt_get_char()`, `mt_get_length()`, `mt_get_number()`, `mt_is_empty()`, `nvp_parse()`, `pop_session_destroy()`, `portal_endpoint_attachments_add()`, `portal_endpoint_attachments_remove()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_contacts_load()`, `portal_endpoint_contacts_remove()`, `portal_endpoint_messages_compose()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_load()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_send()`, `portal_endpoint_messages_tag()`, `portal_get_upload_attachment()`, `portal_parse_json_str_array()`, `register_data_fetch_blocklist()`, `relay_output_help()`, `relay_set_value()`, `relay_validate()`, `servers_output_help()`, `servers_set_value()`, `servers_validate()`, `sess_create()`, `sess_get()`, `smtp_add_bypass_entry()`, `smtp_fetch_inbound()`, `teacher_add_cookie()`, `user_config_edit()`, `user_config_fetch()`, `warehouse_fetch_domains()`, and `warehouse_fetch_patterns()`.

The documentation for this struct was generated from the following file:

- `magma/core/strings/strings.h`

neue_t Struct Reference

```
#include <neue.h>
```

Data Fields

- struct {
- time_t **stamp**
- uint64_t **flags**
- uint64_t **usernum**
- } **info**
- **credential_t** * **cred**
- pthread_rwlock_t **lock**

Detailed Description

Definition at line 46 of file neue.h.

Field Documentation

credential_t* neue_t::cred

Definition at line 54 of file neue.h.

uint64_t neue_t::flags

Definition at line 50 of file neue.h.

struct { ... } neue_t::info

pthread_rwlock_t neue_t::lock

Definition at line 55 of file neue.h.

Referenced by `neue_rlock()`, `neue_unlock()`, and `neue_wlock()`.

time_t neue_t::stamp

Definition at line 49 of file neue.h.

uint64_t neue_t::usernum

Definition at line 51 of file neue.h.

The documentation for this struct was generated from the following file:

- magma/objects/neue/**neue.h**

nvp_t Struct Reference

```
#include <formats.h>
```

Data Fields

- **MAGMA_INDEX options**
 - struct {
 - char **comment**
 - char **value**
 - char **line**
 - } **tokens**
 - **inx_t * pairs**
-

Detailed Description

Definition at line 18 of file formats.h.

Field Documentation

char nvp_t::comment

Definition at line 21 of file formats.h.

Referenced by nvp_alloc(), and nvp_parse().

char nvp_t::line

Definition at line 21 of file formats.h.

Referenced by nvp_alloc(), and nvp_parse().

MAGMA_INDEX nvp_t::options

Definition at line 19 of file formats.h.

inx_t* nvp_t::pairs

Definition at line 23 of file formats.h.

Referenced by config_load_cmdline_settings(), config_load_file_settings(), nvp_alloc(), nvp_free(), and nvp_parse().

struct { ... } nvp_t::tokens

Referenced by nvp_alloc(), and nvp_parse().

char nvp_t::value

Definition at line 21 of file formats.h.

Referenced by nvp_alloc(), and nvp_parse().

The documentation for this struct was generated from the following file:

- magma/core/parsers/formats/**formats.h**

object_cache_t Struct Reference

```
#include <objects.h>
```

Data Fields

- `inx_t * users`
 - `inx_t * sessions`
-

Detailed Description

Definition at line 34 of file `objects.h`.

Field Documentation

`inx_t * object_cache_t::sessions`

Definition at line 35 of file `objects.h`.

Referenced by `obj_cache_prune()`, `obj_cache_start()`, `obj_cache_stop()`, `sess_create()`, and `sess_get()`.

`inx_t* object_cache_t::users`

Definition at line 35 of file `objects.h`.

Referenced by `meta_get()`, `meta_remove()`, `meta_user_prune()`, `obj_cache_prune()`, `obj_cache_start()`, and `obj_cache_stop()`.

The documentation for this struct was generated from the following file:

- `magma/objects/objects.h`

pool_t Struct Reference

```
#include <buckets.h>
```

Data Fields

- uint32_t **count**
- uint32_t **timeout**
- uint64_t **failures**
- sem_t **available**
- pthread_mutex_t **lock**
- status_t * **status**
- void ** **objects**

Detailed Description

Definition at line 59 of file buckets.h.

Field Documentation

sem_t pool_t::available

Definition at line 63 of file buckets.h.

Referenced by pool_alloc(), pool_free(), pool_get_available(), pool_pull(), and pool_release().

uint32_t pool_t::count

Definition at line 60 of file buckets.h.

Referenced by pool_alloc(), pool_get_count(), and pool_pull().

uint64_t pool_t::failures

Definition at line 62 of file buckets.h.

Referenced by pool_get_failures(), and pool_pull().

pthread_mutex_t pool_t::lock

Definition at line 64 of file buckets.h.

Referenced by pool_alloc(), pool_free(), pool_get_failures(), pool_get_obj(), pool_pull(), pool_release(), pool_set_obj(), and pool_swap_obj().

void** pool_t::objects

Definition at line 66 of file buckets.h.

Referenced by pool_alloc(), pool_get_obj(), and pool_set_obj().

status_t* pool_t::status

Definition at line 65 of file buckets.h.

Referenced by pool_alloc(), pool_get_status(), and pool_set_status().

uint32_t pool_t::timeout

Definition at line 61 of file buckets.h.

Referenced by pool_alloc(), pool_get_timeout(), and pool_pull().

The documentation for this struct was generated from the following file:

- magma/core/buckets/**buckets.h**

queue_t Struct Reference

Public Member Functions

- void(* **requeue** (void ***data**))(*)

Data Fields

- void(* **function**)(void ***data**)
- void(*) (*)* **data**
- struct **queue_t** * **next**

Detailed Description

Definition at line 15 of file queue.c.

Member Function Documentation

void(* queue_t::requeue (void * *data*)

Referenced by dequeue(), and requeue().

Field Documentation

void(*) (*) * queue_t::data

Definition at line 16 of file queue.c.

Referenced by dequeue(), and requeue().

void(* queue_t::function)(void *data)

Referenced by dequeue(), and requeue().

struct queue_t* queue_t::next [read]

Definition at line 17 of file queue.c.

Referenced by dequeue(), and requeue().

The documentation for this struct was generated from the following file:

- magma/engine/controller/**queue.c**

register_session_t Struct Reference

```
#include <register.h>
```

Data Fields

- uint64_t **usernum**
- uint16_t **plan**
- stringer_t * **username**
- stringer_t * **password**
- stringer_t * **hvf_value**
- stringer_t * **hvf_input**
- stringer_t * **name**

Detailed Description

Definition at line 22 of file register.h.

Field Documentation

stringer_t * register_session_t::hvf_input

Definition at line 25 of file register.h.

Referenced by register_business_step1(), register_session_cache(), register_session_free(), and register_session_get().

stringer_t * register_session_t::hvf_value

Definition at line 25 of file register.h.

Referenced by register_business_step1(), register_print_captcha(), register_session_cache(), register_session_free(), and register_session_get().

stringer_t * register_session_t::name

Definition at line 25 of file register.h.

Referenced by register_print_step1(), register_print_step2(), register_session_cache(), register_session_free(), register_session_generate(), and register_session_get().

stringer_t * register_session_t::password

Definition at line 25 of file register.h.

Referenced by register_business_step1(), register_data_insert_user(), register_session_cache(), register_session_free(), and register_session_get().

uint16_t register_session_t::plan

Definition at line 24 of file register.h.

Referenced by register_business_step2(), register_data_insert_user(), register_session_cache(), and register_session_get().

stringer_t* register_session_t::username

Definition at line 25 of file register.h.

Referenced by register_business_step1(), register_business_step2(), register_data_insert_user(), register_print_step1(), register_session_cache(), register_session_free(), and register_session_get().

uint64_t register_session_t::usernum

Definition at line 23 of file register.h.

Referenced by register_business_step2(), register_process(), register_session_cache(), and register_session_get().

The documentation for this struct was generated from the following file:

- magma/web/register/**register.h**

relay_keys_t Struct Reference

```
#include <relay.h>
```

Data Fields

- `size_t offset`
 - `multi_t norm`
 - `chr_t * name`
 - `chr_t * description`
 - `bool_t required`
-

Detailed Description

Definition at line 16 of file relay.h.

Field Documentation

`chr_t* relay_keys_t::description`

Definition at line 20 of file relay.h.

`chr_t* relay_keys_t::name`

Definition at line 19 of file relay.h.

Referenced by `relay_set_value()`.

`multi_t relay_keys_t::norm`

Definition at line 18 of file relay.h.

Referenced by `relay_free()`, `relay_output_help()`, `relay_output_settings()`, `relay_set_value()`, and `relay_validate()`.

`size_t relay_keys_t::offset`

Definition at line 17 of file relay.h.

Referenced by `relay_free()`, `relay_output_settings()`, `relay_set_value()`, and `relay_validate()`.

`bool_t relay_keys_t::required`

Definition at line 21 of file relay.h.

Referenced by `relay_output_help()`.

The documentation for this struct was generated from the following file:

- magma/engine/config/relay/**relay.h**

relay_t Struct Reference

```
#include <relay.h>
```

Data Fields

- `bool_t` `secure`
 - `bool_t` `premium`
 - `chr_t *` `name`
 - `uint32_t` `port`
-

Detailed Description

Definition at line 24 of file relay.h.

Field Documentation

`chr_t*` `relay_t::name`

Definition at line 27 of file relay.h.

Referenced by `smtp_client_connect()`.

`uint32_t` `relay_t::port`

Definition at line 28 of file relay.h.

Referenced by `smtp_client_connect()`.

`bool_t` `relay_t::premium`

Definition at line 26 of file relay.h.

`bool_t` `relay_t::secure`

Definition at line 25 of file relay.h.

Referenced by `smtp_client_connect()`.

The documentation for this struct was generated from the following file:

- `magma/engine/config/relay/relay.h`

serialization_t Struct Reference

```
#include <consumers.h>
```

Data Fields

- `size_t position`
 - `stringer_t * data`
-

Detailed Description

Definition at line 17 of file consumers.h.

Field Documentation

`stringer_t* serialization_t::data`

Definition at line 19 of file consumers.h.

Referenced by `deserialize_int16()`, `deserialize_int32()`, `deserialize_int64()`, `deserialize_ns()`, `deserialize_st()`, `deserialize_uint16()`, `deserialize_uint32()`, `deserialize_uint64()`, `register_session_get()`, and `teacher_data_get()`.

`size_t serialization_t::position`

Definition at line 18 of file consumers.h.

Referenced by `deserialize_int16()`, `deserialize_int32()`, `deserialize_int64()`, `deserialize_ns()`, `deserialize_st()`, `deserialize_uint16()`, `deserialize_uint32()`, and `deserialize_uint64()`.

The documentation for this struct was generated from the following file:

- `magma/providers/consumers/consumers.h`

server_config_t Struct Reference

```
#include <servers.h>
```

Data Fields

- `SSL_CTX * ssl_ctx`
- `char * certificate`
- `int ssl_sd`
- `int normal_sd`
- `stringer_t * name`
- `stringer_t * spam`
- `stringer_t * domain`
- `stringer_t * banner`
- `unsigned ssl_port`
- `unsigned normal_port`
- `unsigned listen_queue`
- `unsigned socket_timeout`
- `unsigned bad_command_cutoff`
- `unsigned bad_command_delay`
- `struct server_config_t * next`

Detailed Description

Definition at line 62 of file servers.h.

Field Documentation

unsigned server_config_t::bad_command_cutoff

Definition at line 67 of file servers.h.

unsigned server_config_t::bad_command_delay

Definition at line 67 of file servers.h.

stringer_t * server_config_t::banner

Definition at line 66 of file servers.h.

char* server_config_t::certificate

Definition at line 64 of file servers.h.

stringer_t * server_config_t::domain

Definition at line 66 of file servers.h.

unsigned server_config_t::listen_queue

Definition at line 67 of file servers.h.

stringer_t* server_config_t::name

Definition at line 66 of file servers.h.

struct server_config_t* server_config_t::next [read]

Definition at line 68 of file servers.h.

unsigned server_config_t::normal_port

Definition at line 67 of file servers.h.

int server_config_t::normal_sd

Definition at line 65 of file servers.h.

unsigned server_config_t::socket_timeout

Definition at line 67 of file servers.h.

stringer_t * server_config_t::spam

Definition at line 66 of file servers.h.

SSL_CTX* server_config_t::ssl_ctx

Definition at line 63 of file servers.h.

unsigned server_config_t::ssl_port

Definition at line 67 of file servers.h.

int server_config_t::ssl_sd

Definition at line 65 of file servers.h.

The documentation for this struct was generated from the following file:

- magma/engine/config/servers/**servers.h**

server_keys_t Struct Reference

```
#include <servers.h>
```

Data Fields

- `size_t offset`
 - `multi_t norm`
 - `chr_t * name`
 - `chr_t * description`
 - `bool_t required`
-

Detailed Description

Definition at line 30 of file servers.h.

Field Documentation

`chr_t* server_keys_t::description`

Definition at line 34 of file servers.h.

`chr_t* server_keys_t::name`

Definition at line 33 of file servers.h.

Referenced by `servers_set_value()`.

`multi_t server_keys_t::norm`

Definition at line 32 of file servers.h.

Referenced by `servers_free()`, `servers_output_help()`, `servers_output_settings()`, `servers_set_value()`, and `servers_validate()`.

`size_t server_keys_t::offset`

Definition at line 31 of file servers.h.

Referenced by `servers_free()`, `servers_output_settings()`, `servers_set_value()`, and `servers_validate()`.

`bool_t server_keys_t::required`

Definition at line 35 of file servers.h.

Referenced by `servers_output_help()`.

The documentation for this struct was generated from the following file:

- magma/engine/config/servers/**servers.h**

server_t Struct Reference

```
#include <servers.h>
```

Data Fields

- struct {
- SSL_CTX * **context**
- char * **certificate**
- } **ssl**
- struct {
- int **sockd**
- bool_t **ipv6**
- uint32_t **port**
- uint32_t **timeout**
- uint32_t **listen_queue**
- M_PORT type
- } **network**
- struct {
- uint32_t **delay**
- uint32_t **cutoff**
- } **violations**
- bool_t **enabled**
- stringer_t * **name**
- stringer_t * **domain**
- M_PROTOCOL **protocol**

Detailed Description

Definition at line 38 of file servers.h.

Field Documentation

char* server_t::certificate

Definition at line 41 of file servers.h.

Referenced by servers_encryption_start(), servers_validate(), and ssl_server_create().

SSL_CTX* server_t::context

Definition at line 40 of file servers.h.

Referenced by con_secure(), protocol_process(), servers_encryption_stop(), ssl_alloc(), ssl_server_create(), and ssl_server_destroy().

uint32_t server_t::cutoff

Definition at line 53 of file servers.h.

Referenced by http_process(), http_requeue(), imap_process(), imap_requeue(), pop_process(), pop_requeue(), smtp_process(), and smtp_requeue().

uint32_t server_t::delay

Definition at line 52 of file servers.h.

Referenced by imap_invalid(), pop_invalid(), smtp_disabled(), and smtp_invalid().

stringer_t * server_t::domain

Definition at line 56 of file servers.h.

Referenced by imap_init(), mail_add_inbound_headers(), mail_add_outbound_headers(), mail_build_signature(), servers_network_start(), smtp_ehlo(), smtp_helo(), and smtp_init().

bool_t server_t::enabled

Definition at line 55 of file servers.h.

bool_t server_t::ipv6

Definition at line 45 of file servers.h.

Referenced by net_init().

uint32_t server_t::listen_queue

Definition at line 48 of file servers.h.

Referenced by net_init().

stringer_t* server_t::name

Definition at line 56 of file servers.h.

Referenced by servers_network_start().

struct { ... } server_t::network

Referenced by net_init(), net_listen(), net_shutdown(), protocol_process(), servers_alloc(), servers_get_by_socket(), servers_get_count_using_port(), servers_network_stop(), and servers_validate().

uint32_t server_t::port

Definition at line 46 of file servers.h.

Referenced by net_init(), servers_get_count_using_port(), and servers_validate().

M_PROTOCOL server_t::protocol

Definition at line 57 of file servers.h.

Referenced by con_destroy(), and protocol_enqueue().

int server_t::sockd

Definition at line 44 of file servers.h.

Referenced by net_init(), net_listen(), net_shutdown(), servers_alloc(), servers_get_by_socket(), and servers_network_stop().

struct { ... } server_t::ssl

Referenced by con_secure(), protocol_process(), servers_encryption_start(), servers_encryption_stop(), servers_validate(), ssl_alloc(), ssl_server_create(), and ssl_server_destroy().

uint32_t server_t::timeout

Definition at line 47 of file servers.h.

Referenced by protocol_process().

M_PORT server_t::type

Definition at line 49 of file servers.h.

Referenced by protocol_process().

struct { ... } server_t::violations

Referenced by http_process(), http_requeue(), imap_invalid(), imap_process(), imap_requeue(), pop_invalid(), pop_process(), pop_requeue(), smtp_disabled(), smtp_invalid(), smtp_process(), and smtp_requeue().

The documentation for this struct was generated from the following file:

- magma/engine/config/servers/servers.h

session_t Struct Reference

```
#include <sessions.h>
```

Data Fields

- **int_t** state
- **meta_user_t** * user
- **inx_t** * compositions
- **uint64_t** composed
- struct {
- **bool_t** secure
- **bool_t** httponly
- **stringer_t** * host
- **stringer_t** * path
- **stringer_t** * application
- } **request**
- struct {
- **ip_t** ip
- **uint64_t** key
- **uint64_t** host
- **uint64_t** stamp
- **uint64_t** number
- **stringer_t** * agent
- **stringer_t** * token
- **void** * cred
- } **warden**
- struct {
- **time_t** stamp
- **uint64_t** web
- } **refs**
- struct {
- **time_t** stamp
- **bool_t** trigger
- } **refresh**
- **pthread_mutex_t** lock

Detailed Description

Definition at line 34 of file sessions.h.

Field Documentation

stringer_t* session_t::agent

Definition at line 56 of file sessions.h.

Referenced by sess_create(), and sess_destroy().

stringer_t* session_t::application

Definition at line 46 of file sessions.h.

Referenced by sess_create(), and sess_destroy().

uint64_t session_t::composed

Definition at line 39 of file sessions.h.

inx_t* session_t::compositions

Definition at line 38 of file sessions.h.

Referenced by sess_create(), and sess_destroy().

void* session_t::cred

Definition at line 58 of file sessions.h.

Referenced by sess_destroy().

uint64_t session_t::host

Definition at line 53 of file sessions.h.

stringer_t* session_t::host

Definition at line 44 of file sessions.h.

Referenced by sess_create(), sess_destroy(), and sess_token().

bool_t session_t::httponly

Definition at line 43 of file sessions.h.

Referenced by sess_create().

ip_t session_t::ip

Definition at line 51 of file sessions.h.

Referenced by sess_create().

uint64_t session_t::key

Definition at line 52 of file sessions.h.

Referenced by sess_create(), and sess_token().

pthread_mutex_t session_t::lock

Definition at line 71 of file sessions.h.

Referenced by sess_create(), sess_destroy(), sess_ref_add(), sess_ref_dec(), sess_ref_stamp(), sess_ref_total(), sess_refresh_check(), sess_refresh_flush(), sess_refresh_stamp(), and sess_trigger().

uint64_t session_t::number

Definition at line 55 of file sessions.h.

Referenced by sess_create(), and sess_token().

stringer_t* session_t::path

Definition at line 45 of file sessions.h.

Referenced by sess_create(), and sess_destroy().

struct { ... } session_t::refresh

Referenced by sess_refresh_check(), sess_refresh_flush(), sess_refresh_stamp(), and sess_trigger().

struct { ... } session_t::refs

Referenced by sess_ref_add(), sess_ref_dec(), sess_ref_stamp(), and sess_ref_total().

struct { ... } session_t::request

Referenced by sess_create(), and sess_destroy().

bool_t session_t::secure

Definition at line 42 of file sessions.h.

Referenced by sess_create().

time_t session_t::stamp

Definition at line 62 of file sessions.h.

uint64_t session_t::stamp

Definition at line 54 of file sessions.h.

Referenced by sess_create(), sess_ref_add(), sess_ref_dec(), sess_ref_stamp(), sess_refresh_check(), sess_refresh_flush(), sess_refresh_stamp(), and sess_token().

int_t session_t::state

Definition at line 36 of file sessions.h.

stringer_t* session_t::token

Definition at line 57 of file sessions.h.

Referenced by sess_create(), and sess_destroy().

bool_t session_t::trigger

Definition at line 68 of file sessions.h.

Referenced by sess_refresh_check(), sess_refresh_flush(), and sess_trigger().

meta_user_t* session_t::user

Definition at line 37 of file sessions.h.

Referenced by sess_destroy(), sess_serial_check(), and sess_update().

struct { ... } session_t::warden

Referenced by sess_create(), sess_destroy(), and sess_token().

uint64_t session_t::web

Definition at line 63 of file sessions.h.

Referenced by sess_ref_add(), sess_ref_dec(), and sess_ref_total().

The documentation for this struct was generated from the following file:

- magma/network/sessions.h

smtp_inbound_filter_t Struct Reference

```
#include <smtp.h>
```

Data Fields

- uint64_t **foldernum**
- uint64_t **rulenum**
- unsigned **location**
- unsigned **type**
- unsigned **action**
- **stringer_t** * **field**
- **stringer_t** * **label**
- **stringer_t** * **expression**

Detailed Description

Definition at line 80 of file smtp.h.

Field Documentation

unsigned smtp_inbound_filter_t::action

Definition at line 82 of file smtp.h.

Referenced by smtp_fetch_inbound().

stringer_t * smtp_inbound_filter_t::expression

Definition at line 83 of file smtp.h.

Referenced by smtp_fetch_inbound(), and smtp_list_free_filter().

stringer_t* smtp_inbound_filter_t::field

Definition at line 83 of file smtp.h.

Referenced by smtp_fetch_inbound(), and smtp_list_free_filter().

uint64_t smtp_inbound_filter_t::foldernum

Definition at line 81 of file smtp.h.

Referenced by smtp_fetch_inbound().

stringer_t * smtp_inbound_filter_t::label

Definition at line 83 of file smtp.h.

Referenced by `smtp_fetch_inbound()`, and `smtp_list_free_filter()`.

unsigned smtp_inbound_filter_t::location

Definition at line 82 of file `smtp.h`.

Referenced by `smtp_fetch_inbound()`.

uint64_t smtp_inbound_filter_t::rulenum

Definition at line 81 of file `smtp.h`.

Referenced by `smtp_fetch_inbound()`.

unsigned smtp_inbound_filter_t::type

Definition at line 82 of file `smtp.h`.

Referenced by `smtp_fetch_inbound()`.

The documentation for this struct was generated from the following file:

- `magma/network/smtp.h`

smtp_inbound_prefs_t Struct Reference

```
#include <smtp.h>
```

Data Fields

- **int_t** outcome
- **size_t** rcv_size_limit
- **stringer_t** * rcptto
- **stringer_t** * address
- **stringer_t** * domain
- **stringer_t** * forwarded
- **stringer_t** * spamsig
- **uint32_t** greytime
- **uint32_t** local_size
- **uint32_t** daily_rcv_limit
- **uint32_t** daily_rcv_limit_ip
- **uint64_t** usernum
- **uint64_t** signum
- **uint64_t** spamkey
- **uint64_t** quota
- **uint64_t** stor_size
- **uint64_t** inbox
- **uint64_t** autoreply
- **uint64_t** messagenum
- **uint64_t** foldernum
- **int_t** mark
- **int_t** secure
- **int_t** rollout
- **int_t** spam
- **int_t** virus
- **int_t** greylist
- **int_t** spf
- **int_t** dkim
- **int_t** rbl
- **int_t** phish
- **int_t** overquota
- **int_t** bounces
- **int_t** spfaction
- **int_t** dkimaction
- **int_t** rblaction
- **int_t** spamaction
- **int_t** virusaction
- **int_t** phishaction
- **int_t** spam_checked
- **inx_t** * filters
- **struct** smtp_inbound_prefs_t * next

Detailed Description

Definition at line 87 of file smtp.h.

Field Documentation

stringer_t * smtp_inbound_prefs_t::address

Definition at line 90 of file smtp.h.

Referenced by smtp_fetch_inbound(), smtp_free_inbound(), and smtp_rcpt_to().

uint64_t smtp_inbound_prefs_t::autoreply

Definition at line 92 of file smtp.h.

Referenced by smtp_accept_message(), and smtp_fetch_inbound().

int_t smtp_inbound_prefs_t::bounces

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), smtp_data_inbound(), smtp_fetch_inbound(), and smtp_update_receive_stats().

uint32_t smtp_inbound_prefs_t::daily_rcv_limit

Definition at line 91 of file smtp.h.

Referenced by smtp_check_receive_quota(), smtp_fetch_inbound(), and smtp_rcpt_to().

uint32_t smtp_inbound_prefs_t::daily_rcv_limit_ip

Definition at line 91 of file smtp.h.

Referenced by smtp_check_receive_quota(), smtp_fetch_inbound(), and smtp_rcpt_to().

int_t smtp_inbound_prefs_t::dkim

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), and smtp_fetch_inbound().

int_t smtp_inbound_prefs_t::dkimaction

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), and smtp_fetch_inbound().

stringer_t * smtp_inbound_prefs_t::domain

Definition at line 90 of file smtp.h.

Referenced by smtp_fetch_inbound(), and smtp_free_inbound().

inx_t* smtp_inbound_prefs_t::filters

Definition at line 95 of file smtp.h.

Referenced by smtp_accept_message(), smtp_check_filters(), smtp_fetch_inbound(), and smtp_free_inbound().

uint64_t smtp_inbound_prefs_t::foldernum

Definition at line 92 of file smtp.h.

Referenced by smtp_accept_message(), smtp_check_filters(), and smtp_store_message().

stringer_t * smtp_inbound_prefs_t::forwarded

Definition at line 90 of file smtp.h.

Referenced by smtp_accept_message(), smtp_fetch_inbound(), smtp_free_inbound(), and smtp_rcpt_to().

int_t smtp_inbound_prefs_t::greylist

Definition at line 93 of file smtp.h.

Referenced by smtp_fetch_inbound(), and smtp_rcpt_to().

uint32_t smtp_inbound_prefs_t::greytime

Definition at line 91 of file smtp.h.

Referenced by smtp_check_greylist(), smtp_fetch_inbound(), and smtp_rcpt_to().

uint64_t smtp_inbound_prefs_t::inbox

Definition at line 92 of file smtp.h.

Referenced by smtp_accept_message(), and smtp_fetch_inbound().

uint32_t smtp_inbound_prefs_t::local_size

Definition at line 91 of file smtp.h.

int_t smtp_inbound_prefs_t::mark

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), smtp_check_filters(), and smtp_store_message().

uint64_t smtp_inbound_prefs_t::messagenum

Definition at line 92 of file smtp.h.

Referenced by smtp_store_message().

struct smtp_inbound_prefs_t* smtp_inbound_prefs_t::next [read]

Definition at line 96 of file smtp.h.

Referenced by mail_add_required_headers(), smtp_add_inbound(), smtp_bounce(), smtp_check_duplicate_recipient(), smtp_data_inbound(), and smtp_free_inbound().

int_t smtp_inbound_prefs_t::outcome

Definition at line 88 of file smtp.h.

Referenced by smtp_bounce(), and smtp_data_inbound().

int_t smtp_inbound_prefs_t::overquota

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), smtp_fetch_inbound(), and smtp_rcpt_to().

int_t smtp_inbound_prefs_t::phish

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), and smtp_fetch_inbound().

int_t smtp_inbound_prefs_t::phishaction

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), and smtp_fetch_inbound().

uint64_t smtp_inbound_prefs_t::quota

Definition at line 92 of file smtp.h.

Referenced by smtp_fetch_inbound(), and smtp_rollout().

int_t smtp_inbound_prefs_t::rbl

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), smtp_fetch_inbound(), and smtp_rcpt_to().

int_t smtp_inbound_prefs_t::rblaction

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), smtp_fetch_inbound(), and smtp_rcpt_to().

stringer_t* smtp_inbound_prefs_t::rcptto

Definition at line 90 of file smtp.h.

Referenced by mail_add_inbound_headers(), mail_add_required_headers(), smtp_accept_message(), smtp_bounce(), smtp_fetch_inbound(), smtp_free_inbound(), and smtp_rcpt_to().

size_t smtp_inbound_prefs_t::recv_size_limit

Definition at line 89 of file smtp.h.

Referenced by smtp_accept_message(), smtp_fetch_inbound(), and smtp_rcpt_to().

int_t smtp_inbound_prefs_t::rollout

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), smtp_fetch_inbound(), and smtp_rcpt_to().

int_t smtp_inbound_prefs_t::secure

Definition at line 93 of file smtp.h.

Referenced by smtp_fetch_inbound(), and smtp_store_message().

uint64_t smtp_inbound_prefs_t::signum

Definition at line 92 of file smtp.h.

Referenced by smtp_accept_message(), smtp_store_message(), and smtp_store_spamsig().

int_t smtp_inbound_prefs_t::spam

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), and smtp_fetch_inbound().

int_t smtp_inbound_prefs_t::spam_checked

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message().

int_t smtp_inbound_prefs_t::spamaction

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), and smtp_fetch_inbound().

uint64_t smtp_inbound_prefs_t::spamkey

Definition at line 92 of file smtp.h.

Referenced by smtp_accept_message(), smtp_store_message(), and smtp_store_spamsig().

stringer_t * smtp_inbound_prefs_t::spamsig

Definition at line 90 of file smtp.h.

Referenced by smtp_accept_message(), smtp_free_inbound(), and smtp_insert_spamsig().

int_t smtp_inbound_prefs_t::spf

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), smtp_fetch_inbound(), and smtp_rcpt_to().

int_t smtp_inbound_prefs_t::spfaction

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), smtp_fetch_inbound(), and smtp_rcpt_to().

uint64_t smtp_inbound_prefs_t::stor_size

Definition at line 92 of file smtp.h.

Referenced by smtp_fetch_inbound(), and smtp_rollout().

uint64_t smtp_inbound_prefs_t::usernum

Definition at line 92 of file smtp.h.

Referenced by smtp_accept_message(), smtp_check_duplicate_recipient(), smtp_check_filters(), smtp_check_greylist(), smtp_check_receive_quota(), smtp_fetch_inbound(), smtp_insert_spamsig(), smtp_rcpt_to(), smtp_rollout(), smtp_store_message(), and smtp_update_receive_stats().

int_t smtp_inbound_prefs_t::virus

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), and smtp_fetch_inbound().

int_t smtp_inbound_prefs_t::virusaction

Definition at line 93 of file smtp.h.

Referenced by smtp_accept_message(), and smtp_fetch_inbound().

The documentation for this struct was generated from the following file:

- magma/network/smtp.h

smtp_message_t Struct Reference

```
#include <smtp.h>
```

Data Fields

- **placer_t** to
 - **placer_t** date
 - **placer_t** from
 - **placer_t** subject
 - **stringer_t** * id
 - **stringer_t** * text
 - **size_t** header_length
-

Detailed Description

Definition at line 64 of file smtp.h.

Field Documentation

placer_t smtp_message_t::date

Definition at line 66 of file smtp.h.

Referenced by mail_add_required_headers(), and mail_headers().

placer_t smtp_message_t::from

Definition at line 67 of file smtp.h.

Referenced by mail_add_required_headers(), mail_headers(), and smtp_data_outbound().

size_t smtp_message_t::header_length

Definition at line 71 of file smtp.h.

Referenced by mail_add_required_headers(), mail_create_message(), and mail_headers().

stringer_t* smtp_message_t::id

Definition at line 69 of file smtp.h.

Referenced by mail_add_inbound_headers(), mail_add_outbound_headers(), mail_create_message(), mail_destroy_message(), and smtp_accept_message().

placer_t smtp_message_t::subject

Definition at line 68 of file smtp.h.

Referenced by mail_add_required_headers(), and mail_headers().

stringer_t* smtp_message_t::text

Definition at line 70 of file smtp.h.

Referenced by mail_add_inbound_headers(), mail_add_outbound_headers(), mail_add_required_headers(), mail_create_message(), mail_destroy_message(), mail_headers(), smtp_accept_message(), smtp_bounce(), smtp_data_outbound(), and smtp_relay_message().

placer_t smtp_message_t::to

Definition at line 65 of file smtp.h.

Referenced by mail_add_required_headers(), and mail_headers().

The documentation for this struct was generated from the following file:

- magma/network/smtp.h

smtp_outbound_prefs_t Struct Reference

```
#include <smtp.h>
```

Data Fields

- `uint64_t` **usernum**
- `stringer_t *` **domain**
- `int_t` **ssl**
- `int_t` **importance**
- `uint32_t` **sent_today**
- `uint32_t` **daily_send_limit**
- `uint32_t` **send_size_limit**
- `smtp_recipients_t *` **recipients**

Detailed Description

Definition at line 100 of file smtp.h.

Field Documentation

`uint32_t` **smtp_outbound_prefs_t::daily_send_limit**

Definition at line 104 of file smtp.h.

Referenced by `smtp_check_transmit_quota()`, `smtp_data_outbound()`, `smtp_fetch_authorization()`, and `smtp_rcpt_to()`.

`stringer_t *` **smtp_outbound_prefs_t::domain**

Definition at line 102 of file smtp.h.

Referenced by `smtp_fetch_authorization()`, and `smtp_free_outbound()`.

`int_t` **smtp_outbound_prefs_t::importance**

Definition at line 103 of file smtp.h.

Referenced by `smtp_fetch_authorization()`, and `smtp_relay_message()`.

`smtp_recipients_t *` **smtp_outbound_prefs_t::recipients**

Definition at line 105 of file smtp.h.

Referenced by `mail_add_outbound_headers()`, `mail_add_required_headers()`, `smtp_add_recipient()`, `smtp_data()`, `smtp_free_outbound()`, `smtp_relay_message()`, and `smtp_session_reset()`.

uint32_t smtp_outbound_prefs_t::send_size_limit

Definition at line 104 of file smtp.h.

Referenced by smtp_auth_login(), smtp_auth_plain(), smtp_fetch_authorization(), and smtp_rcpt_to().

uint32_t smtp_outbound_prefs_t::sent_today

Definition at line 104 of file smtp.h.

Referenced by smtp_check_transmit_quota(), smtp_fetch_authorization(), and smtp_rcpt_to().

int_t smtp_outbound_prefs_t::ssl

Definition at line 103 of file smtp.h.

Referenced by smtp_fetch_authorization(), and smtp_rcpt_to().

uint64_t smtp_outbound_prefs_t::usernum

Definition at line 101 of file smtp.h.

Referenced by smtp_data_outbound(), smtp_fetch_authorization(), and smtp_update_transmission_stats().

The documentation for this struct was generated from the following file:

- magma/network/smtp.h

smtp_recipients_t Struct Reference

```
#include <smtp.h>
```

Data Fields

- **stringer_t * address**
 - **struct smtp_recipients_t * next**
-

Detailed Description

Definition at line 75 of file smtp.h.

Field Documentation

stringer_t* smtp_recipients_t::address

Definition at line 76 of file smtp.h.

Referenced by mail_add_outbound_headers(), mail_add_required_headers(), smtp_add_recipient(), smtp_free_recipients(), and smtp_relay_message().

struct smtp_recipients_t* smtp_recipients_t::next [read]

Definition at line 77 of file smtp.h.

Referenced by mail_add_required_headers(), smtp_add_recipient(), smtp_free_recipients(), and smtp_relay_message().

The documentation for this struct was generated from the following file:

- magma/network/smtp.h

smtp_session_t Struct Reference

```
#include <smtp.h>
```

Data Fields

- **bool_t** esmtp
- **bool_t** submission
- **bool_t** authenticated
- **bool_t** suggested_eight_bit
- **size_t** max_length
- **size_t** num_recipients
- **size_t** suggested_length
- **stringer_t** * helo
- **stringer_t** * mailfrom
- **uint32_t** ip_address_v4
- **struct** {
 - **int_t** rbl
 - **int_t** spf
 - **int_t** dkim
 - **int_t** virus
 - **}** checked
- **smtp_message_t** * message
- **smtp_inbound_prefs_t** * in_prefs
- **smtp_outbound_prefs_t** * out_prefs
- **bool_t** bypass

Detailed Description

Definition at line 108 of file smtp.h.

Field Documentation

bool_t smtp_session_t::authenticated

Definition at line 111 of file smtp.h.

Referenced by mail_add_required_headers(), smtp_auth_login(), smtp_auth_plain(), smtp_data(), smtp_mail_from(), and smtp_rcpt_to().

bool_t smtp_session_t::bypass

Definition at line 134 of file smtp.h.

Referenced by smtp_accept_message(), smtp_check_greylist(), smtp_init(), and smtp_rcpt_to().

struct { ... } smtp_session_t::checked

Referenced by smtp_accept_message(), smtp_bounce(), smtp_rcpt_to(), and smtp_session_reset().

int_t smtp_session_t::dkim

Definition at line 126 of file smtp.h.

Referenced by smtp_accept_message(), smtp_bounce(), and smtp_session_reset().

bool_t smtp_session_t::esmtsp

Definition at line 109 of file smtp.h.

Referenced by mail_add_inbound_headers(), mail_add_outbound_headers(), smtp_ehlo(), and smtp_helo().

stringer_t* smtp_session_t::helo

Definition at line 118 of file smtp.h.

Referenced by mail_add_inbound_headers(), mail_add_outbound_headers(), smtp_data(), smtp_ehlo(), smtp_helo(), smtp_mail_from(), smtp_rcpt_to(), and smtp_session_destroy().

smtp_inbound_prefs_t* smtp_session_t::in_prefs

Definition at line 131 of file smtp.h.

Referenced by mail_add_required_headers(), smtp_add_inbound(), smtp_bounce(), smtp_check_duplicate_recipient(), smtp_data(), smtp_data_inbound(), smtp_session_destroy(), and smtp_session_reset().

uint32_t smtp_session_t::ip_address_v4

Definition at line 121 of file smtp.h.

stringer_t* smtp_session_t::mailfrom

Definition at line 119 of file smtp.h.

Referenced by mail_add_inbound_headers(), mail_add_required_headers(), smtp_accept_message(), smtp_bounce(), smtp_data(), smtp_data_outbound(), smtp_mail_from(), smtp_rcpt_to(), smtp_relay_message(), smtp_session_destroy(), smtp_session_reset(), and smtp_update_receive_stats().

size_t smtp_session_t::max_length

Definition at line 114 of file smtp.h.

Referenced by smtp_auth_login(), smtp_auth_plain(), smtp_data(), smtp_data_read(), smtp_rcpt_to(), and smtp_session_reset().

smtp_message_t* smtp_session_t::message

Definition at line 130 of file smtp.h.

Referenced by mail_add_inbound_headers(), mail_add_outbound_headers(), smtp_accept_message(), smtp_bounce(), smtp_data(), smtp_data_outbound(), smtp_relay_message(), smtp_session_destroy(), and smtp_session_reset().

size_t smtp_session_t::num_recipients

Definition at line 115 of file smtp.h.

Referenced by mail_add_outbound_headers(), smtp_add_inbound(), smtp_add_recipient(), smtp_data_inbound(), smtp_data_outbound(), smtp_rcpt_to(), smtp_session_reset(), and smtp_update_transmission_stats().

smtp_outbound_prefs_t* smtp_session_t::out_prefs

Definition at line 132 of file smtp.h.

Referenced by mail_add_outbound_headers(), mail_add_required_headers(), smtp_add_outbound(), smtp_add_recipient(), smtp_data(), smtp_data_outbound(), smtp_rcpt_to(), smtp_relay_message(), smtp_session_destroy(), smtp_session_reset(), and smtp_update_transmission_stats().

int_t smtp_session_t::rbl

Definition at line 124 of file smtp.h.

Referenced by smtp_accept_message(), smtp_rcpt_to(), and smtp_session_reset().

int_t smtp_session_t::spf

Definition at line 125 of file smtp.h.

Referenced by smtp_accept_message(), smtp_bounce(), smtp_rcpt_to(), and smtp_session_reset().

bool_t smtp_session_t::submission

Definition at line 110 of file smtp.h.

Referenced by submission_init().

bool_t smtp_session_t::suggested_eight_bit

Definition at line 112 of file smtp.h.

Referenced by smtp_parse_mail_from_path(), and smtp_session_reset().

size_t smtp_session_t::suggested_length

Definition at line 116 of file smtp.h.

Referenced by smtp_mail_from(), smtp_parse_mail_from_path(), smtp_rcpt_to(), and smtp_session_reset().

int_t smtp_session_t::virus

Definition at line 127 of file smtp.h.

Referenced by smtp_accept_message(), and smtp_session_reset().

The documentation for this struct was generated from the following file:

- magma/network/smtp.h

stacker_node_t Struct Reference

```
#include <buckets.h>
```

Data Fields

- void * **data**
 - struct **stacker_node_t** * **next**
-

Detailed Description

Definition at line 47 of file buckets.h.

Field Documentation

void* stacker_node_t::data

Definition at line 48 of file buckets.h.

Referenced by `stacker_free()`, `stacker_pop()`, and `stacker_push()`.

struct stacker_node_t* stacker_node_t::next [read]

Definition at line 49 of file buckets.h.

Referenced by `stacker_free()`, `stacker_pop()`, and `stacker_push()`.

The documentation for this struct was generated from the following file:

- magma/core/buckets/**buckets.h**

stacker_t Struct Reference

```
#include <buckets.h>
```

Data Fields

- `uint64_t items`
 - `pthread_mutex_t mutex`
 - `stacker_node_t * list`
 - `stacker_node_t * last`
 - `void(* free_function)(void *data)`
-

Detailed Description

Definition at line 52 of file buckets.h.

Field Documentation

`void(* stacker_t::free_function)(void *data)`

Referenced by `stacker_alloc()`, and `stacker_free()`.

`uint64_t stacker_t::items`

Definition at line 53 of file buckets.h.

Referenced by `stacker_nodes()`, `stacker_pop()`, and `stacker_push()`.

`stacker_node_t * stacker_t::last`

Definition at line 55 of file buckets.h.

Referenced by `stacker_pop()`, and `stacker_push()`.

`stacker_node_t* stacker_t::list`

Definition at line 55 of file buckets.h.

Referenced by `stacker_free()`, `stacker_pop()`, and `stacker_push()`.

`pthread_mutex_t stacker_t::mutex`

Definition at line 54 of file buckets.h.

Referenced by `stacker_alloc()`, `stacker_free()`, `stacker_nodes()`, `stacker_pop()`, and `stacker_push()`.

The documentation for this struct was generated from the following file:

- magma/core/buckets/**buckets.h**

statistics_vp_t Struct Reference

```
#include <statistics.h>
```

Data Fields

- `MYSQL_STMT ** stmt`
 - `uint64_t val`
-

Detailed Description

Definition at line 30 of file `statistics.h`.

Field Documentation

`MYSQL_STMT** statistics_vp_t::stmt`

Definition at line 31 of file `statistics.h`.

Referenced by `statistics_init()`, and `statistics_refresh()`.

`uint64_t statistics_vp_t::val`

Definition at line 32 of file `statistics.h`.

Referenced by `statistics_process()`, and `statistics_refresh()`.

The documentation for this struct was generated from the following file:

- `magma/web/statistics/statistics.h`

subnet_t Struct Reference

```
#include <network.h>
```

Data Fields

- `uint32_t mask`
 - `ip_t address`
-

Detailed Description

Definition at line 24 of file network.h.

Field Documentation

`ip_t subnet_t::address`

Definition at line 26 of file network.h.

Referenced by `ip_matches_subnet()`, and `ip_str_subnet()`.

`uint32_t subnet_t::mask`

Definition at line 25 of file network.h.

Referenced by `ip_matches_subnet()`, and `ip_str_subnet()`.

The documentation for this struct was generated from the following file:

- `magma/network/network.h`

symbol_t Struct Reference

```
#include <providers.h>
```

Data Fields

- char * **name**
 - void ** **pointer**
-

Detailed Description

Definition at line 16 of file providers.h.

Field Documentation

char* symbol_t::name

Definition at line 17 of file providers.h.

void symbol_t::pointer**

Definition at line 18 of file providers.h.

The documentation for this struct was generated from the following file:

- magma/providers/**providers.h**

teacher_data_t Struct Reference

```
#include <teacher.h>
```

Data Fields

- **int_t** **completed**
- **int_t** **disposition**
- **uint64_t** **usernum**
- **uint64_t** **signum**
- **uint64_t** **keynum**
- **stringer_t** * **signature**
- **stringer_t** * **username**
- **stringer_t** * **password**

Detailed Description

Definition at line 17 of file teacher.h.

Field Documentation

int_t **teacher_data_t::completed**

Definition at line 18 of file teacher.h.

Referenced by teacher_data_delete(), teacher_data_get(), teacher_data_save(), and teacher_process().

int_t **teacher_data_t::disposition**

Definition at line 18 of file teacher.h.

Referenced by teacher_data_delete(), teacher_data_fetch(), teacher_data_get(), teacher_data_save(), teacher_print_form(), and teacher_process().

uint64_t **teacher_data_t::keynum**

Definition at line 19 of file teacher.h.

Referenced by teacher_data_fetch(), teacher_data_get(), teacher_data_save(), teacher_print_form(), and teacher_process().

stringer_t * **teacher_data_t::password**

Definition at line 20 of file teacher.h.

Referenced by teacher_add_cookie(), teacher_data_delete(), teacher_data_fetch(), teacher_data_free(), teacher_data_get(), teacher_data_save(), and teacher_process().

stringer_t* teacher_data_t::signature

Definition at line 20 of file teacher.h.

Referenced by teacher_data_delete(), teacher_data_fetch(), teacher_data_free(), teacher_data_get(), teacher_data_save(), and teacher_process().

uint64_t teacher_data_t::signum

Definition at line 19 of file teacher.h.

Referenced by teacher_data_delete(), teacher_data_fetch(), teacher_data_get(), teacher_data_save(), and teacher_print_form().

stringer_t * teacher_data_t::username

Definition at line 20 of file teacher.h.

Referenced by teacher_data_delete(), teacher_data_fetch(), teacher_data_free(), teacher_data_get(), teacher_data_save(), and teacher_process().

uint64_t teacher_data_t::usernum

Definition at line 19 of file teacher.h.

Referenced by teacher_data_delete(), teacher_data_fetch(), teacher_data_get(), teacher_data_save(), and teacher_process().

The documentation for this struct was generated from the following file:

- magma/web/teacher/**teacher.h**

tok_state_t Struct Reference

```
#include <parsers.h>
```

Data Fields

- char **token**
 - char * **block**
 - char * **position**
 - size_t **length**
 - size_t **remaining**
-

Detailed Description

Definition at line 16 of file parsers.h.

Field Documentation

char* tok_state_t::block

Definition at line 18 of file parsers.h.

Referenced by tok_pop_init_bl(), and tok_pop_init_st().

size_t tok_state_t::length

Definition at line 19 of file parsers.h.

Referenced by tok_pop_init_bl(), and tok_pop_init_st().

char * tok_state_t::position

Definition at line 18 of file parsers.h.

Referenced by tok_pop(), tok_pop_init_bl(), and tok_pop_init_st().

size_t tok_state_t::remaining

Definition at line 19 of file parsers.h.

Referenced by tok_pop(), tok_pop_init_bl(), and tok_pop_init_st().

char tok_state_t::token

Definition at line 17 of file parsers.h.

Referenced by tok_pop(), tok_pop_init_bl(), and tok_pop_init_st().

The documentation for this struct was generated from the following file:

- magma/core/parsers/**parsers.h**

File Documentation

magma/core/buckets/arrays.c File Reference

A collection of functions used to create, maintain and safely utilize arrays of pointers.

```
#include "magma.h"
```

Functions

- **array_t * ar_alloc** (size_t size)
- *Allocate an array of a specified number of elements.* size_t **ar_avail_get** (array_t *array)
- *Get the maximum number of elements allocated in an array.* size_t **ar_length_get** (array_t *array)
- *Get the number of elements actively stored in an array.* uint32_t **ar_field_type** (array_t *array, size_t element)
- **stringer_t * ar_field_st** (array_t *array, size_t element)
- *Return the value of an element in an array that holds a managed string.* chr_t * **ar_field_ns** (array_t *array, size_t element)
- *Return the value of an element in an array that holds a null-terminated string.* array_t * **ar_field_ar** (array_t *array, size_t element)
- *Return the value of an element in an array that holds another array.* placer_t * **ar_field_pl** (array_t *array, size_t element)
- *Return the value of an element in an array that holds a placer.* void * **ar_field_ptr** (array_t *array, size_t element)
- *Return the value of an element in an array that holds a pointer.* void **ar_length_set** (array_t *array, size_t used)
- *Manually set the number of used elements in an array.* int_t **ar_append** (array_t **array, uint32_t type, void *item)
- void **ar_free** (array_t *array)
- *Free an array object and all of its underlying elements.* array_t * **ar_dupe** (array_t *array)

Create a duplicate copy of the array.

Detailed Description

A collection of functions used to create, maintain and safely utilize arrays of pointers.

Definition in file **arrays.c**.

Function Documentation

array_t* ar_alloc (size_t size)

Allocate an array of a specified number of elements.

arrays.c

Parameters:

size the number of elements the array is capable of holding.

Returns:

NULL on failure, or a pointer to the newly allocated array on success.

Definition at line 20 of file arrays.c.

References ARRAY_MAX_ELEMENTS, log_error, log_pedantic, and mm_wipe().

Referenced by ar_append(), ar_dupe(), mail_mime_part(), mail_mime_split(), mail_mime_type_parameters(), and meta_data_fetch_message_tags().

int_t ar_append (array_t ** array, uint32_t type, void * item)

Definition at line 300 of file arrays.c.

References ar_alloc(), ar_avail_get(), ar_length_get(), ar_length_set(), ARRAY_TYPE_EMPTY, log_pedantic, mm_free(), and mm_move().

Referenced by ar_dupe(), imap_parse_arguments(), imap_parse_array(), imap_parse_dataitems(), mail_mime_part(), mail_mime_split(), mail_mime_type_parameters(), meta_data_fetch_message_tags(), and portal_smtp_create_data().

size_t ar_avail_get (array_t * array)

Get the maximum number of elements allocated in an array.

Parameters:

array a pointer to the array to be examined.

Returns:

NULL on failure, or the number of elements the array can hold.

Definition at line 57 of file arrays.c.

References log_pedantic.

Referenced by ar_append(), ar_dupe(), ar_free(), and ar_length_set().

array_t* ar_dupe (array_t * array)

Create a duplicate copy of the array.

Note:

The new array might not correspond precisely to the layout of the original.

Deep copies will be made of all elements that are also arrays.. Managed strings will be copied, but neither empty strings nor pointers will be.

Parameters:

array a pointer to the array to be copied.

Returns:

NULL on failure, or a pointer to the new copy of the array on success.

Definition at line 417 of file arrays.c.

References ar_alloc(), ar_append(), ar_avail_get(), ar_dupe(), ARRAY_TYPE_ARRAY, ARRAY_TYPE_EMPTY, ARRAY_TYPE_POINTER, ARRAY_TYPE_STRINGER, st_dupe(), and type().

Referenced by ar_dupe(), and meta_message_dupe().

array_t* ar_field_ar (array_t * array, size_t element)

Return the value of an element in an array that holds another array.

Parameters:

array a pointer to the array to be examined. *element* the index of the element in the array to be queried.

Returns:

NULL on failure, or a pointer to an array object with the element's data on success.

Definition at line 188 of file arrays.c.

References `ar_field_type()`, `ar_length_get()`, `ARRAY_TYPE_ARRAY`, and `log_pedantic`.

Referenced by `imap_fetch_body()`.

`chr_t* ar_field_ns (array_t * array, size_t element)`

Return the value of an element in an array that holds a null-terminated string.

Parameters:

array a pointer to the array to be examined. *element* the index of the element in the array to be queried.

Returns:

NULL on failure, or a pointer to a null-terminated string with the element's data on success.

Definition at line 159 of file arrays.c.

References `ar_field_type()`, `ar_length_get()`, `ARRAY_TYPE_NULLER`, and `log_pedantic`.

`placer_t* ar_field_pl (array_t * array, size_t element)`

Return the value of an element in an array that holds a placer.

Parameters:

array a pointer to the array to be examined. *element* the index of the element in the array to be queried.

Returns:

NULL on failure, or a pointer to a placer with the element's data on success.

Definition at line 217 of file arrays.c.

References `ar_field_type()`, `ar_length_get()`, `ARRAY_TYPE_PLACER`, `log_pedantic`, and `placer_t`.

`void* ar_field_ptr (array_t * array, size_t element)`

Return the value of an element in an array that holds a pointer.

Parameters:

array a pointer to the array to be examined. *element* the index of the element in the array to be queried.

Returns:

NULL on failure, or a generic pointer to the element's data on success.

Definition at line 246 of file arrays.c.

References `ar_field_type()`, `ar_length_get()`, `ARRAY_TYPE_POINTER`, and `log_pedantic`.

Referenced by `imap_fetch_body_part()`, `imap_fetch_bodystructure()`, `mail_mime_free()`, `mail_mime_generate_boundary()`, `portal_message_attachments()`, and `portal_message_body()`.

stringer_t* ar_field_st (array_t * array, size_t element)

Return the value of an element in an array that holds a managed string.

Parameters:

array a pointer to the array to be examined. *element* the index of the element in the array to be queried.

Returns:

NULL on failure, or a pointer to a managed string with the element's data on success.

Definition at line 130 of file `arrays.c`.

References `ar_field_type()`, `ar_length_get()`, `ARRAY_TYPE_STRINGER`, and `log_pedantic`.

Referenced by `imap_fetch_bodystructure()`, `mail_mime_part()`, `meta_folders_stats_tags()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_tag()`, and `portal_message_tags_array()`.

uint32_t ar_field_type (array_t * array, size_t element)

Definition at line 99 of file `arrays.c`.

References `ar_length_get()`, `ARRAY_TYPE_EMPTY`, and `log_pedantic`.

Referenced by `ar_field_ar()`, `ar_field_ns()`, `ar_field_pl()`, `ar_field_ptr()`, `ar_field_st()`, and `imap_fetch_body()`.

void ar_free (array_t * array)

Free an array object and all of its underlying elements.

Note:

Array elements that are managed strings, and aren't empty or of `ARRAY_TYPE_POINTER` will be freed.

Returns:

This function returns no value.

Definition at line 374 of file `arrays.c`.

References `ar_avail_get()`, `ar_free()`, `ARRAY_TYPE_ARRAY`, `ARRAY_TYPE_EMPTY`, `ARRAY_TYPE_PLACER`, `ARRAY_TYPE_POINTER`, `ARRAY_TYPE_STRINGER`, `log_pedantic`, `mm_free()`, `st_free()`, and `type()`.

Referenced by `ar_free()`, `imap_command_parser()`, `imap_fetch_bodystructure()`, `imap_parse_arguments()`, `imap_parse_array()`, `imap_session_destroy()`, `mail_mime_free()`, `mail_mime_part()`, `mail_mime_type_parameters()`, `meta_message_free()`, `portal_endpoint_messages_tag()`, and `portal_smtp_create_data()`.

size_t ar_length_get (array_t * array)

Get the number of elements actively stored in an array.

Parameters:

array a pointer to the array to be counted.

Returns:

NULL on failure, or the number of elements currently held in the array.

Definition at line 76 of file arrays.c.

References `log_pedantic`.

Referenced by `ar_append()`, `ar_field_ar()`, `ar_field_ns()`, `ar_field_pl()`, `ar_field_ptr()`, `ar_field_st()`, `ar_field_type()`, `imap_append()`, `imap_close()`, `imap_copy()`, `imap_create()`, `imap_delete()`, `imap_examine()`, `imap_expunge()`, `imap_fetch()`, `imap_fetch_body()`, `imap_fetch_body_header()`, `imap_fetch_body_tag()`, `imap_fetch_bodystructure()`, `imap_flag_parse()`, `imap_get_ar_ar()`, `imap_get_ptr()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `imap_id()`, `imap_list()`, `imap_login()`, `imap_lsub()`, `imap_parse_dataitems()`, `imap_rename()`, `imap_search()`, `imap_search_messages_inner()`, `imap_select()`, `imap_status()`, `imap_store()`, `imap_subscribe()`, `mail_mime_free()`, `mail_mime_generate_boundary()`, `mail_mime_part()`, `mail_mime_type_parameters()`, `meta_folders_stats_tags()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_tag()`, `portal_message_attachments()`, `portal_message_tags_array()`, and `portal_smtp_create_data()`.

void ar_length_set (array_t * *array*, size_t *used*)

Manually set the number of used elements in an array.

Parameters:

array a pointer to the array to be adjusted.

used the new number of elements (less than the array's maximum size) to be set as the array's used length.

Returns:

This function returns no value.

Definition at line 275 of file arrays.c.

References `ar_avail_get()`, and `log_pedantic`.

Referenced by `ar_append()`.

magma/core/buckets/buckets.h File Reference

The function declarations and types for the thread-safe object pool interface.

Data Structures

- struct **stacker_node_t**
- struct **stacker_t**
- struct **pool_t**

Defines

- #define **MAGMA_CORE_POOL_OBJECTS_LIMIT** 4096
- #define **MAGMA_CORE_POOL_TIMEOUT_LIMIT** 86400
- #define **ARRAY_MAX_ELEMENTS** 16384
- #define **ARRAY_TYPE_EMPTY** 0
- #define **ARRAY_TYPE_ARRAY** 1
- #define **ARRAY_TYPE_STRINGER** 2
- #define **ARRAY_TYPE_SIZER** 3
- #define **ARRAY_TYPE_NULLER** 4
- #define **ARRAY_TYPE_PLACER** 5
- #define **ARRAY_TYPE_POINTER** 6

Typedefs

- typedef char **array_t**
- typedef **M_POOL_STATUS** **status_t**

Enumerations

- enum **M_POOL_STATUS** { **PL_ERROR** = -1, **PL_AVAILABLE** = 0, **PL_RESERVED** = 1 }

Functions

- void **pool_free** (**pool_t** *pool)
- *Free an object pool.* **uint32_t pool_get_count** (**pool_t** *pool)
- *Return the total number of items allocated inside a pool.* **uint32_t pool_get_timeout** (**pool_t** *pool)
- *Get the timeout value for a pool.* **uint64_t pool_get_failures** (**pool_t** *pool)
- *Get the number of failed requests for a pool.* **uint32_t pool_get_available** (**pool_t** *pool)
- *Return the total number of items actively available in a pool.* **pool_t * pool_alloc** (**uint32_t** count, **uint32_t** timeout)
- *Allocate a new object pool.* **status_t pool_get_status** (**pool_t** *pool, **uint32_t** item)
- *Get the status flag for an item in a pool.* **status_t pool_set_status** (**pool_t** *pool, **uint32_t** item, **status_t** status)
- *Set the status flag for an item in a pool.* void **pool_release** (**pool_t** *pool, **uint32_t** item)
- *Return an object to a pool and set its status to PL_AVAILABLE.* void * **pool_get_obj** (**pool_t** *pool, **uint32_t** item)
- *Return a specified object from a pool.* **status_t pool_pull** (**pool_t** *pool, **uint32_t** *item)
- *Return the first available object in a pool.* void * **pool_swap_obj** (**pool_t** *pool, **uint32_t** item, void *object)
- *Wait to acquire the value of an object in a pool, and return the original value while updating it with a new value.* void * **pool_set_obj** (**pool_t** *pool, **uint32_t** item, void *object)
- *Set the object pointer for a given item in a pool.* **array_t * ar_alloc** (**size_t** size)
- **arrays.c** **int_t ar_append** (**array_t** **array, **uint32_t** type, void *item)
- **size_t ar_avail_get** (**array_t** *array)
- *Get the maximum number of elements allocated in an array.* **array_t * ar_dupe** (**array_t** *array)

- Create a duplicate copy of the array. **array_t * ar_field_ar** (**array_t** *array, size_t element)
 - Return the value of an element in an array that holds another array. **chr_t * ar_field_ns** (**array_t** *array, size_t element)
 - Return the value of an element in an array that holds a null-terminated string. **placer_t * ar_field_pl** (**array_t** *array, size_t element)
 - Return the value of an element in an array that holds a placer. **void * ar_field_ptr** (**array_t** *array, size_t element)
 - Return the value of an element in an array that holds a pointer. **stringer_t * ar_field_st** (**array_t** *array, size_t element)
 - Return the value of an element in an array that holds a managed string. **uint32_t ar_field_type** (**array_t** *array, size_t element)
 - **void ar_free** (**array_t** *array)
 - Free an array object and all of its underlying elements. **size_t ar_length_get** (**array_t** *array)
 - Get the number of elements actively stored in an array. **void ar_length_set** (**array_t** *array, size_t used)
 - Manually set the number of used elements in an array. **int_t stacker_push** (**stacker_t** *stack, void *data)
 - **stacked.c stacker_t * stacker_alloc** (void *free_function)
 - Allocate a new instance of a stacked list. **unsigned long stacker_nodes** (**stacker_t** *stack)
 - Get the number of nodes in a stacked list. **void * stacker_pop** (**stacker_t** *stack)
 - Pop the last entry off a stacked list. **void stacker_free** (**stacker_t** *stack)
- Free a stacked list and all of its underlying data nodes.
-

Detailed Description

The function declarations and types for the thread-safe object pool interface.

\$\$Author\$\$ \$\$Date\$\$ \$\$Revision\$\$

Definition in file **buckets.h**.

Define Documentation

#define ARRAY_MAX_ELEMENTS 16384

Definition at line 27 of file buckets.h.

Referenced by `ar_alloc()`.

#define ARRAY_TYPE_ARRAY 1

Definition at line 29 of file buckets.h.

Referenced by `ar_dupe()`, `ar_field_ar()`, `ar_free()`, `imap_fetch_body()`, and `imap_parse_dataitems()`.

#define ARRAY_TYPE_EMPTY 0

Definition at line 28 of file buckets.h.

Referenced by `ar_append()`, `ar_dupe()`, `ar_field_type()`, and `ar_free()`.

#define ARRAY_TYPE_NULLER 4

Definition at line 32 of file buckets.h.

Referenced by `ar_field_ns()`.

#define ARRAY_TYPE_PLACER 5

Definition at line 33 of file `buckets.h`.

Referenced by `ar_field_pl()`, and `ar_free()`.

#define ARRAY_TYPE_POINTER 6

Definition at line 34 of file `buckets.h`.

Referenced by `ar_dupe()`, `ar_field_ptr()`, `ar_free()`, `imap_parse_dataitems()`, `mail_mime_part()`, and `portal_smtp_create_data()`.

#define ARRAY_TYPE_SIZER 3

Definition at line 31 of file `buckets.h`.

#define ARRAY_TYPE_STRINGER 2

Definition at line 30 of file `buckets.h`.

Referenced by `ar_dupe()`, `ar_field_st()`, `ar_free()`, `mail_mime_split()`, `mail_mime_type_parameters()`, and `meta_data_fetch_message_tags()`.

#define MAGMA_CORE_POOL_OBJECTS_LIMIT 4096

The maximum number of objects allowed inside a pool.

Definition at line 19 of file `buckets.h`.

Referenced by `pool_alloc()`, and `sanity_check()`.

#define MAGMA_CORE_POOL_TIMEOUT_LIMIT 86400

The maximum number of seconds a pool can be configured to wait in seconds.

Definition at line 24 of file `buckets.h`.

Referenced by `pool_alloc()`, and `sanity_check()`.

Typedef Documentation

typedef char array_t

Definition at line 37 of file `buckets.h`.

typedef M_POOL_STATUS status_t

Definition at line 45 of file `buckets.h`.

Enumeration Type Documentation

enum M_POOL_STATUS

Enumerator:

PL_ERROR
PL_AVAILABLE
PL_RESERVED

Definition at line 39 of file buckets.h.

Function Documentation

array_t* ar_alloc (size_t size)

arrays.c

arrays.c

Parameters:

size the number of elements the array is capable of holding.

Returns:

NULL on failure, or a pointer to the newly allocated array on success.

Definition at line 20 of file arrays.c.

References ARRAY_MAX_ELEMENTS, log_error, log_pedantic, and mm_wipe().

Referenced by ar_append(), ar_dupe(), mail_mime_part(), mail_mime_split(), mail_mime_type_parameters(), and meta_data_fetch_message_tags().

int_t ar_append (array_t ** array, uint32_t type, void * item)

Definition at line 300 of file arrays.c.

References ar_alloc(), ar_avail_get(), ar_length_get(), ar_length_set(), ARRAY_TYPE_EMPTY, log_pedantic, mm_free(), and mm_move().

Referenced by ar_dupe(), imap_parse_arguments(), imap_parse_array(), imap_parse_dataitems(), mail_mime_part(), mail_mime_split(), mail_mime_type_parameters(), meta_data_fetch_message_tags(), and portal_smtp_create_data().

size_t ar_avail_get (array_t * array)

Get the maximum number of elements allocated in an array.

Parameters:

array a pointer to the array to be examined.

Returns:

NULL on failure, or the number of elements the array can hold.

Definition at line 57 of file arrays.c.

References log_pedantic.

Referenced by ar_append(), ar_dupe(), ar_free(), and ar_length_set().

array_t* ar_dupe (array_t * array)

Create a duplicate copy of the array.

Note:

The new array might not correspond precisely to the layout of the original.

Deep copies will be made of all elements that are also arrays.. Managed strings will be copied, but neither empty strings nor pointers will be.

Parameters:

array a pointer to the array to be copied.

Returns:

NULL on failure, or a pointer to the new copy of the array on success.

Definition at line 417 of file arrays.c.

References ar_alloc(), ar_append(), ar_avail_get(), ar_dupe(), ARRAY_TYPE_ARRAY, ARRAY_TYPE_EMPTY, ARRAY_TYPE_POINTER, ARRAY_TYPE_STRINGER, st_dupe(), and type().

Referenced by ar_dupe(), and meta_message_dupe().

array_t* ar_field_ar (array_t * array, size_t element)

Return the value of an element in an array that holds another array.

Parameters:

array a pointer to the array to be examined. *element* the index of the element in the array to be queried.

Returns:

NULL on failure, or a pointer to an array object with the element's data on success.

Definition at line 188 of file arrays.c.

References ar_field_type(), ar_length_get(), ARRAY_TYPE_ARRAY, and log_pedantic.

Referenced by imap_fetch_body().

chr_t* ar_field_ns (array_t * array, size_t element)

Return the value of an element in an array that holds a null-terminated string.

Parameters:

array a pointer to the array to be examined. *element* the index of the element in the array to be queried.

Returns:

NULL on failure, or a pointer to a null-terminated string with the element's data on success.

Definition at line 159 of file arrays.c.

References ar_field_type(), ar_length_get(), ARRAY_TYPE_NULLER, and log_pedantic.

placer_t* ar_field_pl (array_t * array, size_t element)

Return the value of an element in an array that holds a placer.

Parameters:

array a pointer to the array to be examined. *element* the index of the element in the array to be queried.

Returns:

NULL on failure, or a pointer to a placer with the element's data on success.

Definition at line 217 of file arrays.c.

References ar_field_type(), ar_length_get(), ARRAY_TYPE_PLACER, log_pedantic, and placer_t.

void* ar_field_ptr (array_t * array, size_t element)

Return the value of an element in an array that holds a pointer.

Parameters:

array a pointer to the array to be examined. *element* the index of the element in the array to be queried.

Returns:

NULL on failure, or a generic pointer to the element's data on success.

Definition at line 246 of file arrays.c.

References ar_field_type(), ar_length_get(), ARRAY_TYPE_POINTER, and log_pedantic.

Referenced by imap_fetch_body_part(), imap_fetch_bodystructure(), mail_mime_free(), mail_mime_generate_boundary(), portal_message_attachments(), and portal_message_body().

stringer_t* ar_field_st (array_t * array, size_t element)

Return the value of an element in an array that holds a managed string.

Parameters:

array a pointer to the array to be examined. *element* the index of the element in the array to be queried.

Returns:

NULL on failure, or a pointer to a managed string with the element's data on success.

Definition at line 130 of file arrays.c.

References ar_field_type(), ar_length_get(), ARRAY_TYPE_STRINGER, and log_pedantic.

Referenced by imap_fetch_bodystructure(), mail_mime_part(), meta_folders_stats_tags(), portal_endpoint_messages_list(), portal_endpoint_messages_tag(), and portal_message_tags_array().

uint32_t ar_field_type (array_t * array, size_t element)

Definition at line 99 of file arrays.c.

References ar_length_get(), ARRAY_TYPE_EMPTY, and log_pedantic.

Referenced by ar_field_ar(), ar_field_ns(), ar_field_pl(), ar_field_ptr(), ar_field_st(), and imap_fetch_body().

void ar_free (array_t * array)

Free an array object and all of its underlying elements.

Note:

Array elements that are managed strings, and aren't empty or of ARRAY_TYPE_POINTER will be freed.

Returns:

This function returns no value.

Definition at line 374 of file arrays.c.

References ar_avail_get(), ar_free(), ARRAY_TYPE_ARRAY, ARRAY_TYPE_EMPTY, ARRAY_TYPE_PLACER, ARRAY_TYPE_POINTER, ARRAY_TYPE_STRINGER, log_pedantic, mm_free(), st_free(), and type().

Referenced by ar_free(), imap_command_parser(), imap_fetch_bodystructure(), imap_parse_arguments(), imap_parse_array(), imap_session_destroy(), mail_mime_free(), mail_mime_part(), mail_mime_type_parameters(), meta_message_free(), portal_endpoint_messages_tag(), and portal_smtp_create_data().

size_t ar_length_get (array_t * array)

Get the number of elements actively stored in an array.

Parameters:

array a pointer to the array to be counted.

Returns:

NULL on failure, or the number of elements currently held in the array.

Definition at line 76 of file arrays.c.

References log_pedantic.

Referenced by ar_append(), ar_field_ar(), ar_field_ns(), ar_field_pl(), ar_field_ptr(), ar_field_st(), ar_field_type(), imap_append(), imap_close(), imap_copy(), imap_create(), imap_delete(), imap_examine(), imap_expunge(), imap_fetch(), imap_fetch_body(), imap_fetch_body_header(), imap_fetch_body_tag(), imap_fetch_bodystructure(), imap_flag_parse(), imap_get_ar_ar(), imap_get_ptr(), imap_get_st_ar(), imap_get_type_ar(), imap_id(), imap_list(), imap_login(), imap_lsub(), imap_parse_dataitems(), imap_rename(), imap_search(), imap_search_messages_inner(), imap_select(), imap_status(), imap_store(), imap_subscribe(), mail_mime_free(), mail_mime_generate_boundary(), mail_mime_part(), mail_mime_type_parameters(), meta_folders_stats_tags(), portal_endpoint_messages_list(), portal_endpoint_messages_tag(), portal_message_attachments(), portal_message_tags_array(), and portal_smtp_create_data().

void ar_length_set (array_t * array, size_t used)

Manually set the number of used elements in an array.

Parameters:

array a pointer to the array to be adjusted.

used the new number of elements (less than the array's maximum size) to be set as the array's used length.

Returns:

This function returns no value.

Definition at line 275 of file arrays.c.

References `ar_avail_get()`, and `log_pedantic`.

Referenced by `ar_append()`.

`pool_t* pool_alloc (uint32_t count, uint32_t timeout)`

Allocate a new object pool.

Parameters:

count the number of items the pool can hold.

timeout a timeout for pool requests in seconds (specify 0 for infinite wait).

Returns:

NULL on failure,

Definition at line 38 of file pool.c.

References `pool_t::available`, `pool_t::count`, `pool_t::lock`, `log_info`, `MAGMA_CORE_POOL_OBJECTS_LIMIT`, `MAGMA_CORE_POOL_TIMEOUT_LIMIT`, `mm_alloc()`, `mm_free()`, `mutex_init()`, `pool_t::objects`, `pool_t::status`, and `pool_t::timeout`.

Referenced by `cache_start()`, `spf_start()`, and `sql_start()`.

`void pool_free (pool_t * pool)`

Free an object pool.

Warning:

This function will not free the underlying objects contained by the pool!

Parameters:

pool the pool to be released from memory.

Returns:

This function returns no value.

Definition at line 21 of file pool.c.

References `pool_t::available`, `pool_t::lock`, `mm_free()`, and `mutex_destroy()`.

Referenced by `cache_stop()`, `spf_stop()`, and `sql_stop()`.

`uint32_t pool_get_available (pool_t * pool)`

Return the total number of items actively available in a pool.

Parameters:

pool the pool to be queried.

Returns:

0 on failure, or the total number of items actively available in the specified pool.

Definition at line 99 of file pool.c.

References `pool_t::available`.

uint32_t pool_get_count (pool_t * *pool*)

Return the total number of items allocated inside a pool.

Note:

Some of the item slots may be unused, but remain reserved.

Parameters:

pool the pool to be queried.

Returns:

0 on failure, or the allocation count of the pool on success.

Definition at line 88 of file `pool.c`.

References `pool_t::count`.

Referenced by `pool_get_obj()`, and `pool_set_obj()`.

uint64_t pool_get_failures (pool_t * *pool*)

Get the number of failed requests for a pool.

Note:

Most of the time, a failure will correspond to a timeout, but can also be triggered by other error conditions.

Parameters:

pool a pointer to the pool to be examined.

Returns:

the number of failed requests made on the specified pool.

Definition at line 124 of file `pool.c`.

References `pool_t::failures`, `pool_t::lock`, `mutex_lock()`, and `mutex_unlock()`.

void* pool_get_obj (pool_t * *pool*, uint32_t *item*)

Return a specified object from a pool.

Parameters:

pool the pool containing the object.

item the identifier of the object to query.

Returns:

NULL on failure, or the object corresponding to the specified item number.

Definition at line 246 of file `pool.c`.

References `pool_t::lock`, `log_pedantic`, `mutex_lock()`, `mutex_unlock()`, `pool_t::objects`, and `pool_get_count()`.

Referenced by `cache_add()`, `cache_append()`, `cache_delete()`, `cache_flush()`, `cache_get()`, `cache_get_u64()`, `cache_increment()`, `cache_set()`, `cache_set_u64()`, `cache_silent_add()`, `cache_stop()`, `dspam_check()`, `dspam_train()`, `mail_db_update_message_folder()`, `pool_swap_obj()`, `spf_check()`, `spf_stop()`, `sql_ping()`, `sql_query_conn()`, `sql_stop()`, `stmt_rebuild()`, `stmt_start()`, `tran_commit()`, `tran_rollback()`, and `tran_start()`.

status_t pool_get_status (pool_t * *pool*, uint32_t *item*)

Get the status flag for an item in a pool.

Note:

A value of PL_AVAILABLE indicates the object is available for use,; PL_RESERVED indicates the object is in use by a worker thread.

Parameters:

pool the pool containing the specified item.
item the identifier of the item to be queried.

Returns:

PL_ERROR on failure; otherwise, the status of the specified item.

Definition at line 144 of file pool.c.

References log_check, log_pedantic, PL_AVAILABLE, PL_ERROR, PL_RESERVED, and pool_t::status.

Referenced by pool_pull(), and pool_swap_obj().

uint32_t pool_get_timeout (pool_t * *pool*)

Get the timeout value for a pool.

Note:

A timeout of zero will cause threads to wait forever.

Parameters:

pool a pointer to the pool to be examined.

Returns:

the timeout value of the specified pool, in seconds.

Definition at line 112 of file pool.c.

References pool_t::timeout.

status_t pool_pull (pool_t * *pool*, uint32_t * *item*)

Return the first available object in a pool.

Note:

If no object can be returned immediately, wait for the pool's configured timeout value, in seconds, for an object to become available. If the timeout is zero, wait indefinitely.

Parameters:

item A pointer to a number that will store the zero-based indexed of the first available item in the pool.

Returns:

PL_RESERVED on success or PL_ERROR if an object couldn't be reserved.

Definition at line 183 of file pool.c.

References pool_t::available, pool_t::count, pool_t::failures, pool_t::lock, mutex_lock(), mutex_unlock(), PL_AVAILABLE, PL_ERROR, PL_RESERVED, pool_get_status(), pool_set_status(), and pool_t::timeout.

Referenced by `cache_add()`, `cache_append()`, `cache_delete()`, `cache_flush()`, `cache_get()`, `cache_get_u64()`, `cache_increment()`, `cache_set()`, `cache_set_u64()`, `cache_silent_add()`, `dspam_check()`, `dspam_train()`, `spf_check()`, `sql_query()`, `stmt_exec()`, `stmt_exec_affected()`, `stmt_get_result()`, `stmt_insert()`, and `tran_start()`.

void pool_release (pool_t * *pool*, uint32_t *item*)

Return an object to a pool and set its status to PL_AVAILABLE.

Parameters:

pool the pool tracking the returned item.
item the identifier of the item being returned.

Returns:

This function returns no value.

Definition at line 231 of file `pool.c`.

References `pool_t::available`, `pool_t::lock`, `mutex_lock()`, `mutex_unlock()`, `PL_AVAILABLE`, and `pool_set_status()`.

Referenced by `cache_add()`, `cache_append()`, `cache_delete()`, `cache_flush()`, `cache_get()`, `cache_get_u64()`, `cache_increment()`, `cache_set()`, `cache_set_u64()`, `cache_silent_add()`, `dspam_check()`, `dspam_train()`, `spf_check()`, `sql_query()`, `stmt_exec()`, `stmt_exec_affected()`, `stmt_get_result()`, `stmt_insert()`, `tran_commit()`, `tran_rollback()`, and `tran_start()`.

void* pool_set_obj (pool_t * *pool*, uint32_t *item*, void * *object*)

Set the object pointer for a given item in a pool.

Parameters:

pool the pool containing the specified item.
item the item id to be adjusted.
object the new value of the object corresponding to the specified item.

Returns:

NULL on failure, or a pointer to the object that was set.

Definition at line 272 of file `pool.c`.

References `pool_t::lock`, `log_pedantic`, `mutex_lock()`, `mutex_unlock()`, `pool_t::objects`, and `pool_get_count()`.

Referenced by `cache_start()`, `pool_swap_obj()`, `spf_start()`, and `sql_start()`.

status_t pool_set_status (pool_t * *pool*, uint32_t *item*, status_t *status*)

Set the status flag for an item in a pool.

Note:

A value of PL_AVAILABLE indicates the object is available for use,; PL_RESERVED indicates the object is in use by a worker thread.

Parameters:

pool the pool containing the specified item.
item the identifier of the item to be adjusted.

status the new status for the item.

Returns:

PL_ERROR on failure; otherwise, the new status value of the specified item.

Definition at line 164 of file pool.c.

References log_check, log_pedantic, PL_AVAILABLE, PL_ERROR, PL_RESERVED, and pool_t::status.

Referenced by pool_pull(), and pool_release().

void* pool_swap_obj (pool_t * *pool*, uint32_t *item*, void * *object*)

Wait to acquire the value of an object in a pool, and return the original value while updating it with a new value.

Parameters:

pool a pointer to the pool to be modified.

item the zero-based index of the object in the pool to be updated.

object the new value of the object corresponding to the specified item.

Returns:

a pointer to the original pool object's value, or NULL on failure.

LOW: Currently the function loops until the requested object is available. A superior implementation would hook into the release function and detect when the desired object is available and perform the swap at that point.

Definition at line 298 of file pool.c.

References pool_t::lock, mutex_lock(), mutex_unlock(), PL_AVAILABLE, pool_get_obj(), pool_get_status(), and pool_set_obj().

stacker_t* stacker_alloc (void * *free_function*)

Allocate a new instance of a stacked list.

Parameters:

free_function if not NULL, a pointer to a function that will be used to free the data underlying each node in the stacked list.

Returns:

NULL on failure, or a pointer to the newly created stack list on success.

Definition at line 51 of file stacked.c.

References stacker_t::free_function, log_pedantic, mm_alloc(), mm_free(), stacker_t::mutex, and mutex_init().

void stacker_free (stacker_t * *stack*)

Free a stacked list and all of its underlying data nodes.

Parameters:

stack a pointer to the stacked list to be freed.

Returns:

This function returns no value.

Definition at line 20 of file `stacked.c`.

References `stacker_node_t::data`, `stacker_t::free_function`, `stacker_t::list`, `mm_free()`, `stacker_t::mutex`, `mutex_destroy()`, and `stacker_node_t::next`.

unsigned long stacker_nodes (stacker_t * *stack*)

Get the number of nodes in a stacked list.

Parameters:

stack a pointer to the stacked list to be queried.

Returns:

the number of nodes currently held by the specified stacked list.

Definition at line 80 of file `stacked.c`.

References `stacker_t::items`, `stacker_t::mutex`, `mutex_lock()`, and `mutex_unlock()`.

void* stacker_pop (stacker_t * *stack*)

Pop the last entry off a stacked list.

Parameters:

stack a pointer to the stacked list to be queried.

Returns:

NULL on failure or the value of the last node in the stacked list.

Definition at line 146 of file `stacked.c`.

References `stacker_node_t::data`, `stacker_t::items`, `stacker_t::last`, `stacker_t::list`, `mm_free()`, `stacker_t::mutex`, `mutex_lock()`, `mutex_unlock()`, and `stacker_node_t::next`.

int_t stacker_push (stacker_t * *stack*, void * *data*)

`stacked.c`

`stacked.c`

Parameters:

stack a pointer to the stacked list to be updated.

data a pointer to the data to associated with the new stacked list node.

Returns:

0 on failure or 1 on success.

Definition at line 101 of file `stacked.c`.

References `stacker_node_t::data`, `stacker_t::items`, `stacker_t::last`, `stacker_t::list`, `log_error`, `log_pedantic`, `mm_alloc()`, `stacker_t::mutex`, `mutex_lock()`, `mutex_unlock()`, and `stacker_node_t::next`.

magma/core/buckets/pool.c File Reference

A collection of functions used to create, maintain and safely utilize collections of object pointers that are accessed by multiple threads.

```
#include "magma.h"
```

Functions

- void **pool_free** (**pool_t** *pool)
- *Free an object pool.* **pool_t** * **pool_alloc** (uint32_t count, uint32_t timeout)
- *Allocate a new object pool.* uint32_t **pool_get_count** (**pool_t** *pool)
- *Return the total number of items allocated inside a pool.* uint32_t **pool_get_available** (**pool_t** *pool)
- *Return the total number of items actively available in a pool.* uint32_t **pool_get_timeout** (**pool_t** *pool)
- *Get the timeout value for a pool.* uint64_t **pool_get_failures** (**pool_t** *pool)
- *Get the number of failed requests for a pool.* **status_t** **pool_get_status** (**pool_t** *pool, uint32_t item)
- *Get the status flag for an item in a pool.* **status_t** **pool_set_status** (**pool_t** *pool, uint32_t item, **status_t** status)
- *Set the status flag for an item in a pool.* **status_t** **pool_pull** (**pool_t** *pool, uint32_t *item)
- *Return the first available object in a pool.* void **pool_release** (**pool_t** *pool, uint32_t item)
- *Return an object to a pool and set its status to PL_AVAILABLE.* void * **pool_get_obj** (**pool_t** *pool, uint32_t item)
- *Return a specified object from a pool.* void * **pool_set_obj** (**pool_t** *pool, uint32_t item, void *object)
- *Set the object pointer for a given item in a pool.* void * **pool_swap_obj** (**pool_t** *pool, uint32_t item, void *object)

Wait to acquire the value of an object in a pool, and return the original value while updating it with a new value.

Detailed Description

A collection of functions used to create, maintain and safely utilize collections of object pointers that are accessed by multiple threads.

Definition in file **pool.c**.

Function Documentation

pool_t* **pool_alloc** (uint32_t count, uint32_t timeout)

Allocate a new object pool.

Parameters:

count the number of items the pool can hold.

timeout a timeout for pool requests in seconds (specify 0 for infinite wait).

Returns:

NULL on failure,

Definition at line 38 of file pool.c.

References pool_t::available, pool_t::count, pool_t::lock, log_info, MAGMA_CORE_POOL_OBJECTS_LIMIT, MAGMA_CORE_POOL_TIMEOUT_LIMIT, mm_alloc(), mm_free(), mutex_init(), pool_t::objects, pool_t::status, and pool_t::timeout.

Referenced by `cache_start()`, `spf_start()`, and `sql_start()`.

void pool_free (pool_t * *pool*)

Free an object pool.

Warning:

This function will not free the underlying objects contained by the pool!

Parameters:

pool the pool to be released from memory.

Returns:

This function returns no value.

Definition at line 21 of file `pool.c`.

References `pool_t::available`, `pool_t::lock`, `mm_free()`, and `mutex_destroy()`.

Referenced by `cache_stop()`, `spf_stop()`, and `sql_stop()`.

uint32_t pool_get_available (pool_t * *pool*)

Return the total number of items actively available in a pool.

Parameters:

pool the pool to be queried.

Returns:

0 on failure, or the total number of items actively available in the specified pool.

Definition at line 99 of file `pool.c`.

References `pool_t::available`.

uint32_t pool_get_count (pool_t * *pool*)

Return the total number of items allocated inside a pool.

Note:

Some of the item slots may be unused, but remain reserved.

Parameters:

pool the pool to be queried.

Returns:

0 on failure, or the allocation count of the pool on success.

Definition at line 88 of file `pool.c`.

References `pool_t::count`.

Referenced by `pool_get_obj()`, and `pool_set_obj()`.

uint64_t pool_get_failures (pool_t * *pool*)

Get the number of failed requests for a pool.

Note:

Most of the time, a failure will correspond to a timeout, but can also be triggered by other error conditions.

Parameters:

pool a pointer to the pool to be examined.

Returns:

the number of failed requests made on the specified pool.

Definition at line 124 of file pool.c.

References pool_t::failures, pool_t::lock, mutex_lock(), and mutex_unlock().

void* pool_get_obj (pool_t * *pool*, uint32_t *item*)

Return a specified object from a pool.

Parameters:

pool the pool containing the object.

item the identifier of the object to query.

Returns:

NULL on failure, or the object corresponding to the specified item number.

Definition at line 246 of file pool.c.

References pool_t::lock, log_pedantic, mutex_lock(), mutex_unlock(), pool_t::objects, and pool_get_count().

Referenced by cache_add(), cache_append(), cache_delete(), cache_flush(), cache_get(), cache_get_u64(), cache_increment(), cache_set(), cache_set_u64(), cache_silent_add(), cache_stop(), dspam_check(), dspam_train(), mail_db_update_message_folder(), pool_swap_obj(), spf_check(), spf_stop(), sql_ping(), sql_query_conn(), sql_stop(), stmt_rebuild(), stmt_start(), tran_commit(), tran_rollback(), and tran_start().

status_t pool_get_status (pool_t * *pool*, uint32_t *item*)

Get the status flag for an item in a pool.

Note:

A value of PL_AVAILABLE indicates the object is available for use,; PL_RESERVED indicates the object is in use by a worker thread.

Parameters:

pool the pool containing the specified item.

item the identifier of the item to be queried.

Returns:

PL_ERROR on failure; otherwise, the status of the specified item.

Definition at line 144 of file pool.c.

References log_check, log_pedantic, PL_AVAILABLE, PL_ERROR, PL_RESERVED, and pool_t::status.

Referenced by pool_pull(), and pool_swap_obj().

uint32_t pool_get_timeout (pool_t * *pool*)

Get the timeout value for a pool.

Note:

A timeout of zero will cause threads to wait forever.

Parameters:

pool a pointer to the pool to be examined.

Returns:

the timeout value of the specified pool, in seconds.

Definition at line 112 of file pool.c.

References pool_t::timeout.

status_t pool_pull (pool_t * *pool*, uint32_t * *item*)

Return the first available object in a pool.

Note:

If no object can be returned immediately, wait for the pool's configured timeout value, in seconds, for an object to become available. If the timeout is zero, wait indefinitely.

Parameters:

item A pointer to a number that will store the zero-based indexed of the first available item in the pool.

Returns:

PL_RESERVED on success or PL_ERROR if an object couldn't be reserved.

Definition at line 183 of file pool.c.

References pool_t::available, pool_t::count, pool_t::failures, pool_t::lock, mutex_lock(), mutex_unlock(), PL_AVAILABLE, PL_ERROR, PL_RESERVED, pool_get_status(), pool_set_status(), and pool_t::timeout.

Referenced by cache_add(), cache_append(), cache_delete(), cache_flush(), cache_get(), cache_get_u64(), cache_increment(), cache_set(), cache_set_u64(), cache_silent_add(), dspam_check(), dspam_train(), spf_check(), sql_query(), stmt_exec(), stmt_exec_affected(), stmt_get_result(), stmt_insert(), and tran_start().

void pool_release (pool_t * *pool*, uint32_t *item*)

Return an object to a pool and set its status to PL_AVAILABLE.

Parameters:

pool the pool tracking the returned item.

item the identifier of the item being returned.

Returns:

This function returns no value.

Definition at line 231 of file pool.c.

References pool_t::available, pool_t::lock, mutex_lock(), mutex_unlock(), PL_AVAILABLE, and pool_set_status().

Referenced by `cache_add()`, `cache_append()`, `cache_delete()`, `cache_flush()`, `cache_get()`, `cache_get_u64()`, `cache_increment()`, `cache_set()`, `cache_set_u64()`, `cache_silent_add()`, `dspam_check()`, `dspam_train()`, `spf_check()`, `sql_query()`, `stmt_exec()`, `stmt_exec_affected()`, `stmt_get_result()`, `stmt_insert()`, `tran_commit()`, `tran_rollback()`, and `tran_start()`.

`void* pool_set_obj (pool_t * pool, uint32_t item, void * object)`

Set the object pointer for a given item in a pool.

Parameters:

pool the pool containing the specified item.

item the item id to be adjusted.

object the new value of the object corresponding to the specified item.

Returns:

NULL on failure, or a pointer to the object that was set.

Definition at line 272 of file `pool.c`.

References `pool_t::lock`, `log_pedantic`, `mutex_lock()`, `mutex_unlock()`, `pool_t::objects`, and `pool_get_count()`.

Referenced by `cache_start()`, `pool_swap_obj()`, `spf_start()`, and `sql_start()`.

`status_t pool_set_status (pool_t * pool, uint32_t item, status_t status)`

Set the status flag for an item in a pool.

Note:

A value of `PL_AVAILABLE` indicates the object is available for use,; `PL_RESERVED` indicates the object is in use by a worker thread.

Parameters:

pool the pool containing the specified item.

item the identifier of the item to be adjusted.

status the new status for the item.

Returns:

`PL_ERROR` on failure; otherwise, the new status value of the specified item.

Definition at line 164 of file `pool.c`.

References `log_check`, `log_pedantic`, `PL_AVAILABLE`, `PL_ERROR`, `PL_RESERVED`, and `pool_t::status`.

Referenced by `pool_pull()`, and `pool_release()`.

`void* pool_swap_obj (pool_t * pool, uint32_t item, void * object)`

Wait to acquire the value of an object in a pool, and return the original value while updating it with a new value.

Parameters:

pool a pointer to the pool to be modified.

item the zero-based index of the object in the pool to be updated.

object the new value of the object corresponding to the specified item.

Returns:

a pointer to the original pool object's value, or NULL on failure.

LOW: Currently the function loops until the requested object is available. A superior implementation would hook into the release function and detect when the desired object is available and perform the swap at that point.

Definition at line 298 of file pool.c.

References pool_t::lock, mutex_lock(), mutex_unlock(), PL_AVAILABLE, pool_get_obj(), pool_get_status(), and pool_set_obj().

magma/core/buckets/stacked.c File Reference

An interface for handling FIFO stacks.

```
#include "magma.h"
```

Functions

- void **stacker_free** (**stacker_t** *stack)
- *Free a stacked list and all of its underlying data nodes.* **stacker_t** * **stacker_alloc** (void *free_function)
- *Allocate a new instance of a stacked list.* uint64_t **stacker_nodes** (**stacker_t** *stack)
- *Get the number of nodes in a stacked list.* int_t **stacker_push** (**stacker_t** *stack, void *data)
- *Push a new entry onto the end of a stacked list.* void * **stacker_pop** (**stacker_t** *stack)

Pop the last entry off a stacked list.

Detailed Description

An interface for handling FIFO stacks.

Definition in file **stacked.c**.

Function Documentation

stacker_t* **stacker_alloc** (void * *free_function*)

Allocate a new instance of a stacked list.

Parameters:

free_function if not NULL, a pointer to a function that will be used to free the data underlying each node in the stacked list.

Returns:

NULL on failure, or a pointer to the newly created stack list on success.

Definition at line 51 of file stacked.c.

References `stacker_t::free_function`, `log_pedantic`, `mm_alloc()`, `mm_free()`, `stacker_t::mutex`, and `mutex_init()`.

void stacker_free (**stacker_t** * *stack*)

Free a stacked list and all of its underlying data nodes.

Parameters:

stack a pointer to the stacked list to be freed.

Returns:

This function returns no value.

Definition at line 20 of file stacked.c.

References `stacker_node_t::data`, `stacker_t::free_function`, `stacker_t::list`, `mm_free()`, `stacker_t::mutex`, `mutex_destroy()`, and `stacker_node_t::next`.

uint64_t stacker_nodes (stacker_t * *stack*)

Get the number of nodes in a stacked list.

Parameters:

stack a pointer to the stacked list to be queried.

Returns:

the number of nodes currently held by the specified stacked list.

Definition at line 80 of file stacked.c.

References stacker_t::items, stacker_t::mutex, mutex_lock(), and mutex_unlock().

void* stacker_pop (stacker_t * *stack*)

Pop the last entry off a stacked list.

Parameters:

stack a pointer to the stacked list to be queried.

Returns:

NULL on failure or the value of the last node in the stacked list.

Definition at line 146 of file stacked.c.

References stacker_node_t::data, stacker_t::items, stacker_t::last, stacker_t::list, mm_free(), stacker_t::mutex, mutex_lock(), mutex_unlock(), and stacker_node_t::next.

int_t stacker_push (stacker_t * *stack*, void * *data*)

Push a new entry onto the end of a stacked list.

stacked.c

Parameters:

stack a pointer to the stacked list to be updated.

data a pointer to the data to associated with the new stacked list node.

Returns:

0 on failure or 1 on success.

Definition at line 101 of file stacked.c.

References stacker_node_t::data, stacker_t::items, stacker_t::last, stacker_t::list, log_error, log_pedantic, mm_alloc(), stacker_t::mutex, mutex_lock(), mutex_unlock(), and stacker_node_t::next.

magma/core/classify/ascii.c File Reference

Functions used to classify ASCII characters.

```
#include "magma.h"
```

Functions

- **bool_t chr_ascii (uchr_t c)**
- *Determine whether a specified character is an ASCII character. **bool_t chr_printable (uchr_t c)***
- *Determine whether a specified character is a printable character. **bool_t chr_alphanumeric (uchr_t c)***
- *Determine whether a specified character is alpha-numeric. **bool_t chr_lower (uchr_t c)***
- *Determine whether a specified character is a lowercase character. **bool_t chr_upper (uchr_t c)***
- *Determine whether a specified character is an uppercase character. **bool_t chr_numeric (uchr_t c)***
- *Determine whether a specified character is a numeric character. **bool_t chr_punctuation (uchr_t c)***
- *Determine whether a specified character is a punctuation character. **bool_t chr_blank (uchr_t c)***
- *Determine whether a specified character is a blank character (space or tab). **bool_t chr_whitespace (uchr_t c)***
- *Determine whether a specified character is a whitespace character (blank or*

- *) . **bool_t chr_is_class (uchr_t c, uchr_t *chrs, size_t chrlen)***

Determine whether a character belongs to a custom set of values.

Detailed Description

Functions used to classify ASCII characters.

Definition in file **ascii.c**.

Function Documentation

bool_t chr_alphanumeric (uchr_t c)

Determine whether a specified character is alpha-numeric.

ascii.c

Parameters:

c the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 49 of file **ascii.c**.

bool_t chr_ascii (uchr_t c)

Determine whether a specified character is an ASCII character.

Parameters:

c the character to be verified.

Returns:

true if the character passed the test, or false otherwise.
Definition at line 20 of file `ascii.c`.

bool_t chr_blank (uchar_t c)

Determine whether a specified character is a blank character (space or tab).

Parameters:

c the character to be verified.

Returns:

true if the character passed the test, or false otherwise.
Definition at line 118 of file `ascii.c`.

bool_t chr_is_class (uchar_t c, uchar_t * chrs, size_t chrlen)

Determine whether a character belongs to a custom set of values.

Parameters:

c the character to be verified.

chrs a pointer to a buffer containing the family of valid character values.

chrlen the number of characters in the testing set.

Returns:

true if the character passed the test, or false otherwise.
Definition at line 149 of file `ascii.c`.

bool_t chr_lower (uchar_t c)

Determine whether a specified character is a lowercase character.

Parameters:

c the character to be verified.

Returns:

true if the character passed the test, or false otherwise.
Definition at line 62 of file `ascii.c`.

bool_t chr_numeric (uchar_t c)

Determine whether a specified character is a numeric character.

Parameters:

c the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 90 of file `ascii.c`.

Referenced by `process_kill()`.

`bool_t chr_printable (uchar_t c)`

Determine whether a specified character is a printable character.

Parameters:

`c` the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 34 of file `ascii.c`.

Referenced by `contact_validate_detail()`, `contact_validate_name()`, and `hex_encode_st_debug()`.

`bool_t chr_punctuation (uchar_t c)`

Determine whether a specified character is a punctuation character.

Parameters:

`c` the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 104 of file `ascii.c`.

`bool_t chr_upper (uchar_t c)`

Determine whether a specified character is an uppercase character.

Parameters:

`c` the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 76 of file `ascii.c`.

`bool_t chr_whitespace (uchar_t c)`

Determine whether a specified character is a whitespace character (blank or).

Parameters:

`c` the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 132 of file `ascii.c`.

Referenced by `credential_address()`, and `imap_command_log_safe()`.

magma/core/classify/classify.h File Reference

Functions used to classify characters into specific classes.

Functions

- **bool_t chr_alphanumeric (uchr_t c)**
 - *ascii.c* **bool_t chr_ascii (uchr_t c)**
 - *Determine whether a specified character is an ASCII character.* **bool_t chr_blank (uchr_t c)**
 - *Determine whether a specified character is a blank character (space or tab).* **bool_t chr_lower (uchr_t c)**
 - *Determine whether a specified character is a lowercase character.* **bool_t chr_numeric (uchr_t c)**
 - *Determine whether a specified character is a numeric character.* **bool_t chr_printable (uchr_t c)**
 - *Determine whether a specified character is a printable character.* **bool_t chr_punctuation (uchr_t c)**
 - *Determine whether a specified character is a punctuation character.* **bool_t chr_upper (uchr_t c)**
 - *Determine whether a specified character is an uppercase character.* **bool_t chr_whitespace (uchr_t c)**
 - *Determine whether a specified character is a whitespace character (blank or*
 - *).* **bool_t chr_is_class (uchr_t c, uchr_t *chrs, size_t chrln)**
- Determine whether a character belongs to a custom set of values.*
-

Detailed Description

Functions used to classify characters into specific classes.

Definition in file **classify.h**.

Function Documentation

bool_t chr_alphanumeric (uchr_t c)

ascii.c

ascii.c

Parameters:

c the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 49 of file *ascii.c*.

bool_t chr_ascii (uchr_t c)

Determine whether a specified character is an ASCII character.

Parameters:

c the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 20 of file `ascii.c`.

`bool_t chr_blank (uchar_t c)`

Determine whether a specified character is a blank character (space or tab).

Parameters:

c the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 118 of file `ascii.c`.

`bool_t chr_is_class (uchar_t c, uchar_t * chrs, size_t chrlen)`

Determine whether a character belongs to a custom set of values.

Parameters:

c the character to be verified.

chrs a pointer to a buffer containing the family of valid character values.

chrlen the number of characters in the testing set.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 149 of file `ascii.c`.

`bool_t chr_lower (uchar_t c)`

Determine whether a specified character is a lowercase character.

Parameters:

c the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 62 of file `ascii.c`.

`bool_t chr_numeric (uchar_t c)`

Determine whether a specified character is a numeric character.

Parameters:

c the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 90 of file `ascii.c`.

Referenced by `process_kill()`.

`bool_t chr_printable (uchar_t c)`

Determine whether a specified character is a printable character.

Parameters:

`c` the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 34 of file `ascii.c`.

Referenced by `contact_validate_detail()`, `contact_validate_name()`, and `hex_encode_st_debug()`.

`bool_t chr_punctuation (uchar_t c)`

Determine whether a specified character is a punctuation character.

Parameters:

`c` the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 104 of file `ascii.c`.

`bool_t chr_upper (uchar_t c)`

Determine whether a specified character is an uppercase character.

Parameters:

`c` the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 76 of file `ascii.c`.

`bool_t chr_whitespace (uchar_t c)`

Determine whether a specified character is a whitespace character (blank or).

Parameters:

`c` the character to be verified.

Returns:

true if the character passed the test, or false otherwise.

Definition at line 132 of file `ascii.c`.

Referenced by `credential_address()`, and `imap_command_log_safe()`.

magma/core/compare/compare.h File Reference

Function declarations for various data and string comparison functions.

Functions

- **int_t st_cmp_ci_ends** (**stringer_t** *s, **stringer_t** *ends)
 - *ends.c* **int_t st_cmp_cs_ends** (**stringer_t** *s, **stringer_t** *ends)
 - *Perform a case-sensitive check to see if one string ends in another, comparing backwards.* **int_t mm_cmp_ci_eq** (**void** *a, **void** *b, **size_t** len)
 - *equal.c* **int_t mm_cmp_cs_eq** (**void** *a, **void** *b, **size_t** len)
 - *Perform a case-sensitive comparison of two memory blocks.* **int_t st_cmp_ci_eq** (**stringer_t** *a, **stringer_t** *b)
 - *Perform a case-insensitive comparison of two managed strings.* **int_t st_cmp_cs_eq** (**stringer_t** *a, **stringer_t** *b)
 - *Perform a case-sensitive comparison of two managed strings.* **bool_t st_search_ci** (**stringer_t** *haystack, **stringer_t** *needle, **size_t** *location)
 - *search.c* **bool_t st_search_cs** (**stringer_t** *haystack, **stringer_t** *needle, **size_t** *location)
 - *Search one managed string for an occurrence of another in a case-sensitive manner, and save its location.* **bool_t st_search_chr** (**stringer_t** *haystack, **chr_t** needle, **size_t** *location)
 - *Search for a character inside of a managed string, and save its location.* **int_t st_cmp_ci_starts** (**stringer_t** *s, **stringer_t** *starts)
 - *starts.c* **int_t st_cmp_cs_starts** (**stringer_t** *s, **stringer_t** *starts)
- Perform a case-sensitive check to see if one string starts with another.*
-

Detailed Description

Function declarations for various data and string comparison functions.

Definition in file **compare.h**.

Function Documentation

int_t mm_cmp_ci_eq (**void** * a, **void** * b, **size_t** len)

equal.c

equal.c

Parameters:

- a* a pointer to the first buffer in memory to be compared.
- b* a pointer to the second buffer in memory to be compared.
- len* the length, in bytes, of the comparison that is to take place.

Returns:

- 1 if *a* < *b*, 1 if *b* < *a*, or 0 if the two memory blocks are equal.

Definition at line 58 of file **equal.c**.

References **lower_chr()**, and **mm_empty()**.

Referenced by **imap_fetch_body_header()**, **mail_add_forward_headers()**, **mail_add_outbound_headers()**, **mail_discover_encoding()**, **mail_discover_type()**, **mail_header_fetch_all()**, **mail_headers()**, **mail_message()**, **mail_message_cleanup()**, **mail_mime_encoding()**, **mail_mime_type()**, and **mail_setup_basic()**.

int_t mm_cmp_cs_eq (void * a, void * b, size_t len)

Perform a case-sensitive comparison of two memory blocks.

Parameters:

a a pointer to the first buffer in memory to be compared.
b a pointer to the second buffer in memory to be compared.
len the length, in bytes, of the comparison that is to take place.

Returns:

-1 if *a* < *b*, 1 if *b* < *a*, or 0 if the two memory blocks are equal.

Definition at line 23 of file equal.c.

References mm_empty().

Referenced by args_parse(), mail_get_chunk(), mail_mime_child(), mail_mime_count(), and mail_mime_get_media_type().

int_t st_cmp_ci_ends (stringer_t * s, stringer_t * ends)

ends.c

ends.c

Note:

The value of the result depends on a backwards character comparison pattern, not a forward one.

Parameters:

s the managed string to have its ending characters examined.
ends the managed string to be compared against the end of *s*.

Returns:

-1 if *s* < *ends*, 1 if *ends* < *s* or 0 if *s* ends with the content of *ends*.

Definition at line 65 of file ends.c.

References lower_chr(), and st_empty_out().

Referenced by http_content_load_fonts(), http_load_file(), and register_captcha_random_font().

int_t st_cmp_ci_eq (stringer_t * a, stringer_t * b)

Perform a case-insensitive comparison of two managed strings.

Parameters:

a the first managed string to be compared.
b the second managed string to be compared.

Returns:

-1 if *a* < *b*, 1 if *b* < *a*, or 0 if the two memory blocks are equal.

Definition at line 130 of file equal.c.

References lower_chr(), and st_empty_out().

Referenced by cache_config(), cache_set_value(), config_key_lookup(), config_value_set(), contact_edit(), contact_find_detail(), contact_find_name(), http_data_get(), http_parse_header(), http_response_allow_cross(), imap_command_parser(), imap_compare(), imap_fetch_body(), imap_flag_action(), imap_folder_compare(),

imap_folder_create(), imap_folder_remove(), imap_folder_rename(), imap_get_flag(), imap_list(), imap_lsub(),
imap_parse_dataitems(), imap_search_messages_date_compare(), imap_search_messages_inner(),
imap_status(), lib_load(), magma_folder_find_full_name(), magma_folder_find_name(),
mail_add_inbound_headers(), mail_discover_insertion_point(), meta_folders_by_name(), molten_compare(),
pop_compare(), portal_endpoint_compare(), portal_endpoint_contacts_add(), portal_endpoint_folders_list(),
portal_endpoint_messages_flag(), portal_endpoint_messages_tag(), portal_parse_action(),
portal_parse_context(), portal_parse_flags(), portal_parse_sections(), portal_upload(),
register_business_step1(), relay_config(), relay_set_value(), servers_config(), servers_set_value(),
servers_validate(), smtp_accept_message(), smtp_auth_login(), smtp_auth_plain(), smtp_check_filters(),
smtp_compare(), smtp_data_outbound(), and smtp_rcpt_to().

int_t st_cmp_ci_starts (stringer_t * s, stringer_t * starts)

starts.c

starts.c

Parameters:

s the managed string to have its beginning characters examined.
starts the managed string to be compared against the beginning of *s*.

Returns:

-1 if *s* < *starts*, 1 if *starts* < *s* or 0 if *s* begins with *starts*.

Definition at line 59 of file starts.c.

References lower_chr(), and st_empty_out().

Referenced by cache_config(), config_load_cmdline_settings(), config_load_database_settings(),
config_load_file_settings(), http_parse_context(), http_parse_method(), http_parse_origin(), http_response(),
imap_compare(), mail_add_forward_headers(), mail_count_received(), mail_header_fetch_cleaned(),
mail_mod_subject(), molten_compare(), pop_compare(), portal_endpoint_cookies(), process_kill(),
register_abuse_check_blocklist(), relay_config(), servers_config(), smtp_compare(),
smtp_parse_mail_from_path(), and virus_check().

int_t st_cmp_cs_ends (stringer_t * s, stringer_t * ends)

Perform a case-sensitive check to see if one string ends in another, comparing backwards.

Note:

The value of the result depends on a backwards character comparison pattern, not a forward one.

Parameters:

s the managed string to have its ending characters examined.
ends the managed string to be compared against the end of *s*.

Returns:

-1 if *s* < *ends*, 1 if *ends* < *s* or 0 if *s* ends with the content of *ends*.

Definition at line 22 of file ends.c.

References st_empty_out().

Referenced by http_content_load_directory(), http_content_load_fonts(), http_content_start(), http_load_file(),
and stats_sum_errors().

int_t st_cmp_cs_eq (stringer_t * a, stringer_t * b)

Perform a case-sensitive comparison of two managed strings.

Parameters:

- a* the first managed string to be compared.
- b* the second managed string to be compared.

Returns:

- 1 if $a < b$, 1 if $b < a$, or 0 if the two memory blocks are equal.

Definition at line 91 of file equal.c.

References st_empty_out().

Referenced by args_parse(), cmp_mt_mt(), config_free(), config_output_value(), config_output_value_generic(), config_value_set(), contact_business(), contact_business_add_error(), contact_print_form(), contact_print_message(), contact_process(), credential_username(), get_header_opt(), http_content_load_directory(), http_load_file(), http_response(), ident_mt_mt(), imap_fetch(), imap_fetch_bodystructure(), lib_symbols(), magma_folder_find_full_name(), magma_folder_find_name(), mail_add_inbound_headers(), mail_add_outbound_headers(), meta_folders_by_name(), meta_get(), portal_endpoint_contacts_add(), register_business_step2(), sanity_check(), servers_output_settings(), sess_get(), smtp_accept_message(), smtp_bounce(), smtp_get_action(), smtp_parse_mail_from_path(), smtp_update_receive_stats(), spool_check_file(), stats_get_name_pos(), stats_sum_errors(), teacher_process(), and user_config_edit().

int_t st_cmp_cs_starts (stringer_t * s, stringer_t * starts)

Perform a case-sensitive check to see if one string starts with another.

Parameters:

- s* the managed string to have its beginning characters examined.
- ends* the managed string to be compared against the beginning of *s*.

Returns:

- 1 if $s < starts$, 1 if $starts < s$ or 0 if *s* begins with *starts*.

Definition at line 21 of file starts.c.

References st_empty_out().

Referenced by http_response(), imap_parse_dataitems(), smtp_auth_login(), smtp_auth_plain(), st_replace(), stats_sum_errors(), and teacher_process().

bool_t st_search_chr (stringer_t * haystack, chr_t needle, size_t * location)

Search for a character inside of a managed string, and save its location.

Parameters:

- haystack* the managed string to be searched.
- needle* the character to be found in the string.
- location* if not NULL, a pointer to store the index of needle if found, or 0 on no match.

Returns:

- true if the specified character was found or false otherwise.

Definition at line 122 of file search.c.

References log_pedantic, and st_empty_out().

bool_t st_search_ci (stringer_t * *haystack*, stringer_t * *needle*, size_t * *location*)

search.c

search.c

Parameters:

haystack the managed string to be searched.

needle the managed string to be found.

location if not NULL, a pointer to store the index of needle if found, or 0 on no match.

Returns:

true if the string is found or false otherwise.

Definition at line 73 of file search.c.

References log_pedantic, lower_chr(), and st_empty_out().

Referenced by imap_command_log_safe(), imap_search_messages_body(), imap_search_messages_header(),
imap_search_messages_text(), mail_get_boundary(), mail_mime_boundary(),
mail_mime_generate_boundary(), and pattern_check().

bool_t st_search_cs (stringer_t * *haystack*, stringer_t * *needle*, size_t * *location*)

Search one managed string for an occurrence of another in a case-sensitive manner, and save its location.

Parameters:

haystack the managed string to be searched.

needle the managed string to be found.

location if not NULL, a pointer to store the index of needle if found, or 0 on no match.

Returns:

true if the string is found or false otherwise.

Definition at line 22 of file search.c.

References log_pedantic, and st_empty_out().

Referenced by credential_alloc_auth(), credential_alloc_mail(), credential_username(), http_parse_header(),
http_parse_pairs(), mail_get_chunk(), str_tok_get_bl(), and str_tok_get_count_bl().

magma/core/compare/ends.c File Reference

Functions used to compare the ends of strings with other strings.

```
#include "magma.h"
```

Functions

- **int_t st_cmp_cs_ends** (stringer_t *s, stringer_t *ends)
- *Perform a case-sensitive check to see if one string ends in another, comparing backwards.* **int_t st_cmp_ci_ends** (stringer_t *s, stringer_t *ends)

Perform a case-insensitive check to see if one string ends in another, comparing backwards.

Detailed Description

Functions used to compare the ends of strings with other strings.

Definition in file **ends.c**.

Function Documentation

int_t st_cmp_ci_ends (stringer_t * s, stringer_t * ends)

Perform a case-insensitive check to see if one string ends in another, comparing backwards.

ends.c

Note:

The value of the result depends on a backwards character comparison pattern, not a forward one.

Parameters:

s the managed string to have its ending characters examined.

ends the managed string to be compared against the end of *s*.

Returns:

-1 if *s* < *ends*, 1 if *ends* < *s* or 0 if *s* ends with the content of *ends*.

Definition at line 65 of file **ends.c**.

References **lower_chr()**, and **st_empty_out()**.

Referenced by **http_content_load_fonts()**, **http_load_file()**, and **register_captcha_random_font()**.

int_t st_cmp_cs_ends (stringer_t * s, stringer_t * ends)

Perform a case-sensitive check to see if one string ends in another, comparing backwards.

Note:

The value of the result depends on a backwards character comparison pattern, not a forward one.

Parameters:

s the managed string to have its ending characters examined.

ends the managed string to be compared against the end of *s*.

Returns:

-1 if $s < \text{ends}$, 1 if $\text{ends} < s$ or 0 if s ends with the content of ends .

Definition at line 22 of file `ends.c`.

References `st_empty_out()`.

Referenced by `http_content_load_directory()`, `http_content_load_fonts()`, `http_content_start()`, `http_load_file()`, and `stats_sum_errors()`.

magma/core/compare/equal.c File Reference

Functions to check for string equality.

```
#include "magma.h"
```

Functions

- **int_t mm_cmp_cs_eq** (void *a, void *b, size_t len)
- *Perform a case-sensitive comparison of two memory blocks.* **int_t mm_cmp_ci_eq** (void *a, void *b, size_t len)
- *Perform a case-insensitive comparison of two memory blocks.* **int_t st_cmp_cs_eq** (stringer_t *a, stringer_t *b)
- *Perform a case-sensitive comparison of two managed strings.* **int_t st_cmp_ci_eq** (stringer_t *a, stringer_t *b)

Perform a case-insensitive comparison of two managed strings.

Detailed Description

Functions to check for string equality.

Definition in file **equal.c**.

Function Documentation

int_t mm_cmp_ci_eq (void * a, void * b, size_t len)

Perform a case-insensitive comparison of two memory blocks.

equal.c

Parameters:

- a* a pointer to the first buffer in memory to be compared.
- b* a pointer to the second buffer in memory to be compared.
- len* the length, in bytes, of the comparison that is to take place.

Returns:

- 1 if *a* < *b*, 1 if *b* < *a*, or 0 if the two memory blocks are equal.

Definition at line 58 of file **equal.c**.

References **lower_chr()**, and **mm_empty()**.

Referenced by **imap_fetch_body_header()**, **mail_add_forward_headers()**, **mail_add_outbound_headers()**, **mail_discover_encoding()**, **mail_discover_type()**, **mail_header_fetch_all()**, **mail_headers()**, **mail_message()**, **mail_message_cleanup()**, **mail_mime_encoding()**, **mail_mime_type()**, and **mail_setup_basic()**.

int_t mm_cmp_cs_eq (void * a, void * b, size_t len)

Perform a case-sensitive comparison of two memory blocks.

Parameters:

a a pointer to the first buffer in memory to be compared.
b a pointer to the second buffer in memory to be compared.
len the length, in bytes, of the comparison that is to take place.

Returns:

-1 if $a < b$, 1 if $b < a$, or 0 if the two memory blocks are equal.

Definition at line 23 of file equal.c.

References mm_empty().

Referenced by args_parse(), mail_get_chunk(), mail_mime_child(), mail_mime_count(), and mail_mime_get_media_type().

int_t st_cmp_ci_eq (stringer_t * a, stringer_t * b)

Perform a case-insensitive comparison of two managed strings.

Parameters:

a the first managed string to be compared.
b the second managed string to be compared.

Returns:

-1 if $a < b$, 1 if $b < a$, or 0 if the two memory blocks are equal.

Definition at line 130 of file equal.c.

References lower_chr(), and st_empty_out().

Referenced by cache_config(), cache_set_value(), config_key_lookup(), config_value_set(), contact_edit(), contact_find_detail(), contact_find_name(), http_data_get(), http_parse_header(), http_response_allow_cross(), imap_command_parser(), imap_compare(), imap_fetch_body(), imap_flag_action(), imap_folder_compare(), imap_folder_create(), imap_folder_remove(), imap_folder_rename(), imap_get_flag(), imap_list(), imap_lsub(), imap_parse_dataitems(), imap_search_messages_date_compare(), imap_search_messages_inner(), imap_status(), lib_load(), magma_folder_find_full_name(), magma_folder_find_name(), mail_add_inbound_headers(), mail_discover_insertion_point(), meta_folders_by_name(), molten_compare(), pop_compare(), portal_endpoint_compare(), portal_endpoint_contacts_add(), portal_endpoint_folders_list(), portal_endpoint_messages_flag(), portal_endpoint_messages_tag(), portal_parse_action(), portal_parse_context(), portal_parse_flags(), portal_parse_sections(), portal_upload(), register_business_step1(), relay_config(), relay_set_value(), servers_config(), servers_set_value(), servers_validate(), smtp_accept_message(), smtp_auth_login(), smtp_auth_plain(), smtp_check_filters(), smtp_compare(), smtp_data_outbound(), and smtp_rcpt_to().

int_t st_cmp_cs_eq (stringer_t * a, stringer_t * b)

Perform a case-sensitive comparison of two managed strings.

Parameters:

a the first managed string to be compared.
b the second managed string to be compared.

Returns:

-1 if $a < b$, 1 if $b < a$, or 0 if the two memory blocks are equal.

Definition at line 91 of file equal.c.

References st_empty_out().

Referenced by args_parse(), cmp_mt_mt(), config_free(), config_output_value(), config_output_value_generic(), config_value_set(), contact_business(), contact_business_add_error(), contact_print_form(), contact_print_message(), contact_process(), credential_username(), get_header_opt(), http_content_load_directory(), http_load_file(), http_response(), ident_mt_mt(), imap_fetch(), imap_fetch_bodystructure(), lib_symbols(), magma_folder_find_full_name(), magma_folder_find_name(), mail_add_inbound_headers(), mail_add_outbound_headers(), meta_folders_by_name(), meta_get(), portal_endpoint_contacts_add(), register_business_step2(), sanity_check(), servers_output_settings(), sess_get(), smtp_accept_message(), smtp_bounce(), smtp_get_action(), smtp_parse_mail_from_path(), smtp_update_receive_stats(), spool_check_file(), stats_get_name_pos(), stats_sum_errors(), teacher_process(), and user_config_edit().

magma/core/compare/search.c File Reference

String search functions.

```
#include "magma.h"
```

Functions

- **bool_t st_search_cs** (**stringer_t** *haystack, **stringer_t** *needle, **size_t** *location)
- *Search one managed string for an occurrence of another in a case-sensitive manner, and save its location.*
bool_t st_search_ci (**stringer_t** *haystack, **stringer_t** *needle, **size_t** *location)
- *Search one managed string for an occurrence of another in a case-insensitive manner, and save its location.*
bool_t st_search_chr (**stringer_t** *haystack, **chr_t** needle, **size_t** *location)

Search for a character inside of a managed string, and save its location.

Detailed Description

String search functions.

Definition in file **search.c**.

Function Documentation

bool_t st_search_chr (**stringer_t** * *haystack*, **chr_t** *needle*, **size_t** * *location*)

Search for a character inside of a managed string, and save its location.

Parameters:

haystack the managed string to be searched.

needle the character to be found in the string.

location if not NULL, a pointer to store the index of needle if found, or 0 on no match.

Returns:

true if the specified character was found or false otherwise.

Definition at line 122 of file search.c.

References log_pedantic, and st_empty_out().

bool_t st_search_ci (**stringer_t** * *haystack*, **stringer_t** * *needle*, **size_t** * *location*)

Search one managed string for an occurrence of another in a case-insensitive manner, and save its location.

search.c

Parameters:

haystack the managed string to be searched.

needle the managed string to be found.

location if not NULL, a pointer to store the index of needle if found, or 0 on no match.

Returns:

true if the string is found or false otherwise.

Definition at line 73 of file search.c.

References log_pedantic, lower_chr(), and st_empty_out().

Referenced by imap_command_log_safe(), imap_search_messages_body(), imap_search_messages_header(),
imap_search_messages_text(), mail_get_boundary(), mail_mime_boundary(),
mail_mime_generate_boundary(), and pattern_check().

bool_t st_search_cs (stringer_t * *haystack*, stringer_t * *needle*, size_t * *location*)

Search one managed string for an occurrence of another in a case-sensitive manner, and save its location.

Parameters:

haystack the managed string to be searched.

needle the managed string to be found.

location if not NULL, a pointer to store the index of needle if found, or 0 on no match.

Returns:

true if the string is found or false otherwise.

Definition at line 22 of file search.c.

References log_pedantic, and st_empty_out().

Referenced by credential_alloc_auth(), credential_alloc_mail(), credential_username(), http_parse_header(),
http_parse_pairs(), mail_get_chunk(), str_tok_get_bl(), and str_tok_get_count_bl().

magma/servers/imap/search.c File Reference

Functions used to handle IMAP commands/actions.

```
#include "magma.h"
```

Functions

- **int_t imap_search_messages_date_compare** (**stringer_t** *one, **stringer_t** *two)
- **int_t imap_search_messages_date** (**meta_user_t** *user, **mail_message_t** **data, **stringer_t** **header, **meta_message_t** *active, **stringer_t** *date, **int_t** internal, **int_t** expected)
- **int_t imap_search_messages_header** (**meta_user_t** *user, **mail_message_t** **data, **stringer_t** **header, **meta_message_t** *active, **stringer_t** *field, **stringer_t** *value)
- **int_t imap_search_messages_body** (**meta_user_t** *user, **mail_message_t** **data, **meta_message_t** *active, **stringer_t** *value)
- **int_t imap_search_messages_text** (**meta_user_t** *user, **mail_message_t** **data, **meta_message_t** *active, **stringer_t** *value)
- **int_t imap_search_messages_size** (**meta_message_t** *active, **stringer_t** *value, **int_t** expected)
- **int_t imap_search_flag** (**uint32_t** status, **uint32_t** flag, **int_t** has)
- *search.c* **int_t imap_search_messages_range** (**meta_message_t** *active, **stringer_t** *range, **int_t** uid)
- **int_t imap_search_messages_inner** (**meta_user_t** *user, **mail_message_t** **message, **stringer_t** **header, **meta_message_t** *current, **imap_arguments_t** *array, unsigned recursion)
- **inx_t * imap_search_messages** (**connection_t** *con)

Variables

- **chr_t * MONTH_LOOKUP** [] = { "JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC" }

Detailed Description

Functions used to handle IMAP commands/actions.

Definition in file **search.c**.

Function Documentation

int_t imap_search_flag (**uint32_t** status, **uint32_t** flag, **int_t** has)

search.c

Definition at line 271 of file search.c.

Referenced by `imap_search_messages_inner()`.

inx_t* imap_search_messages (**connection_t** * con)

LOW: Is a read lock necessary now that were using index reference counters and thread safe iteration cursors?

Definition at line 569 of file search.c.

References `count`, `meta_message_t::foldernum`, `connection_t::imap`, `imap_search_messages_inner()`, `inx_alloc()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_key_active()`, `inx_cursor_value_next()`, `inx_find()`, `inx_insert()`, `M_INX_LINKED`, `M_TYPE_UINT64`, `mail_destroy()`, `mail_destroy_header()`, `meta_message_t::messagenum`, `meta_message_dup()`, `meta_message_free()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_message_t::sequencenum`, `status`, `multi_t::u64`, and `multi_t::val`.

Referenced by `imap_search()`.

```
int_t imap_search_messages_body (meta_user_t * user,    mail_message_t ** data,
meta_message_t * active,    stringer_t * value)
```

Definition at line 202 of file search.c.

References `mail_load_message()`, `mail_mime_update()`, `st_free()`, and `st_search_ci()`.

Referenced by `imap_search_messages_inner()`.

```
int_t imap_search_messages_date (meta_user_t * user,    mail_message_t ** data,    stringer_t **
header,    meta_message_t * active,    stringer_t * date,    int_t internal,    int_t expected)
```

Definition at line 87 of file search.c.

References `meta_message_t::created`, `imap_search_messages_date_compare()`, `mail_header_fetch_cleaned()`, `mail_load_header()`, `ns_length_get()`, `pl_empty()`, `pl_init()`, `pl_null()`, `PLACER`, `placer_t`, `st_char_get()`, `st_free()`, `st_import()`, `st_length_get()`, `st_merge`, `tok_get_count_st()`, and `tok_get_st()`.

Referenced by `imap_search_messages_inner()`.

```
int_t imap_search_messages_date_compare (stringer_t * one,    stringer_t * two)
```

Definition at line 17 of file search.c.

References `MONTH_LOOKUP`, `PLACER`, `placer_t`, `st_cmp_ci_eq()`, `tok_get_count_st()`, `tok_get_st()`, and `uint32_conv_st()`.

Referenced by `imap_search_messages_date()`.

```
int_t imap_search_messages_header (meta_user_t * user,    mail_message_t ** data,    stringer_t
** header,    meta_message_t * active,    stringer_t * field,    stringer_t * value)
```

Definition at line 166 of file search.c.

References `mail_header_fetch_all()`, `mail_load_header()`, `pl_empty()`, `pl_init()`, `pl_null()`, `placer_t`, `st_char_get()`, `st_free()`, `st_length_get()`, and `st_search_ci()`.

Referenced by `imap_search_messages_inner()`.

```
int_t imap_search_messages_inner (meta_user_t * user,    mail_message_t ** message,
stringer_t ** header,    meta_message_t * current,    imap_arguments_t * array,    unsigned
recursion)
```

Definition at line 353 of file search.c.

References `ar_length_get()`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_get_ar_ar()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `imap_search_flag()`, `imap_search_messages_body()`, `imap_search_messages_date()`,

imap_search_messages_header(), imap_search_messages_inner(), imap_search_messages_range(),
imap_search_messages_size(), imap_search_messages_text(), IMAP_SEARCH_RECURSION_LIMIT,
imap_valid_sequence(), log_pedantic, MAIL_STATUS_ANSWERED, MAIL_STATUS_DELETED,
MAIL_STATUS_DRAFT, MAIL_STATUS_FLAGGED, MAIL_STATUS_RECENT,
MAIL_STATUS_SEEN, number, PLACER, st_cmp_ci_eq(), and meta_message_t::status.

Referenced by imap_search_messages(), and imap_search_messages_inner().

int_t imap_search_messages_range (meta_message_t * active, stringer_t * range, int_t uid)

Definition at line 281 of file search.c.

References meta_message_t::messagenum, number, pl_char_get(), pl_empty(), pl_null(), placer_t,
meta_message_t::sequencenum, tok_get_count_st(), tok_get_st(), and uint64_conv_st().

Referenced by imap_search_messages_inner().

int_t imap_search_messages_size (meta_message_t * active, stringer_t * value, int_t expected)

Definition at line 250 of file search.c.

References meta_message_t::size, and uint64_conv_st().

Referenced by imap_search_messages_inner().

int_t imap_search_messages_text (meta_user_t * user, mail_message_t ** data, meta_message_t * active, stringer_t * value)

Definition at line 226 of file search.c.

References mail_load_message(), mail_mime_update(), st_free(), and st_search_ci().

Referenced by imap_search_messages_inner().

Variable Documentation

chr_t* MONTH_LOOKUP[] = { "JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC" }

Definition at line 15 of file search.c.

Referenced by imap_search_messages_date_compare().

magma/core/compare/starts.c File Reference

Functions used to compare the starts of strings with other strings.

```
#include "magma.h"
```

Functions

- **int_t st_cmp_cs_starts** (**stringer_t** *s, **stringer_t** *starts)
- *Perform a case-sensitive check to see if one string starts with another.* **int_t st_cmp_ci_starts** (**stringer_t** *s, **stringer_t** *starts)

Perform a case-insensitive check to see if one string starts with another.

Detailed Description

Functions used to compare the starts of strings with other strings.

Definition in file **starts.c**.

Function Documentation

int_t st_cmp_ci_starts (**stringer_t** * s, **stringer_t** * starts)

Perform a case-insensitive check to see if one string starts with another.

starts.c

Parameters:

s the managed string to have its beginning characters examined.
starts the managed string to be compared against the beginning of s.

Returns:

-1 if s < starts, 1 if starts < s or 0 if s begins with starts.

Definition at line 59 of file starts.c.

References `lower_chr()`, and `st_empty_out()`.

Referenced by `cache_config()`, `config_load_cmdline_settings()`, `config_load_database_settings()`, `config_load_file_settings()`, `http_parse_context()`, `http_parse_method()`, `http_parse_origin()`, `http_response()`, `imap_compare()`, `mail_add_forward_headers()`, `mail_count_received()`, `mail_header_fetch_cleaned()`, `mail_mod_subject()`, `molten_compare()`, `pop_compare()`, `portal_endpoint_cookies()`, `process_kill()`, `register_abuse_check_blocklist()`, `relay_config()`, `servers_config()`, `smtp_compare()`, `smtp_parse_mail_from_path()`, and `virus_check()`.

int_t st_cmp_cs_starts (**stringer_t** * s, **stringer_t** * starts)

Perform a case-sensitive check to see if one string starts with another.

Parameters:

s the managed string to have its beginning characters examined.
ends the managed string to be compared against the beginning of s.

Returns:

-1 if $s < \text{starts}$, 1 if $\text{starts} < s$ or 0 if s begins with starts .

Definition at line 21 of file `starts.c`.

References `st_empty_out()`.

Referenced by `http_response()`, `imap_parse_dataitems()`, `smtp_auth_login()`, `smtp_auth_plain()`, `st_replace()`, `stats_sum_errors()`, and `teacher_process()`.

magma/core/core.h File Reference

A collection of types, declarations and includes needed when accessing the core module and the type definitions needed to parse the header files that follow.

```
#include "memory/memory.h"
#include "strings/strings.h"
#include "classify/classify.h"
#include "encodings/encodings.h"
#include "log/log.h"
#include "indexes/indexes.h"
#include "compare/compare.h"
#include "thread/thread.h"
#include "buckets/buckets.h"
#include "parsers/parsers.h"
#include "hash/hash.h"
#include "host/host.h"
```

Typedefs

- typedef char **chr_t**
- typedef bool **bool_t**
- typedef int32_t **int_t**
- typedef uint32_t **uint_t**
- typedef unsigned char **uchr_t**
- typedef unsigned char **byte_t**

Enumerations

- enum **M_TYPE** { **M_TYPE_MULTI** = 1, **M_TYPE_ENUM**, **M_TYPE_BOOLEAN**, **M_TYPE_BLOCK**, **M_TYPE_NULLER**, **M_TYPE_PLACER**, **M_TYPE_STRINGER**, **M_TYPE_INT8**, **M_TYPE_INT16**, **M_TYPE_INT32**, **M_TYPE_INT64**, **M_TYPE_UINT8**, **M_TYPE_UINT16**, **M_TYPE_UINT32**, **M_TYPE_UINT64**, **M_TYPE_FLOAT**, **M_TYPE_DOUBLE** }
- enum { **EMPTY** = 0 }

Functions

- char * **type** (**M_TYPE** type)

Detailed Description

A collection of types, declarations and includes needed when accessing the core module and the type definitions needed to parse the header files that follow.

Definition in file **core.h**.

Typedef Documentation

typedef bool bool_t

Definition at line 21 of file core.h.

typedef unsigned char byte_t

Definition at line 25 of file core.h.

typedef char chr_t

The type definitions used by Magma that are not defined by the system headers. The bool type requires the inclusion of stdbool.h and the use of the C99.

Definition at line 20 of file core.h.

typedef int32_t int_t

Definition at line 22 of file core.h.

typedef unsigned char uchr_t

Definition at line 24 of file core.h.

typedef uint32_t uint_t

Definition at line 23 of file core.h.

Enumeration Type Documentation

anonymous enum

Enumerator:

EMPTY

Definition at line 50 of file core.h.

enum M_TYPE

Different types used throughout.

Enumerator:

M_TYPE_MULTI M_TYPE_MULTI is **multi_t**.

M_TYPE_ENUM M_TYPE_ENUM is enum.

M_TYPE_BOOLEAN M_TYPE_BOOLEAN is bool_t.

M_TYPE_BLOCK M_TYPE_BLOCK is void pointer.

M_TYPE_NULLER M_TYPE_NULLER is char pointer.

M_TYPE_PLACER M_TYPE_PLACER is placer_t struct.

M_TYPE_STRINGER M_TYPE_STRINGER is stringer_t pointer.

M_TYPE_INT8 M_TYPE_INT8 is int8_t.
M_TYPE_INT16 M_TYPE_INT16 is int16_t.
M_TYPE_INT32 M_TYPE_INT32 is int32_t.
M_TYPE_INT64 M_TYPE_INT64 is int64_t.
M_TYPE_UINT8 M_TYPE_UINT8 is uint8_t.
M_TYPE_UINT16 M_TYPE_UINT16 is uint16_t.
M_TYPE_UINT32 M_TYPE_UINT32 is uint32_t.
M_TYPE_UINT64 M_TYPE_UINT64 is uint64_t.
M_TYPE_FLOAT M_TYPE_FLOAT is float.
M_TYPE_DOUBLE M_TYPE_DOUBLE is double.

Definition at line 30 of file core.h.

Function Documentation

char* type (M_TYPE type)

Takes a type code and returns the fully enumerated string associated with that type.

Parameters:

type The type code to evaluate.

Returns:

Null terminated string containing type name. String is stored in static buffer and returned as a pointer.

Definition at line 22 of file type.c.

References M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_ENUM, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_MULTI, M_TYPE_NULLER, M_TYPE_PLACER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, and M_TYPE_UINT8.

Referenced by ar_dupe(), ar_free(), cache_free(), cache_output_settings(), cache_set_value(), cache_validate(), config_free(), imap_parse_array(), imap_parse_dataitems(), mail_modify_part(), meta_data_delete_folder(), meta_data_fetch_folders(), meta_data_insert_folder(), meta_data_update_folder_name(), mt_dupe(), mt_get_char(), mt_get_length(), mt_get_number(), mt_get_type(), mt_is_empty(), mt_is_number(), mt_set_type(), portal_message_attachments(), portal_message_body(), relay_free(), relay_output_settings(), relay_set_value(), relay_validate(), servers_free(), servers_output_settings(), servers_set_value(), and servers_validate().

magma/core/encodings/base64.c File Reference

Functions for base64 encoding/decoding of data.

```
#include "magma.h"
```

Functions

- **stringer_t * base64_encode_opts** (**stringer_t** *s, uint32_t opts, **bool_t** modified)
- *Perform base64 encoding on a managed string with padding and line splitting at BASE64_LINE_WRAP_LENGTH characters. **stringer_t * base64_encode** (**stringer_t** *s, **stringer_t** *output)*
- *Perform base64 encoding on a managed string with padding and line splitting at BASE64_LINE_WRAP_LENGTH characters. **stringer_t * base64_encode_mod** (**stringer_t** *s, **stringer_t** *output)*
- *Perform modified base64 encoding on a managed string without padding or line splitting. **stringer_t * base64_decode** (**stringer_t** *s, **stringer_t** *output)*
- *Perform base64 decoding on a managed string. **stringer_t * base64_decode_opts** (**stringer_t** *s, uint32_t opts, **bool_t** modified)*
- *Perform base64 decoding on a managed string. **stringer_t * base64_decode_mod** (**stringer_t** *s, **stringer_t** *output)*

Perform modified base64 decoding on a managed string. without padding or line splitting.

Detailed Description

Functions for base64 encoding/decoding of data.

Definition in file **base64.c**.

Function Documentation

stringer_t* base64_decode (**stringer_t** * s, **stringer_t** * *output*)

Perform base64 decoding on a managed string.

base64.c

Parameters:

s the managed string to be base64 decoded.

output a managed string to receive the encoded output; if passed as NULL, one will be allocated to the caller.

Returns:

NULL on failure, or a newly allocated managed string containing the decoded result on success.

Definition at line 256 of file base64.c.

References mappings_t::base64, BASE64_DECODED_LEN, log_pedantic, mappings, st_alloc, st_avail_get(), st_data_get(), st_empty_out(), st_length_get(), st_length_set(), st_valid_avail(), st_valid_destination(), st_valid_tracked(), and mappings_t::values.

Referenced by base64_decode_opts(), mail_insert_chunk_base64(), smtp_auth_login(), and smtp_auth_plain().

stringer_t* base64_decode_mod (**stringer_t** * s, **stringer_t** * *output*)

Perform modified base64 decoding on a managed string. without padding or line splitting.

Note:

In this function, the '+' and '/' characters are replaced with '-' and '_' respectively, making the output suitable for use in URL parameters without additional encoding.

Parameters:

s the managed string to be base64 decoded.

output a managed string to receive the encoded output; if passed as NULL, one will be allocated to the caller.

Returns:

NULL on failure, or a newly allocated managed string containing the modified decoded result on success.

Definition at line 396 of file base64.c.

References BASE64_DECODED_MOD_LEN, mappings_t::base64_mod, log_pedantic, mappings, st_alloc, st_avail_get(), st_data_get(), st_empty_out(), st_length_get(), st_length_set(), st_valid_avail(), st_valid_destination(), st_valid_tracked(), and mappings_t::values.

Referenced by base64_decode_opts(), and meta_data_user_build_storage_keys().

stringer_t* base64_decode_opts (stringer_t * s, uint32_t opts, bool_t *modified*)

Perform base64 decoding on a managed string.

Parameters:

s the managed string to be base64 decoded.

opts the options value of the managed string which will be allocated and receive the decoded output.

modified if true, use modified base64 encoding; if false, use normal base64 decoding.

Returns:

NULL on failure, or a pointer to a newly allocated managed string containing the decoded result on success.

Definition at line 355 of file base64.c.

References base64_decode(), base64_decode_mod(), BASE64_DECODED_LEN, BASE64_DECODED_MOD_LEN, log_pedantic, st_alloc_opts(), st_empty_out(), and st_free().

Referenced by meta_data_user_build_storage_keys().

stringer_t* base64_encode (stringer_t * s, stringer_t * *output*)

Perform base64 encoding on a managed string with padding and line splitting at BASE64_LINE_WRAP_LENGTH characters.

Parameters:

s the managed string to be base64 encoded.

output a managed string to receive the encoded output; if passed as NULL, one will be allocated to the caller.

Returns:

NULL on failure, or a pointer to the managed string containing the encoded result on success.

Definition at line 61 of file base64.c.

References mappings_t::base64, BASE64_ENCODED_LEN, BASE64_LINE_WRAP_LENGTH, mappings_t::characters, log_error, log_pedantic, mappings, st_alloc, st_avail_get(), st_data_get(), st_empty_out(), st_length_get(), st_length_set(), st_valid_avail(), st_valid_destination(), and st_valid_tracked().

Referenced by base64_encode_opts(), mail_insert_chunk_base64(), and mail_mime_encode_part().

stringer_t* base64_encode_mod (stringer_t * s, stringer_t * output)

Perform modified base64 encoding on a managed string without padding or line splitting.

Note:

In this function, the '+' and '/' characters are replaced with '-' and '_' respectively, making the output suitable for use in URL parameters without additional encoding.

Parameters:

s the managed string to be base64 modified-encoded.

output a managed string to receive the encoded output; if passed as NULL, one will be allocated to the caller.

Returns:

NULL on failure, or a newly allocated managed string containing the modified encoded result on success.

Definition at line 172 of file base64.c.

References BASE64_ENCODED_MOD_LEN, mappings_t::base64_mod, mappings_t::characters, debug_hook(), log_pedantic, mappings, st_alloc, st_avail_get(), st_data_get(), st_empty_out(), st_length_get(), st_length_set(), st_valid_avail(), st_valid_destination(), and st_valid_tracked().

Referenced by base64_encode_opts(), and meta_data_user_save_storage_keys().

stringer_t* base64_encode_opts (stringer_t * s, uint32_t opts, bool_t modified)

Perform base64 encoding on a managed string with padding and line splitting at BASE64_LINE_WRAP_LENGTH characters.

Parameters:

s the managed string to be base64 encoded.

opts the options value of the managed string which will be allocated and receive the encoded output.

modified if true, use modified base64 encoding; if false, use normal base64 encoding.

Returns:

NULL on failure, or a pointer to a newly allocated managed string containing the encoded result on success.

Definition at line 22 of file base64.c.

References base64_encode(), base64_encode_mod(), BASE64_ENCODED_LEN, BASE64_ENCODED_MOD_LEN, log_pedantic, st_alloc_opts(), st_empty_out(), and st_free().

magma/core/encodings/encodings.h File Reference

Functions used to encode and decode data in various formats.

Data Structures

- struct **mappings_t**

Defines

- #define **URL_MAX_LENGTH** 1048576
- #define **QP_LINE_WRAP_LENGTH** 76
- #define **BASE64_LINE_WRAP_LENGTH** 76
- #define **BASE64_ENCODED_LEN**(len) ((len * 4/3) + (((len * 4/3) / BASE64_LINE_WRAP_LENGTH) * 2) + 64)
- #define **BASE64_ENCODED_MOD_LEN**(len) ((len * 4 / 3) + (len * 4 / 3) + 64)
- #define **BASE64_DECODED_LEN**(len) (len * 3/4)
- #define **BASE64_DECODED_MOD_LEN** BASE64_DECODED_LEN

Functions

- **stringer_t * base64_decode** (**stringer_t** *s, **stringer_t** *output)
- *base64.c* **stringer_t * base64_decode_opts** (**stringer_t** *s, **uint32_t** opts, **bool_t** modified)
- *Perform base64 decoding on a managed string.* **stringer_t * base64_decode_mod** (**stringer_t** *s, **stringer_t** *output)
- *Perform modified base64 decoding on a managed string, without padding or line splitting.* **stringer_t * base64_encode** (**stringer_t** *s, **stringer_t** *output)
- *Perform base64 encoding on a managed string with padding and line splitting at BASE64_LINE_WRAP_LENGTH characters.* **stringer_t * base64_encode_opts** (**stringer_t** *s, **uint32_t** opts, **bool_t** modified)
- *Perform base64 encoding on a managed string with padding and line splitting at BASE64_LINE_WRAP_LENGTH characters.* **stringer_t * base64_encode_mod** (**stringer_t** *s, **stringer_t** *output)
- *Perform modified base64 encoding on a managed string without padding or line splitting.* **stringer_t * decode_base64_modified_st** (**stringer_t** *string)
- **stringer_t * encode_base64_modified_st** (**stringer_t** *string)
- **bool_t hex_valid_chr** (**uchr_t** c)
- *hex.c* **byte_t hex_decode_chr** (**uchr_t** a, **uchr_t** b)
- *Decode a hex character pair as a single byte (usually for URL decoding).* **size_t hex_count_st** (**stringer_t** *s)
- *Count the number of hex characters in a string.* **size_t hex_valid_st** (**stringer_t** *s)
- *Determine whether a managed string is a properly formatted hex string and return the number found.* **stringer_t * hex_decode_st** (**stringer_t** *h, **stringer_t** *output)
- *Convert a hex string into a binary data blob.* **stringer_t * hex_encode_st** (**stringer_t** *b, **stringer_t** *output)
- *Convert a block of binary data into a hex string.* **uchr_t * hex_encode_chr** (**byte_t** b, **uchr_t** *output)
- **stringer_t * hex_encode_st_debug** (**stringer_t** *input, **size_t** maxlen)
- *Encode data in a human readable way, for debugging purposes.* **stringer_t * qp_decode** (**stringer_t** *s)
- *qp.c* **stringer_t * qp_encode** (**stringer_t** *s)
- *Perform QP (quoted-printable) encoding of a string.* **bool_t url_valid_chr** (**uchr_t** c)
- *url.c* **size_t url_valid_st** (**stringer_t** *s)
- *Check a URL string for validity.* **stringer_t * url_decode** (**stringer_t** *s)
- *Decode a URL-encoded string into its original representation.* **stringer_t * url_encode** (**stringer_t** *s)
- *Encode a data buffer as a valid URL component.* **stringer_t * zbase32_decode** (**stringer_t** *s)
- *zbase32.c* **stringer_t * zbase32_encode** (**stringer_t** *s)

Encode data as a zbase32 string. Variables

- `mappings_t mappings`

Detailed Description

Functions used to encode and decode data in various formats.

Definition in file `encodings.h`.

Define Documentation

`#define BASE64_DECODED_LEN(len) (len * 3/4)`

Definition at line 34 of file `encodings.h`.

Referenced by `base64_decode()`, and `base64_decode_opts()`.

`#define BASE64_DECODED_MOD_LEN BASE64_DECODED_LEN`

Definition at line 35 of file `encodings.h`.

Referenced by `base64_decode_mod()`, and `base64_decode_opts()`.

`#define BASE64_ENCODED_LEN(len) (((len * 4/3) + (((len * 4/3) / BASE64_LINE_WRAP_LENGTH) * 2) + 64)`

Definition at line 32 of file `encodings.h`.

Referenced by `base64_encode()`, and `base64_encode_opts()`.

`#define BASE64_ENCODED_MOD_LEN(len) ((len * 4 / 3) + (len * 4 / 3) + 64)`

Definition at line 33 of file `encodings.h`.

Referenced by `base64_encode_mod()`, and `base64_encode_opts()`.

`#define BASE64_LINE_WRAP_LENGTH 76`

Definition at line 30 of file `encodings.h`.

Referenced by `base64_encode()`.

`#define QP_LINE_WRAP_LENGTH 76`

Definition at line 29 of file `encodings.h`.

Referenced by `qp_encode()`.

#define URL_MAX_LENGTH 1048576

Note:

When considering a change to the URL length limit, the following statistics may be useful:

Internet Explorer: 2,048 characters for the host/path and 2,083 characters overall Firefox: limited to 65,536 visible characters, but longer URL will still work Safari: at least 80,000 characters Opera: at least 190,000 characters IIS: by default the limit is 16,384 but can be increased Apache: by default 4,000 characters

Definition at line 28 of file encodings.h.

Function Documentation

stringer_t* base64_decode (stringer_t * s, stringer_t * *output*)

base64.c

base64.c

Parameters:

s the managed string to be base64 decoded.

output a managed string to receive the encoded output; if passed as NULL, one will be allocated to the caller.

Returns:

NULL on failure, or a newly allocated managed string containing the decoded result on success.

Definition at line 256 of file base64.c.

References mappings_t::base64, BASE64_DECODED_LEN, log_pedantic, mappings, st_alloc, st_avail_get(), st_data_get(), st_empty_out(), st_length_get(), st_length_set(), st_valid_avail(), st_valid_destination(), st_valid_tracked(), and mappings_t::values.

Referenced by base64_decode_opts(), mail_insert_chunk_base64(), smtp_auth_login(), and smtp_auth_plain().

stringer_t* base64_decode_mod (stringer_t * s, stringer_t * *output*)

Perform modified base64 decoding on a managed string. without padding or line splitting.

Note:

In this function, the '+' and '/' characters are replaced with '-' and '_' respectively, making the output suitable for use in URL parameters without additional encoding.

Parameters:

s the managed string to be base64 decoded.

output a managed string to receive the encoded output; if passed as NULL, one will be allocated to the caller.

Returns:

NULL on failure, or a newly allocated managed string containing the modified decoded result on success.

Definition at line 396 of file base64.c.

References `BASE64_DECODED_MOD_LEN`, `mappings_t::base64_mod`, `log_pedantic`, `mappings`, `st_alloc`, `st_avail_get()`, `st_data_get()`, `st_empty_out()`, `st_length_get()`, `st_length_set()`, `st_valid_avail()`, `st_valid_destination()`, `st_valid_tracked()`, and `mappings_t::values`.

Referenced by `base64_decode_opts()`, and `meta_data_user_build_storage_keys()`.

`stringer_t* base64_decode_opts (stringer_t * s, uint32_t opts, bool_t modified)`

Perform base64 decoding on a managed string.

Parameters:

s the managed string to be base64 decoded.

opts the options value of the managed string which will be allocated and receive the decoded output.

modified if true, use modified base64 encoding; if false, use normal base64 decoding.

Returns:

NULL on failure, or a pointer to a newly allocated managed string containing the decoded result on success.

Definition at line 355 of file `base64.c`.

References `base64_decode()`, `base64_decode_mod()`, `BASE64_DECODED_LEN`, `BASE64_DECODED_MOD_LEN`, `log_pedantic`, `st_alloc_opts()`, `st_empty_out()`, and `st_free()`.

Referenced by `meta_data_user_build_storage_keys()`.

`stringer_t* base64_encode (stringer_t * s, stringer_t * output)`

Perform base64 encoding on a managed string with padding and line splitting at `BASE64_LINE_WRAP_LENGTH` characters.

Parameters:

s the managed string to be base64 encoded.

output a managed string to receive the encoded output; if passed as NULL, one will be allocated to the caller.

Returns:

NULL on failure, or a pointer to the managed string containing the encoded result on success.

Definition at line 61 of file `base64.c`.

References `mappings_t::base64`, `BASE64_ENCODED_LEN`, `BASE64_LINE_WRAP_LENGTH`, `mappings_t::characters`, `log_error`, `log_pedantic`, `mappings`, `st_alloc`, `st_avail_get()`, `st_data_get()`, `st_empty_out()`, `st_length_get()`, `st_length_set()`, `st_valid_avail()`, `st_valid_destination()`, and `st_valid_tracked()`.

Referenced by `base64_encode_opts()`, `mail_insert_chunk_base64()`, and `mail_mime_encode_part()`.

`stringer_t* base64_encode_mod (stringer_t * s, stringer_t * output)`

Perform modified base64 encoding on a managed string without padding or line splitting.

Note:

In this function, the '+' and '/' characters are replaced with '-' and '_' respectively, making the output suitable for use in URL parameters without additional encoding.

Parameters:

s the managed string to be base64 modified-encoded.

output a managed string to receive the encoded output; if passed as NULL, one will be allocated to the caller.

Returns:

NULL on failure, or a newly allocated managed string containing the modified encoded result on success.

Definition at line 172 of file base64.c.

References BASE64_ENCODED_MOD_LEN, mappings_t::base64_mod, mappings_t::characters, debug_hook(), log_pedantic, mappings, st_alloc, st_avail_get(), st_data_get(), st_empty_out(), st_length_get(), st_length_set(), st_valid_avail(), st_valid_destination(), and st_valid_tracked().

Referenced by base64_encode_opts(), and meta_data_user_save_storage_keys().

stringer_t* base64_encode_opts (stringer_t * s, uint32_t opts, bool_t modified)

Perform base64 encoding on a managed string with padding and line splitting at BASE64_LINE_WRAP_LENGTH characters.

Parameters:

s the managed string to be base64 encoded.

opts the options value of the managed string which will be allocated and receive the encoded output.

modified if true, use modified base64 encoding; if false, use normal base64 encoding.

Returns:

NULL on failure, or a pointer to a newly allocated managed string containing the encoded result on success.

Definition at line 22 of file base64.c.

References base64_encode(), base64_encode_mod(), BASE64_ENCODED_LEN, BASE64_ENCODED_MOD_LEN, log_pedantic, st_alloc_opts(), st_empty_out(), and st_free().

stringer_t* decode_base64_modified_st (stringer_t * string)

stringer_t* encode_base64_modified_st (stringer_t * string)

size_t hex_count_st (stringer_t * s)

Count the number of hex characters in a string.

Parameters:

s the managed string to be scanned.

Returns:

the number of valid hexadecimal characters found in the string.

Definition at line 63 of file hex.c.

References hex_valid_chr(), and st_empty_out().

Referenced by hex_decode_st().

byte_t hex_decode_chr (uchar_t a, uchar_t b)

Decode a hex character pair as a single byte (usually for URL decoding).

Parameters:

- a* the higher order 4-bit hexadecimal character of the byte to be encoded.
- b* the lower order 4-bit hexadecimal character of the byte to be decoded.

Returns:

a byte containing the value of the decoded hexadecimal character pair, or 0 on failure.

Definition at line 258 of file hex.c.

References hex_valid_chr(), log_pedantic, and lower_chr().

Referenced by hex_decode_st(), http_data_value_decode(), qp_decode(), and url_decode().

stringer_t* hex_decode_st (stringer_t * *h*, stringer_t * *output*)

Convert a hex string into a binary data blob.

Note:

All hex strings should be composed of pairs of two hex characters representing individual bytes. Invalid hex characters will simply be ignored during processing.

Parameters:

- h* a managed string containing the input hex string to be decoded.
- output* if not NULL, a pointer to a managed string to contain the decoded binary output; if NULL, a new string will be allocated and returned to the caller.

Returns:

a pointer to a managed string containing the decoded output, or NULL on failure.

Definition at line 293 of file hex.c.

References hex_count_st(), hex_decode_chr(), hex_valid_chr(), log_pedantic, st_alloc, st_avail_get(), st_data_get(), st_empty_out(), st_length_get(), st_length_set(), st_valid_avail(), st_valid_destination(), and st_valid_tracked().

uchar_t* hex_encode_chr (byte_t *b*, uchar_t * *output*)

Converts a binary octet into a pair of lowercase hex characters. The result is returned inside a thread specific static character buffer. If a valid pointer is also supplied using the output variable then the result will also be written to the memory the pointer indicates.

Parameters:

- b* The binary octet being encoded.
- output* A pointer to the memory where the result should be stored or NULL if the result should only be returned.

Returns:

Returns the two character hex representation of the binary input using a thread localized character buffer.

Definition at line 89 of file hex.c.

References mm_wipe(), and number.

Referenced by hex_encode_st(), and hex_encode_st_debug().

stringer_t* hex_encode_st (stringer_t * *b*, stringer_t * *output*)

Convert a block of binary data into a hex string.

Parameters:

b a managed string containing the raw data to be encoded.

output if not NULL, a pointer to a managed string that will store the encoded output; if NULL, a new managed string will be allocated and returned to the caller.

Returns:

NULL on failure, or a pointer to a managed string containing the hex-encoded output on success.

Definition at line 129 of file hex.c.

References `hex_encode_chr()`, `log_pedantic`, `st_alloc`, `st_avail_get()`, `st_data_get()`, `st_empty_out()`, `st_length_get()`, `st_length_set()`, `st_valid_avail()`, `st_valid_destination()`, and `st_valid_tracked()`.

Referenced by `credential_alloc_auth()`.

stringer_t* hex_encode_st_debug (stringer_t * *input*, size_t *maxlen*)

Encode data in a human readable way, for debugging purposes.

Note:

All printable ASCII data will be displayed as-is; all other characters will be shown as formatted hex pairs.

Parameters:

input a pointer to a managed string containing the data to be formatted.

maxlen the maximum number of bytes to be displayed. If more characters are available, an ellipsis will be shown in the middle of the string to represent the abridged data, with only the leading and trailing characters returned as part of the display string.

Returns:

NULL on failure, or a pointer to a newly allocated managed string containing the encoded debug data on success.

Definition at line 183 of file hex.c.

References `chr_printable()`, `hex_encode_chr()`, `log_error`, `st_alloc`, `st_char_get()`, `st_length_get()`, and `st_length_set()`.

bool_t hex_valid_chr (uchar_t *c*)

hex.c

hex.c

Parameters:

c the character to be tested.

Returns:

true if the character is a valid hexadecimal character; false otherwise.

Definition at line 20 of file hex.c.

Referenced by `hex_count_st()`, `hex_decode_chr()`, `hex_decode_st()`, `hex_valid_st()`, `qp_decode()`, `url_decode()`, and `url_valid_st()`.

size_t hex_valid_st (stringer_t * *s*)

Determine whether a managed string is a properly formatted hex string and return the number found.

Note:

A valid hex string consists of only hexadecimal characters and whitespace, and the number of hex characters is divisible by two.

Parameters:

s the managed string to be tested.

Returns:

0 on failure, or the number of hexadecimal characters found in the managed string.

Definition at line 33 of file hex.c.

References hex_valid_chr(), and st_empty_out().

stringer_t* qp_decode (stringer_t * s)

qp.c

qp.c

Parameters:

s the managed string containing data to be decoded.

Returns:

a pointer to a managed string containing the 8-bit decoded output, or NULL on failure.

Definition at line 104 of file qp.c.

References hex_decode_chr(), hex_valid_chr(), log_pedantic, st_alloc, st_data_get(), st_empty_out(), st_length_get(), and st_length_set().

stringer_t* qp_encode (stringer_t * s)

Perform QP (quoted-printable) encoding of a string.

Parameters:

s a pointer to a managed string containing data to be encoded.

Returns:

a pointer to a managed string containing the QP encoded data, or NULL on failure.

Definition at line 22 of file qp.c.

References HEAP, JOINTED, log_pedantic, MANAGED_T, PLACER, QP_LINE_WRAP_LENGTH, st_alloc_opts(), st_append, st_data_get(), and st_empty_out().

Referenced by mail_build_signature(), and mail_mime_encode_part().

stringer_t* url_decode (stringer_t * s)

Decode a URL-encoded string into its original representation.

Parameters:

s a managed string containing the UR componentL to be decoded.

Returns:

NULL on failure, or a freshly allocated managed string containing the original data represented by the URL-encoded input on success.

Definition at line 132 of file url.c.

References `hex_decode_chr()`, `hex_valid_chr()`, `log_pedantic`, `st_alloc`, `st_data_get()`, `st_empty_out()`, `st_length_get()`, and `st_length_set()`.

stringer_t* url_encode (stringer_t * s)

Encode a data buffer as a valid URL component.

Parameters:

s a managed string containing the data to be encoded.

Returns:

NULL on failure, or a freshly allocated managed string containing the fully-escaped string suitable for use in a URL.

Definition at line 70 of file url.c.

References `HEAP`, `JOINTED`, `log_pedantic`, `MANAGED_T`, `PLACER`, `st_alloc_opts()`, `st_append`, `st_data_get()`, `st_empty_out()`, `st_length_set()`, and `url_valid_chr()`.

bool_t url_valid_chr (uchr_t c)

url.c

url.c

Parameters:

c the character to be examined.

Returns:

true if the character is valid in a URL or false if it must be escaped.

Definition at line 20 of file url.c.

Referenced by `url_encode()`, and `url_valid_st()`.

size_t url_valid_st (stringer_t * s)

Check a URL string for validity.

Note:

This function confirms that the URL consists of only legal characters and that all escaped sequences are also valid.

Parameters:

s a managed string containing the URL to be verified.

Returns:

0 on failure, or the number of valid characters in the URL if the string is valid.

Definition at line 33 of file url.c.

References `hex_valid_chr()`, `st_empty_out()`, and `url_valid_chr()`.

stringer_t* zbase32_decode (stringer_t * s)

zbase32.c

zbase32.c

Parameters:

s a managed string containing the data to be decoded.

Returns:

NULL on failure, or a freshly allocated managed string containing the zbase32-decoded data on success.

Definition at line 69 of file zbase32.c.

References `log_pedantic`, `mappings`, `st_alloc`, `st_data_get()`, `st_empty_out()`, `st_free()`, `st_length_set()`, `mappings_t::values`, and `mappings_t::zbase32`.

Referenced by `sess_get()`.

stringer_t* zbase32_encode (stringer_t * s)

Encode data as a zbase32 string.

Parameters:

s a managed string containing the data to be encoded.

Returns:

NULL on failure, or a freshly allocated managed string containing the zbase32-encoded data on success.

Definition at line 21 of file zbase32.c.

References `mappings_t::characters`, `log_pedantic`, `mappings`, `st_alloc`, `st_data_get()`, `st_empty_out()`, `st_length_set()`, and `mappings_t::zbase32`.

Referenced by `sess_token()`.

Variable Documentation

mappings_t mappings

Definition at line 15 of file mappings.c.

Referenced by `base64_decode()`, `base64_decode_mod()`, `base64_encode()`, `base64_encode_mod()`, `zbase32_decode()`, and `zbase32_encode()`.

magma/core/encodings/hex.c File Reference

Functions for encoding/decoding hexadecimal data.

```
#include "magma.h"
```

Functions

- **bool_t hex_valid_chr (uchar_t c)**
- *Determine whether a character is a valid hexadecimal (base 16) character. size_t **hex_valid_st (stringer_t *s)***
- *Determine whether a managed string is a properly formatted hex string and return the number found. size_t **hex_count_st (stringer_t *s)***
- *Count the number of hex characters in a string. uchar_t * **hex_encode_chr (byte_t b, uchar_t *output)***
- **stringer_t * hex_encode_st (stringer_t *b, stringer_t *output)**
- *Convert a block of binary data into a hex string. stringer_t * **hex_encode_st_debug (stringer_t *input, size_t maxlen)***
- *Encode data in a human readable way, for debugging purposes. byte_t **hex_decode_chr (uchar_t a, uchar_t b)***
- *Decode a hex character pair as a single byte (usually for URL decoding). stringer_t * **hex_decode_st (stringer_t *h, stringer_t *output)***

Convert a hex string into a binary data blob.

Detailed Description

Functions for encoding/decoding hexadecimal data.

Definition in file **hex.c**.

Function Documentation

size_t hex_count_st (stringer_t * s)

Count the number of hex characters in a string.

Parameters:

s the managed string to be scanned.

Returns:

the number of valid hexadecimal characters found in the string.

Definition at line 63 of file hex.c.

References `hex_valid_chr()`, and `st_empty_out()`.

Referenced by `hex_decode_st()`.

byte_t hex_decode_chr (uchar_t a, uchar_t b)

Decode a hex character pair as a single byte (usually for URL decoding).

Parameters:

- a* the higher order 4-bit hexadecimal character of the byte to be encoded.
- b* the lower order 4-bit hexadecimal character of the byte to be decoded.

Returns:

a byte containing the value of the decoded hexadecimal character pair, or 0 on failure.

Definition at line 258 of file hex.c.

References hex_valid_chr(), log_pedantic, and lower_chr().

Referenced by hex_decode_st(), http_data_value_decode(), qp_decode(), and url_decode().

stringer_t* hex_decode_st (stringer_t * *h*, stringer_t * *output*)

Convert a hex string into a binary data blob.

Note:

All hex strings should be composed of pairs of two hex characters representing individual bytes. Invalid hex characters will simply be ignored during processing.

Parameters:

- h* a managed string containing the input hex string to be decoded.
- output* if not NULL, a pointer to a managed string to contain the decoded binary output; if NULL, a new string will be allocated and returned to the caller.

Returns:

a pointer to a managed string containing the decoded output, or NULL on failure.

Definition at line 293 of file hex.c.

References hex_count_st(), hex_decode_chr(), hex_valid_chr(), log_pedantic, st_alloc, st_avail_get(), st_data_get(), st_empty_out(), st_length_get(), st_length_set(), st_valid_avail(), st_valid_destination(), and st_valid_tracked().

uchar_t* hex_encode_chr (byte_t *b*, uchar_t * *output*)

Converts a binary octet into a pair of lowercase hex characters. The result is returned inside a thread specific static character buffer. If a valid pointer is also supplied using the output variable then the result will also be written to the memory the pointer indicates.

Parameters:

- b* The binary octet being encoded.
- output* A pointer to the memory where the result should be stored or NULL if the result should only be returned.

Returns:

Returns the two character hex representation of the binary input using a thread localized character buffer.

Definition at line 89 of file hex.c.

References mm_wipe(), and number.

Referenced by hex_encode_st(), and hex_encode_st_debug().

stringer_t* hex_encode_st (stringer_t * *b*, stringer_t * *output*)

Convert a block of binary data into a hex string.

Parameters:

b a managed string containing the raw data to be encoded.
output if not NULL, a pointer to a managed string that will store the encoded output; if NULL, a new managed string will be allocated and returned to the caller.

Returns:

NULL on failure, or a pointer to a managed string containing the hex-encoded output on success.

Definition at line 129 of file hex.c.

References `hex_encode_chr()`, `log_pedantic`, `st_alloc`, `st_avail_get()`, `st_data_get()`, `st_empty_out()`, `st_length_get()`, `st_length_set()`, `st_valid_avail()`, `st_valid_destination()`, and `st_valid_tracked()`.

Referenced by `credential_alloc_auth()`.

stringer_t* hex_encode_st_debug (stringer_t * input, size_t maxlen)

Encode data in a human readable way, for debugging purposes.

Note:

All printable ASCII data will be displayed as-is; all other characters will be shown as formatted hex pairs.

Parameters:

input a pointer to a managed string containing the data to be formatted.
maxlen the maximum number of bytes to be displayed. If more characters are available, an ellipsis will be shown in the middle of the string to represent the abridged data, with only the leading and trailing characters returned as part of the display string.

Returns:

NULL on failure, or a pointer to a newly allocated managed string containing the encoded debug data on success.

Definition at line 183 of file hex.c.

References `chr_printable()`, `hex_encode_chr()`, `log_error`, `st_alloc`, `st_char_get()`, `st_length_get()`, and `st_length_set()`.

bool_t hex_valid_chr (uchar_t c)

Determine whether a character is a valid hexadecimal (base 16) character.

hex.c**Parameters:**

c the character to be tested.

Returns:

true if the character is a valid hexadecimal character; false otherwise.

Definition at line 20 of file hex.c.

Referenced by `hex_count_st()`, `hex_decode_chr()`, `hex_decode_st()`, `hex_valid_st()`, `qp_decode()`, `url_decode()`, and `url_valid_st()`.

size_t hex_valid_st (stringer_t * s)

Determine whether a managed string is a properly formatted hex string and return the number found.

Note:

A valid hex string consists of only hexadecimal characters and whitespace, and the number of hex characters is divisible by two.

Parameters:

s the managed string to be tested.

Returns:

0 on failure, or the number of hexadecimal characters found in the managed string.

Definition at line 33 of file hex.c.

References `hex_valid_chr()`, and `st_empty_out()`.

magma/core/encodings/mappings.c File Reference

Character-to-value mappings employed by the various base64 encoders.

```
#include "magma.h"
```

Variables

- `mappings_t mappings`

Detailed Description

Character-to-value mappings employed by the various base64 encoders.

Definition in file `mappings.c`.

Variable Documentation

`mappings_t mappings`

```
Initial value: {
    .zbase32 = {
        .characters = "ybndrfg8ejkmcpqxotluwisa345h769",
        .values = {
            255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 18, 255, 25, 26,
27, 30, 29, 7, 31, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 24, 1, 12, 3, 8, 5, 6, 28, 21, 9, 10, 255, 11, 2, 16,
13, 14, 4, 22, 17, 19, 255, 20, 15, 0, 23, 255, 255, 255, 255, 255
        },
    },
    .base64 = {
        .characters =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/",
        .values = {
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 63, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 0, 0, 0, 0, 0, 0,
0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 25, 0, 0, 0, 0, 0, 0, 26, 27, 28, 29, 30,
31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 0, 0, 0, 0, 0
        },
    },
    .base64_mod = {
        .characters =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_",
        .values = {
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 62, 0, 0, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 0, 0, 0, 0, 0, 0,
0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 25, 0, 0, 0, 0, 63, 0, 26, 27, 28, 29,
30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
45, 46, 47, 48, 49, 50, 51, 0, 0, 0, 0, 0
        },
    }
}
```

```
}
```

Definition at line 15 of file mappings.c.

Referenced by `base64_decode()`, `base64_decode_mod()`, `base64_encode()`, `base64_encode_mod()`, `zbase32_decode()`, and `zbase32_encode()`.

magma/core/host/mappings.c File Reference

Map numeric system values to their string equivalents.

```
#include "magma.h"
```

Functions

- **chr_t * errno_name** (int *errnum*, char **buffer*, size_t *length*)
- *Translate an error number into a descriptive, human-readable message.* **chr_t * signal_name** (int *signal*, char **buffer*, size_t *length*)

Translate a numeric signal into a human readable name.

Detailed Description

Map numeric system values to their string equivalents.

Definition in file **mappings.c**.

Function Documentation

chr_t* errno_name (int *errnum*, char * *buffer*, size_t *length*)

Translate an error number into a descriptive, human-readable message.

mappings.c

Parameters:

error the error number (errno) to be looked up.

buffer a pointer to a character buffer that will store the error description string.

length the length, in bytes, of the output buffer, which should be at minimum of 32 bytes.

Returns:

a pointer to a null-terminated string containing the descriptive error message.

Definition at line 22 of file mappings.c.

References log_pedantic.

chr_t* signal_name (int *signal*, char * *buffer*, size_t *length*)

Translate a numeric signal into a human readable name.

Parameters:

signal the input signal number.

buffer a buffer to receive the signal description.

length the length of the buffer to receive the output, which should be greater than 32 bytes.

Returns:

a pointer to a null-terminated string containing the textual name of the specified signal.

Definition at line 51 of file mappings.c.

References log_pedantic.

Referenced by `signal_segfault()`, `signal_shutdown()`, and `signal_status()`.

magma/core/encodings/qp.c File Reference

Functions for encoding/decoding quoted printable data, as described by RFC 2045, section 6.7.

```
#include "magma.h"
```

Functions

- **stringer_t * qp_encode (stringer_t *s)**
 - *Perform QP (quoted-printable) encoding of a string.* **stringer_t * qp_decode (stringer_t *s)**
Perform QP (quoted-printable) decoding of a string.
-

Detailed Description

Functions for encoding/decoding quoted printable data, as described by RFC 2045, section 6.7.

Note:

This function operates on standard 8-bit characters, transforming non-printable characters into printable ones. It is used as a MIME content encoding, and wraps lines at 76 characters.

Definition in file **qp.c**.

Function Documentation

stringer_t* qp_decode (stringer_t * s)

Perform QP (quoted-printable) decoding of a string.

qp.c

Parameters:

s the managed string containing data to be decoded.

Returns:

a pointer to a managed string containing the 8-bit decoded output, or NULL on failure.

Definition at line 104 of file qp.c.

References `hex_decode_chr()`, `hex_valid_chr()`, `log_pedantic`, `st_alloc`, `st_data_get()`, `st_empty_out()`, `st_length_get()`, and `st_length_set()`.

stringer_t* qp_encode (stringer_t * s)

Perform QP (quoted-printable) encoding of a string.

Parameters:

s a pointer to a managed string containing data to be encoded.

Returns:

a pointer to a managed string containing the QP encoded data, or NULL on failure.

Definition at line 22 of file qp.c.

References `HEAP`, `JOINTED`, `log_pedantic`, `MANAGED_T`, `PLACER`, `QP_LINE_WRAP_LENGTH`, `st_alloc_opts()`, `st_append`, `st_data_get()`, and `st_empty_out()`.

Referenced by `mail_build_signature()`, and `mail_mime_encode_part()`.

magma/core/encodings/url.c File Reference

Functions used to encode and decode website URLs.

```
#include "magma.h"
```

Functions

- **bool_t url_valid_chr (uchr_t c)**
- *Determine whether a given character is a valid character in a URL.* **size_t url_valid_st (stringer_t *s)**
- *Check a URL string for validity.* **stringer_t * url_encode (stringer_t *s)**
- *Encode a data buffer as a valid URL component.* **stringer_t * url_decode (stringer_t *s)**

Decode a URL-encoded string into its original representation.

Detailed Description

Functions used to encode and decode website URLs.

Definition in file **url.c**.

Function Documentation

stringer_t* url_decode (stringer_t * s)

Decode a URL-encoded string into its original representation.

Parameters:

s a managed string containing the UR componentL to be decoded.

Returns:

NULL on failure, or a freshly allocated managed string containing the original data represented by the URL-encoded input on success.

Definition at line 132 of file url.c.

References `hex_decode_chr()`, `hex_valid_chr()`, `log_pedantic`, `st_alloc`, `st_data_get()`, `st_empty_out()`, `st_length_get()`, and `st_length_set()`.

stringer_t* url_encode (stringer_t * s)

Encode a data buffer as a valid URL component.

Parameters:

s a managed string containing the data to be encoded.

Returns:

NULL on failure, or a freshly allocated managed string containing the fully-escaped string suitable for use in a URL.

Definition at line 70 of file url.c.

References `HEAP`, `JOINTED`, `log_pedantic`, `MANAGED_T`, `PLACER`, `st_alloc_opts()`, `st_append`, `st_data_get()`, `st_empty_out()`, `st_length_set()`, and `url_valid_chr()`.

bool_t url_valid_chr (uchr_t c)

Determine whether a given character is a valid character in a URL.

url.c

Parameters:

c the character to be examined.

Returns:

true if the character is valid in a URL or false if it must be escaped.

Definition at line 20 of file url.c.

Referenced by `url_encode()`, and `url_valid_st()`.

size_t url_valid_st (stringer_t * s)

Check a URL string for validity.

Note:

This function confirms that the URL consists of only legal characters and that all escaped sequences are also valid.

Parameters:

s a managed string containing the URL to be verified.

Returns:

0 on failure, or the number of valid characters in the URL if the string is valid.

Definition at line 33 of file url.c.

References `hex_valid_chr()`, `st_empty_out()`, and `url_valid_chr()`.

magma/core/encodings/zbase32.c File Reference

A modified base32 encoding routine (zbase32) which selects characters to enhance readability. zbase32 strings may be used in URLs without any further encoding.

```
#include "magma.h"
```

Functions

- **stringer_t * zbase32_encode (stringer_t *s)**
 - *Encode data as a zbase32 string. stringer_t * zbase32_decode (stringer_t *s)*
Decode a zbase32 string.
-

Detailed Description

A modified base32 encoding routine (zbase32) which selects characters to enhance readability. zbase32 strings may be used in URLs without any further encoding.

Definition in file **zbase32.c**.

Function Documentation

stringer_t* zbase32_decode (stringer_t * s)

Decode a zbase32 string.

zbase32.c

Parameters:

s a managed string containing the data to be decoded.

Returns:

NULL on failure, or a freshly allocated managed string containing the zbase32-decoded data on success.

Definition at line 69 of file zbase32.c.

References `log_pedantic`, `mappings`, `st_alloc`, `st_data_get()`, `st_empty_out()`, `st_free()`, `st_length_set()`, `mappings_t::values`, and `mappings_t::zbase32`.

Referenced by `sess_get()`.

stringer_t* zbase32_encode (stringer_t * s)

Encode data as a zbase32 string.

Parameters:

s a managed string containing the data to be encoded.

Returns:

NULL on failure, or a freshly allocated managed string containing the zbase32-encoded data on success.

Definition at line 21 of file zbase32.c.

References mappings_t::characters, log_pedantic, mappings, st_alloc, st_data_get(), st_empty_out(), st_length_set(), and mappings_t::zbase32.

Referenced by sess_token().

magma/core/hash/adler.c File Reference

An x86 implementation of the Adler hash algorithm.

```
#include "magma.h"
```

Functions

- `uint32_t hash_adler32 (void *buffer, size_t length)`

Return an Adler-32 hash of the specified data.

Detailed Description

An x86 implementation of the Adler hash algorithm.

Definition in file `adler.c`.

Function Documentation

`uint32_t hash_adler32 (void * buffer, size_t length)`

Return an Adler-32 hash of the specified data.

Parameters:

buffer a pointer to the data to be hashed.

length the length, in bytes, of the data to be hashed.

Returns:

a 32 bit number containing the Adler-32 hash of the data.

Definition at line 21 of file `adler.c`.

Referenced by `compress_bzip()`, `compress_import()`, `compress_lzo()`, `compress_zlib()`, `decompress_bzip()`, `decompress_lzo()`, `decompress_zlib()`, `scramble_decrypt()`, `scramble_encrypt()`, and `scramble_import()`.

magma/core/hash/crc.c File Reference

An x86 implementation of the 32-bit and 64-bit CRC algorithms.

```
#include "magma.h"
```

Defines

- `#define A1 A`
- `#define A(x) ((x) & 0xFF)`
- `#define B(x) (((x) >> 8) & 0xFF)`
- `#define C(x) (((x) >> 16) & 0xFF)`
- `#define D(x) ((x) >> 24)`
- `#define S8(x) ((x) >> 8)`
- `#define S32(x) ((x) >> 32)`

Functions

- `uint32_t hash_crc32_update` (void *buffer, size_t length, uint32_t crc)
- *Update a 64-bit CRC value with a check of additional data.* `uint32_t hash_crc32` (void *buffer, size_t length)
- *Get 32-bit CRC value for a specified block of data.* `uint64_t hash_crc64_update` (void *buffer, size_t length, uint64_t crc)
- *Update a 64-bit CRC value with a check of additional data.* `uint64_t hash_crc64` (void *buffer, size_t length)

Get 64-bit CRC value for a specified block of data. Variables

- `const uint32_t hash_crc32_table` [8][256]
- `const uint64_t hash_crc64_table` [4][256]

Detailed Description

An x86 implementation of the 32-bit and 64-bit CRC algorithms.

Definition in file `crc.c`.

Define Documentation

`#define A(x) ((x) & 0xFF)`

Definition at line 16 of file `crc.c`.

Referenced by `hash_crc32_update()`, and `hash_crc64_update()`.

`#define A1 A`

Definition at line 15 of file `crc.c`.

Referenced by `hash_crc64_update()`.

#define B(x) (((x) >> 8) & 0xFF)

Definition at line 17 of file crc.c.

Referenced by hash_crc32_update(), and hash_crc64_update().

#define C(x) (((x) >> 16) & 0xFF)

Definition at line 18 of file crc.c.

Referenced by hash_crc32_update(), and hash_crc64_update().

#define D(x) ((x) >> 24)

Definition at line 19 of file crc.c.

Referenced by hash_crc32_update(), and hash_crc64_update().

#define S32(x) ((x) >> 32)

Definition at line 22 of file crc.c.

Referenced by hash_crc64_update().

#define S8(x) ((x) >> 8)

Definition at line 21 of file crc.c.

Referenced by hash_crc32_update(), and hash_crc64_update().

Function Documentation

uint32_t hash_crc32 (void * *buffer*, size_t *length*)

Get 32-bit CRC value for a specified block of data.

Parameters:

buffer a pointer to the data to be checked.

length the length, in bytes, of the input buffer.

Returns:

the 32-bit CRC value of the specified data.

Definition at line 67 of file crc.c.

References hash_crc32_update().

uint32_t hash_crc32_update (void * *buffer*, size_t *length*, uint32_t *crc*)

Update a 64-bit CRC value with a check of additional data.

Parameters:

buffer a pointer to the data to be checked.
length the length, in bytes, of the input buffer.
crc the previously computed CRC value, or 0 if this is the initial pass.

Returns:

the updated 32-bit CRC value of the specified data.
Definition at line 34 of file `crc.c`.
References `A`, `B`, `C`, `D`, `hash_crc32_table`, and `S8`.
Referenced by `hash_crc32()`.

uint64_t hash_crc64 (void * *buffer*, size_t *length*)

Get 64-bit CRC value for a specified block of data.

Parameters:

buffer a pointer to the data to be checked.
length the length, in bytes, of the input buffer.

Returns:

the 64-bit CRC value of the specified data.
Definition at line 108 of file `crc.c`.
References `hash_crc64_update()`.
Referenced by `smtp_bounce()`, and `smtp_reply()`.

uint64_t hash_crc64_update (void * *buffer*, size_t *length*, uint64_t *crc*)

Update a 64-bit CRC value with a check of additional data.

Parameters:

buffer a pointer to the data to be checked.
length the length, in bytes, of the input buffer.
crc the previously computed CRC value, or 0 if this is the initial pass.

Returns:

the updated 64-bit CRC value of the specified data.
Definition at line 78 of file `crc.c`.
References `A`, `A1`, `B`, `C`, `D`, `hash_crc64_table`, `S32`, and `S8`.
Referenced by `hash_crc64()`.

Variable Documentation

const uint32_t hash_crc32_table

Definition at line 113 of file `crc.c`.

Referenced by hash_crc32_update().

const uint64_t hash_crc64_table

Definition at line 637 of file crc.c.

Referenced by hash_crc64_update().

magma/core/hash/fletcher.c File Reference

An implementation of the Fletcher hash algorithm.

```
#include "magma.h"
```

Functions

- `uint32_t hash_fletcher32 (void *buffer, size_t length)`

Computer a 32-bit Fletcher hash for a block of data.

Detailed Description

An implementation of the Fletcher hash algorithm.

Definition in file `fletcher.c`.

Function Documentation

`uint32_t hash_fletcher32 (void * buffer, size_t length)`

Computer a 32-bit Fletcher hash for a block of data.

Parameters:

buffer a pointer to the data buffer to be checked.

length the length, in bytes, of the data to be checked.

Returns:

a 32-bit number containing the Fletcher hash of the specified data.

Definition at line 21 of file `fletcher.c`.

Referenced by `hashed_bucket()`.

magma/core/hash/hash.h File Reference

Declarations for hash functions that have been implemented internally.

Functions

- `uint32_t hash_crc32` (void *buffer, size_t length)
 - *Get 32-bit CRC value for a specified block of data.* `uint64_t hash_crc64` (void *buffer, size_t length)
 - *Get 64-bit CRC value for a specified block of data.* `uint32_t hash_crc32_update` (void *buffer, size_t length, uint32_t crc)
 - *Update a 64-bit CRC value with a check of additional data.* `uint64_t hash_crc64_update` (void *buffer, size_t length, uint64_t crc)
 - *Update a 64-bit CRC value with a check of additional data.* `uint32_t hash_adler32` (void *buffer, size_t length)
 - *Return an Adler-32 hash of the specified data.* `uint32_t hash_murmur32` (void *buffer, size_t length)
 - *Generate a 32-bit Murmur hash of a block of data.* `uint64_t hash_murmur64` (void *buffer, size_t length)
 - *Generate a 64-bit Murmur hash of a block of data.* `uint32_t hash_fletcher32` (void *buffer, size_t length)
- Computer a 32-bit Fletcher hash for a block of data.*
-

Detailed Description

Declarations for hash functions that have been implemented internally.

Definition in file **hash.h**.

Function Documentation

uint32_t hash_adler32 (void * *buffer*, size_t *length*)

Return an Adler-32 hash of the specified data.

Parameters:

buffer a pointer to the data to be hashed.

length the length, in bytes, of the data to be hashed.

Returns:

a 32 bit number containing the Adler-32 hash of the data.

Definition at line 21 of file `adler.c`.

Referenced by `compress_bzip()`, `compress_import()`, `compress_lzo()`, `compress_zlib()`, `decompress_bzip()`, `decompress_lzo()`, `decompress_zlib()`, `scramble_decrypt()`, `scramble_encrypt()`, and `scramble_import()`.

uint32_t hash_crc32 (void * *buffer*, size_t *length*)

Get 32-bit CRC value for a specified block of data.

Parameters:

buffer a pointer to the data to be checked.

length the length, in bytes, of the input buffer.

Returns:

the 32-bit CRC value of the specified data.

Definition at line 67 of file `crc.c`.

References `hash_crc32_update()`.

uint32_t hash_crc32_update (void * *buffer*, size_t *length*, uint32_t *crc*)

Update a 64-bit CRC value with a check of additional data.

Parameters:

buffer a pointer to the data to be checked.

length the length, in bytes, of the input buffer.

crc the previously computed CRC value, or 0 if this is the initial pass.

Returns:

the updated 32-bit CRC value of the specified data.

Definition at line 34 of file `crc.c`.

References A, B, C, D, `hash_crc32_table`, and S8.

Referenced by `hash_crc32()`.

uint64_t hash_crc64 (void * *buffer*, size_t *length*)

Get 64-bit CRC value for a specified block of data.

Parameters:

buffer a pointer to the data to be checked.

length the length, in bytes, of the input buffer.

Returns:

the 64-bit CRC value of the specified data.

Definition at line 108 of file `crc.c`.

References `hash_crc64_update()`.

Referenced by `smtp_bounce()`, and `smtp_reply()`.

uint64_t hash_crc64_update (void * *buffer*, size_t *length*, uint64_t *crc*)

Update a 64-bit CRC value with a check of additional data.

Parameters:

buffer a pointer to the data to be checked.

length the length, in bytes, of the input buffer.

crc the previously computed CRC value, or 0 if this is the initial pass.

Returns:

the updated 64-bit CRC value of the specified data.

Definition at line 78 of file `crc.c`.

References A, A1, B, C, D, hash_crc64_table, S32, and S8.

Referenced by hash_crc64().

uint32_t hash_fletcher32 (void * *buffer*, size_t *length*)

Computer a 32-bit Fletcher hash for a block of data.

Parameters:

buffer a pointer to the data buffer to be checked.

length the length, in bytes, of the data to be checked.

Returns:

a 32-bit number containing the Fletcher hash of the specified data.

Definition at line 21 of file fletcher.c.

Referenced by hashed_bucket().

uint32_t hash_murmur32 (void * *buffer*, size_t *length*)

Generate a 32-bit Murmur hash of a block of data.

Parameters:

buffer a pointer to the block of data to be hashed.

length the length, in bytes, of the block of data to be hashed.

Returns:

the 32-bit value of the Murmur hash of the specified block of data.

Definition at line 22 of file murmur.c.

References data.

uint64_t hash_murmur64 (void * *buffer*, size_t *length*)

Generate a 64-bit Murmur hash of a block of data.

Parameters:

buffer a pointer to the block of data to be hashed.

length the length, in bytes, of the block of data to be hashed.

Returns:

the 64-bit value of the Murmur hash of the specified block of data.

Definition at line 77 of file murmur.c.

References data.

magma/core/hash/murmur.c File Reference

An x64 implementation of the Murmur hash function.

```
#include "magma.h"
```

Functions

- `uint32_t hash_murmur32` (void **buffer*, `size_t length`)
 - *Generate a 32-bit Murmur hash of a block of data.* `uint64_t hash_murmur64` (void **buffer*, `size_t length`)
Generate a 64-bit Murmur hash of a block of data.
-

Detailed Description

An x64 implementation of the Murmur hash function.

Definition in file `murmur.c`.

Function Documentation

`uint32_t hash_murmur32` (void * *buffer*, `size_t length`)

Generate a 32-bit Murmur hash of a block of data.

Parameters:

buffer a pointer to the block of data to be hashed.
length the length, in bytes, of the block of data to be hashed.

Returns:

the 32-bit value of the Murmur hash of the specified block of data.
Definition at line 22 of file `murmur.c`.
References `data`.

`uint64_t hash_murmur64` (void * *buffer*, `size_t length`)

Generate a 64-bit Murmur hash of a block of data.

Parameters:

buffer a pointer to the block of data to be hashed.
length the length, in bytes, of the block of data to be hashed.

Returns:

the 64-bit value of the Murmur hash of the specified block of data.
Definition at line 77 of file `murmur.c`.
References `data`.

magma/core/host/files.c File Reference

Generic system file I/O operations.

```
#include "magma.h"
```

Functions

- **int_t file_read** (char *name, **stringer_t** *output)
- *Get the contents of a file on disk.* **stringer_t * file_load** (char *name)
- *Get the contents of a file on disk.* **int_t get_temp_file_handle** (**chr_t** *pdir, **stringer_t** **tmpname)
- *Get a file handle to a temporary file created in a specified directory.* **bool_t file_accessible** (const **chr_t** *path)
- *Determine whether a given filename or directory path is accessible by a user.* **bool_t file_readwritable** (const **chr_t** *path)
- *Determine whether a given filename or directory path is readable and writable by a user.* **bool_t file_world_accessible** (const **chr_t** *path)

Determine whether a given filename or directory path is readable or writable by other users.

Detailed Description

Generic system file I/O operations.

\$Author\$ \$Author\$ \$Revision\$

Definition in file **files.c**.

Function Documentation

bool_t file_accessible (const **chr_t** * *path*)

Determine whether a given filename or directory path is accessible by a user.

Parameters:

path a null-terminated string with the name of the filename or directory to be checked for existence/access.

Returns:

true if the pathname exists and is readable, or false otherwise.

Definition at line 160 of file files.c.

stringer_t* file_load (char * *name*)

Get the contents of a file on disk.

files.c

Parameters:

name a character pointer to the full pathname of the file to be opened.

Returns:

NULL on failure, or a managed string containing all the data in the file.

Definition at line 56 of file files.c.

References log_info, st_alloc, st_avail_get(), st_data_get(), and st_length_set().

Referenced by `adjust_message_encryption()`, `config_load_file_settings()`, and `ssl_start()`.

`int_t file_read (char * name, stringer_t * output)`

Get the contents of a file on disk.

See also:

`file_load()`

Note:

This is similar in function to **`file_load()`** but there is no `read()` length checking, so it can be used on non-regular files.

Parameters:

name a character pointer to the full pathname of the file to be opened.

output a managed string where the results of the file read operation will be stored.

Returns:

-1 on failure or the number of bytes read from the file on success.

Definition at line 22 of file `files.c`.

References `log_info`, `log_pedantic`, `MEMORYBUF`, `st_avail_get()`, `st_data_get()`, and `st_length_set()`.

Referenced by `process_kill()`.

`bool_t file_readwritable (const chr_t * path)`

Determine whether a given filename or directory path is readable and writable by a user.

Parameters:

path a null-terminated string with the name of the filename or directory to be checked for permissions.

Returns:

true if the pathname exists and is readable and writable, or false otherwise.

Definition at line 170 of file `files.c`.

Referenced by `servers_validate()`.

`bool_t file_world_accessible (const chr_t * path)`

Determine whether a given filename or directory path is readable or writable by other users.

Parameters:

path a null-terminated string with the name of the filename or directory to be checked for world permissions.

Returns:

true if the pathname exists and is readable or writable by others, or false otherwise.

Definition at line 180 of file `files.c`.

Referenced by `servers_validate()`, and `ssl_start()`.

int_t get_temp_file_handle (chr_t * *pdir*, stringer_t ** *tmpname*)

Get a file handle to a temporary file created in a specified directory.

Parameters:

pdir the parent directory in which to create the temporary file, or NULL for the default spool dir.

tmpname an optional pointer to a managed string to receive the name of the created temp file.

Returns:

-1 on failure or the new temporary file's file descriptor on success.

Definition at line 104 of file files.c.

References `CONSTANT`, `HEAP`, `JOINTED`, `log_pedantic`, `MAGMA_SPOOL_BASE`, `MANAGED_T`, `ns_length_get()`, `spool_path()`, `st_append`, `st_char_get()`, `st_dupe()`, `st_dupe_opts()`, `st_free()`, and `st_import()`.

Referenced by `adjust_message_encryption()`.

magma/core/host/folder.c File Reference

Functions for folder operations.

```
#include "magma.h"
```

Functions

- **int_t folder_exists** (**stringer_t** *path, **bool_t** create)

Check to see if a specified directory exists, or if specified, create it if it doesn't exist.

Detailed Description

Functions for folder operations.

Definition in file **folder.c**.

Function Documentation

int_t folder_exists (**stringer_t** * path, **bool_t** create)

Check to see if a specified directory exists, or if specified, create it if it doesn't exist.

folder.c

Parameters:

path a managed string containing the full pathname of the directory.

create if true, attempt to create the directory if it does not already exist.

Returns:

-1 if the directory doesn't exist or couldn't be created, 0 if the directory exists, or 1 if the directory was created.

Definition at line 21 of file folder.c.

References `st_char_get()`.

Referenced by `log_start()`, and `spool_check()`.

magma/core/host/host.c File Reference

Functions to retrieve information about the operating system.

```
#include "magma.h"
```

Functions

- **stringer_t * host_platform** (stringer_t *output)
 - *Get a description of the local operating system.* **stringer_t * host_version** (stringer_t *output)
- Get release information about the local OS.*
-

Detailed Description

Functions to retrieve information about the operating system.

Definition in file **host.c**.

Function Documentation

stringer_t* host_platform (stringer_t * output)

Get a description of the local operating system.

host.c

Parameters:

output a pointer to a managed string to receive the OS description.

Returns:

NULL on failure, or the user-specified managed string containing the OS info on success.

Definition at line 21 of file host.c.

References log_pedantic, ns_length_get(), st_avail_get(), and st_sprint().

Referenced by lib_load().

stringer_t* host_version (stringer_t * output)

Get release information about the local OS.

Parameters:

output a pointer to a managed string to receive the release information.

Returns:

NULL on failure, or the user-specified managed string containing the release info on success.

Definition at line 47 of file host.c.

References log_pedantic, ns_length_get(), st_avail_get(), and st_sprint().

Referenced by lib_load().

magma/core/host/host.h File Reference

Provide access to system interfaces.

Defines

- `#define MAGMA_PROC_PATH "/proc"`

Enumerations

- `enum { MAGMA_SPOOL_BASE = 0, MAGMA_SPOOL_DATA = 1, MAGMA_SPOOL_SCAN = 2 }`

Functions

- `stringer_t * file_load` (char *name)
- *files.c* `int_t file_read` (char *name, `stringer_t` *output)
- *Get the contents of a file on disk.* `int_t get_temp_file_handle` (`chr_t` *pdir, `stringer_t` **tmpname)
- *Get a file handle to a temporary file created in a specified directory.* `bool_t file_accessible` (const `chr_t` *path)
- *Determine whether a given filename or directory path is accessible by a user.* `bool_t file_readwritable` (const `chr_t` *path)
- *Determine whether a given filename or directory path is readable and writable by a user.* `bool_t file_world_accessible` (const `chr_t` *path)
- *Determine whether a given filename or directory path is readable or writable by other users.* `stringer_t * host_platform` (`stringer_t` *output)
- *host.c* `stringer_t * host_version` (`stringer_t` *output)
- *Get release information about the local OS.* `chr_t * errno_name` (int errnum, char *buffer, size_t length)
- *mappings.c* `chr_t * signal_name` (int signal, char *buffer, size_t length)
- *Translate a numeric signal into a human readable name.* `int_t folder_exists` (`stringer_t` *path, `bool_t` create)
- *folder.c* `int_t process_kill` (`stringer_t` *name, `int_t` signal, `int_t` wait)
- *process.c* `int_t spool_check` (`stringer_t` *path)
- *spool.c* `int_t spool_check_file` (const char *file, const struct stat *info, int type)
- *An internal function used by the ftw() function to cleanup the file contents of a spool directory.* `int_t spool_cleanup` (void)
- *Clean up the file contents in the spool base directory.* `uint64_t spool_error_stats` (void)
- *Get the number of spool errors encountered.* `int_t spool_mktemp` (`int_t` spool, `chr_t` *prefix)
- *Create a temporary file in a specified spool directory.* `stringer_t * spool_path` (`int_t` spool)
- *Get the full path to a requested spool directory.* `bool_t spool_start` (void)
- *Create and check the spool directories, and clean them for use.* `void spool_stop` (void)

Clean up the file contents in the spool base directory.

Detailed Description

Provide access to system interfaces.

Definition in file `host.h`.

Define Documentation

`#define MAGMA_PROC_PATH "/proc"`

Definition at line 16 of file host.h.

Referenced by process_kill().

Enumeration Type Documentation

anonymous enum

Enumerator:

MAGMA_SPOOL_BASE

MAGMA_SPOOL_DATA

MAGMA_SPOOL_SCAN

Definition at line 19 of file host.h.

Function Documentation

chr_t* errno_name (int *errnum*, char * *buffer*, size_t *length*)

mappings.c

mappings.c

Parameters:

error the error number (errno) to be looked up.

buffer a pointer to a character buffer that will store the error description string.

length the length, in bytes, of the output buffer, which should be at minimum of 32 bytes.

Returns:

a pointer to a null-terminated string containing the descriptive error message.

Definition at line 22 of file mappings.c.

References log_pedantic.

bool_t file_accessible (const chr_t * *path*)

Determine whether a given filename or directory path is accessible by a user.

Parameters:

path a null-terminated string with the name of the filename or directory to be checked for existence/access.

Returns:

true if the pathname exists and is readable, or false otherwise.

Definition at line 160 of file files.c.

stringer_t* file_load (char * *name*)

files.c

files.c

Parameters:

name a character pointer to the full pathname of the file to be opened.

Returns:

NULL on failure, or a managed string containing all the data in the file.

Definition at line 56 of file files.c.

References log_info, st_alloc, st_avail_get(), st_data_get(), and st_length_set().

Referenced by adjust_message_encryption(), config_load_file_settings(), and ssl_start().

int_t file_read (char * *name*, stringer_t * *output*)

Get the contents of a file on disk.

See also:

file_load()

Note:

This is similar in function to **file_load()** but there is no read() length checking, so it can be used on non-regular files.

Parameters:

name a character pointer to the full pathname of the file to be opened.

output a managed string where the results of the file read operation will be stored.

Returns:

-1 on failure or the number of bytes read from the file on success.

Definition at line 22 of file files.c.

References log_info, log_pedantic, MEMORYBUF, st_avail_get(), st_data_get(), and st_length_set().

Referenced by process_kill().

bool_t file_readwritable (const chr_t * *path*)

Determine whether a given filename or directory path is readable and writable by a user.

Parameters:

path a null-terminated string with the name of the filename or directory to be checked for permissions.

Returns:

true if the pathname exists and is readable and writable, or false otherwise.

Definition at line 170 of file files.c.

Referenced by servers_validate().

bool_t file_world_accessible (const chr_t * *path*)

Determine whether a given filename or directory path is readable or writable by other users.

Parameters:

path a null-terminated string with the name of the filename or directory to be checked for world permissions.

Returns:

true if the pathname exists and is readable or writable by others, or false otherwise.

Definition at line 180 of file files.c.

Referenced by servers_validate(), and ssl_start().

int_t folder_exists (stringer_t * *path*, bool_t *create*)

folder.c

folder.c

Parameters:

path a managed string containing the full pathname of the directory.

create if true, attempt to create the directory if it does not already exist.

Returns:

-1 if the directory doesn't exist or couldn't be created, 0 if the directory exists, or 1 if the directory was created.

Definition at line 21 of file folder.c.

References st_char_get().

Referenced by log_start(), and spool_check().

int_t get_temp_file_handle (chr_t * *pdir*, stringer_t ** *tmpname*)

Get a file handle to a temporary file created in a specified directory.

Parameters:

pdir the parent directory in which to create the temporary file, or NULL for the default spool dir.

tmpname an optional pointer to a managed string to receive the name of the created temp file.

Returns:

-1 on failure or the new temporary file's file descriptor on success.

Definition at line 104 of file files.c.

References CONSTANT, HEAP, JOINTED, log_pedantic, MAGMA_SPOOL_BASE, MANAGED_T, ns_length_get(), spool_path(), st_append, st_char_get(), st_dupe(), st_dupe_opts(), st_free(), and st_import().

Referenced by adjust_message_encryption().

stringer_t* host_platform (stringer_t * *output*)

host.c

host.c

Parameters:

output a pointer to a managed string to receive the OS description.

Returns:

NULL on failure, or the user-specified managed string containing the OS info on success.

Definition at line 21 of file host.c.

stringer_t* host_version (stringer_t * *output*)

Get release information about the local OS.

Parameters:

output a pointer to a managed string to receive the release information.

Returns:

NULL on failure, or the user-specified managed string containing the release info on success.

Definition at line 47 of file host.c.

int_t process_kill (stringer_t * *name*, int_t *signal*, int_t *wait*)

process.c

process.c

Parameters:

name the name of the process to kill (matches any process that starts with the specified name).

signal the signal number to be sent to the matching process.

wait the number of times to re-send the signal, with one second rest intervals in between.

Returns:

-2 for a generic failure, -1 on timeout, 0 if process not found, and 1 if the process was successfully killed.

Definition at line 22 of file process.c.

References chr_numeric(), file_read(), int32_conv_ns(), log_pedantic, MAGMA_FILEPATH_MAX, MAGMA_PROC_PATH, MANAGEDBUF, MEMORYBUF, pid, st_char_get(), st_cmp_ci_starts(), st_length_int(), and st_swap().

chr_t* signal_name (int *signal*, char * *buffer*, size_t *length*)

Translate a numeric signal into a human readable name.

Parameters:

signal the input signal number.

buffer a buffer to receive the signal description.

length the length of the buffer to receive the output, which should be greater than 32 bytes.

Returns:

a pointer to a null-terminated string containing the textual name of the specified signal.

Definition at line 51 of file mappings.c.

References log_pedantic.

Referenced by signal_segfault(), signal_shutdown(), and signal_status().

int_t spool_check (stringer_t * *path*)

spool.c

spool.c

Parameters:

path a managed string with the spool directory to be checked.

Returns:

0 if the specified spool path exists, 1 if it was created, or -1 on error.

Definition at line 76 of file spool.c.

References `folder_exists()`, `log_critical`, `log_info`, `mutex_lock()`, `mutex_unlock()`, `st_char_get()`, and `st_length_int()`.

Referenced by `spool_mktemp()`, `spool_start()`, and `virus_start()`.

int_t spool_check_file (const char * *file*, const struct stat * *info*, int *type*)

An internal function used by the `ftw()` function to cleanup the file contents of a spool directory.

Parameters:

file a pointer to a null-terminated string containing the pathname of the spool file.

info a pointer to a stat object containing the filesystem info of the specified file.

the ftw type flag of the specified file (only FTW_F is handled).

Returns:

This function always returns 0.

Definition at line 186 of file spool.c.

References `buflen`, `bufptr`, `log_error`, `mutex_lock()`, `mutex_unlock()`, `NULLER`, `PLACER`, and `st_cmp_cs_eq()`.

Referenced by `spool_cleanup()`, and `spool_stop()`.

int_t spool_cleanup (void)

Clean up the file contents in the spool base directory.

Returns:

-1 on error or the number of files that were purged from the spool base directory.

Definition at line 214 of file spool.c.

References `log_error`, `log_pedantic`, `MAGMA_SPOOL_BASE`, `rwlock_lock_write()`, `rwlock_unlock()`, `spool_check_file()`, `spool_path()`, `st_char_get()`, `st_free()`, and `st_length_int()`.

Referenced by `spool_start()`.

uint64_t spool_error_stats (void)

Get the number of spool errors encountered.

Returns:

the total number of spool errors.

Definition at line 30 of file spool.c.

References `mutex_lock()`, and `mutex_unlock()`.

Referenced by `derived_value()`.

int_t spool_mktemp (int_t spool, chr_t * prefix)

Create a temporary file in a specified spool directory.

Note:

The temp file is automatically unlinked as soon as it is created.

Parameters:

spool the spool directory id in which the temp file will be stored (MAGMA_SPOOL_BASE, MAGMA_SPOOL_DATA, or MAGMA_SPOOL_SCAN).

prefix an optional prefix for the temp file name ("magma" will be used if prefix is NULL0.

Returns:

-1 on failure or the file descriptor to the newly created temp file on success.

Definition at line 114 of file `spool.c`.

References `log_critical`, `log_pedantic`, `MAGMA_SPOOL_BASE`, `MEMORYBUF`, `mutex_lock()`, `mutex_unlock()`, `rand_get_uint64()`, `rwlock_lock_read()`, `rwlock_unlock()`, `spool_check()`, `spool_path()`, `st_aprint()`, `st_char_get()`, `st_free()`, `st_length_int()`, and `thread_get_thread_id()`.

Referenced by `st_alloc_opts()`, and `virus_check()`.

stringer_t* spool_path (int_t spool)

Get the full path to a requested spool directory.

Parameters:

the spool id (MAGMA_SPOOL_BASE, MAGMA_SPOOL_DATA, or MAGMA_SPOOL_SCAN); defaults to MAGMA_SPOOL_DATA.

Returns:

NULL on failure, or a managed string pointing to the requested path inside the magma spool parent directory, or `/tmp/magma` if the spool isn't configured.

Definition at line 44 of file `spool.c`.

References `CONTIGUOUS`, `HEAP`, `magma`, `MAGMA_SPOOL_BASE`, `MAGMA_SPOOL_SCAN`, `ns_length_get()`, `NULLER_T`, `magma_t::spool`, and `st_merge_opts()`.

Referenced by `dspam_check()`, `dspam_train()`, `get_temp_file_handle()`, `spool_cleanup()`, `spool_mktemp()`, `spool_start()`, and `virus_start()`.

bool_t spool_start (void)

Create and check the spool directories, and clean them for use.

Returns:

true on success or false on failure.

Definition at line 278 of file `spool.c`.

References `log_critical`, `MAGMA_SPOOL_BASE`, `MAGMA_SPOOL_DATA`, `MAGMA_SPOOL_SCAN`, `mutex_lock()`, `mutex_unlock()`, `spool_check()`, `spool_cleanup()`, `spool_path()`, `st_char_get()`, `st_free()`, and `st_length_int()`.

Referenced by `process_start()`.

void spool_stop (void)

Clean up the file contents in the spool base directory.

Note:

This is like a fast version of `spool_cleanup()`.

Returns:

This function returns no value.

Definition at line 253 of file `spool.c`.

References `log_pedantic`, `spool_check_file()`, `st_char_get()`, `st_free()`, and `st_length_int()`.

Referenced by `process_stop()`.

magma/core/host/process.c File Reference

Functions for managing processes.

```
#include "magma.h"
```

Functions

- **int_t process_kill** (stringer_t *name, int_t signal, int_t wait)

Kill a named process with the specified signal, and retry if necessary.

Detailed Description

Functions for managing processes.

Definition in file **process.c**.

Function Documentation

int_t process_kill (stringer_t * *name*, int_t *signal*, int_t *wait*)

Kill a named process with the specified signal, and retry if necessary.

process.c

Parameters:

name the name of the process to kill (matches any process that starts with the specified name).

signal the signal number to be sent to the matching process.

wait the number of times to re-send the signal, with one second rest intervals in between.

Returns:

-2 for a generic failure, -1 on timeout, 0 if process not found, and 1 if the process was successfully killed.

Definition at line 22 of file process.c.

References chr_numeric(), file_read(), int32_conv_ns(), log_pedantic, MAGMA_FILEPATH_MAX, MAGMA_PROC_PATH, MANAGEDBUF, MEMORYBUF, pid, st_char_get(), st_cmp_ci_starts(), st_length_int(), and st_swap().

magma/engine/context/process.c File Reference

Functions used to start and stop the daemon, including the execution of the module init and module clean up functions.

```
#include "magma.h"
```

Functions

- void **process_maint** (void)
- *The entry point for the process maintenance thread, which runs in a continuous loop unless canceled.* void **process_stop** (void)
- *Execute the magma shutdown routines.* **bool_t process_start** (void)

Execute the magma initializations routines, making sure they all run properly.

Variables

- **bool_t exit_and_dump**
- **uint64_t day** = 0
- **pthread_t * maint** = NULL

Detailed Description

Functions used to start and stop the daemon, including the execution of the module init and module clean up functions.

Definition in file **process.c**.

Function Documentation

void process_maint (void)

The entry point for the process maintenance thread, which runs in a continuous loop unless canceled.

Note:

Execute once daily: rotate the log files, update the warehouse, and perform tank maintenance. Execute every few (0-10) minutes: refresh the virus engine and prune the object cache.

Returns:

This function returns no value.

Definition at line 25 of file process.c.

References `day`, `log_rotate()`, `obj_cache_prune()`, `rand_get_uint32()`, `status`, `thread_cancel_disable()`, `thread_cancel_enable()`, `thread_start()`, `thread_stop()`, `time_datestamp()`, `time_till_midnight()`, `virus_engine_refresh()`, and `warehouse_update()`.

Referenced by `process_start()`.

bool_t process_start (void)

Execute the magma initializations routines, making sure they all run properly.

Note:

If any of the init routines returns with failure, this function fails and logs an error message. Certain checks are made that the init routines are run in a particular order, especially in waiting for daemonization and privilege dropping before starting logging. The system maintenance thread is also launched from here.

Returns:

true if all starter functions returned true, or false if any of them failed..

Definition at line 166 of file process.c.

References cache_start(), config_load_cmdline_settings(), config_load_database_settings(), config_load_defaults(), config_load_file_settings(), config_validate_settings(), dkim_start(), dspam_start(), ecies_start(), exit_and_dump, magma_t::host, http_content_start(), magma_t::init, lib_load(), log_critical, log_info, log_start(), magma, MAGMA_HOSTNAME_MAX, mail_cache_start(), maint, mm_alloc(), mm_free(), mm_sec_start(), magma_t::name, obj_cache_start(), magma_t::page_length, process_maint(), protocol_init(), queue_init(), rand_start(), sanity_check(), servers_encryption_start(), servers_network_start(), signal_start(), spf_start(), spool_start(), sql_start(), ssl_start(), stats_init(), status_process(), system_change_root_directory(), system_fork_daemon(), system_init_core_dumps(), system_init_impersonation(), system_init_resource_limits(), system_init_umask(), thread_launch(), virus_start(), warehouse_start(), and xml_start().

Referenced by main().

void process_stop (void)

Execute the magma shutdown routines.

Note:

This function will execute a shutdown routine for each of the startup initialization routines that was performed. It will also signal and terminate the maintenance thread.

Returns:

This function returns no value.

Definition at line 71 of file process.c.

References cache_stop(), config_free(), dkim_stop(), dspam_stop(), ecies_stop(), http_content_stop(), magma_t::init, lib_unload(), log_critical, log_info, log_pedantic, magma, mail_cache_stop(), maint, mm_free(), mm_sec_stop(), obj_cache_stop(), queue_shutdown(), rand_stop(), servers_encryption_stop(), servers_network_stop(), spf_stop(), spool_stop(), sql_stop(), ssl_stop(), stats_shutdown(), status, thread_join(), thread_signal(), virus_stop(), warehouse_stop(), and xml_stop().

Referenced by main().

Variable Documentation

uint64_t day = 0

Definition at line 16 of file process.c.

Referenced by process_maint().

bool_t exit_and_dump

Definition at line 18 of file global.c.

Referenced by args_parse(), config_validate_settings(), and process_start().

pthread_t* maint = NULL

Definition at line 17 of file process.c.

Referenced by process_start(), and process_stop().

magma/core/host/spool.c File Reference

Functions for checking, creating, maintaining and using the spool.

```
#include "magma.h"
```

Functions

- `uint64_t spool_error_stats` (void)
- *Get the number of spool errors encountered.* `stringer_t * spool_path` (int_t spool)
- *Get the full path to a requested spool directory.* `int_t spool_check` (stringer_t *path)
- *Check to see if the specified spool directory exists.* `int_t spool_mktemp` (int_t spool, chr_t *prefix)
- *Create a temporary file in a specified spool directory.* `int_t spool_check_file` (const char *file, const struct stat *info, int type)
- *An internal function used by the ftw() function to cleanup the file contents of a spool directory.* `int_t spool_cleanup` (void)
- *Clean up the file contents in the spool base directory.* void `spool_stop` (void)
- *Clean up the file contents in the spool base directory.* `bool_t spool_start` (void)

Create and check the spool directories, and clean them for use.

Detailed Description

Functions for checking, creating, maintaining and using the spool.

Definition in file `spool.c`.

Function Documentation

`int_t spool_check` (stringer_t * path)

Check to see if the specified spool directory exists.

spool.c

Parameters:

path a managed string with the spool directory to be checked.

Returns:

0 if the specified spool path exists, 1 if it was created, or -1 on error.

Definition at line 76 of file `spool.c`.

References `folder_exists()`, `log_critical`, `log_info`, `mutex_lock()`, `mutex_unlock()`, `st_char_get()`, and `st_length_int()`.

Referenced by `spool_mktemp()`, `spool_start()`, and `virus_start()`.

`int_t spool_check_file` (const char * file, const struct stat * info, int type)

An internal function used by the `ftw()` function to cleanup the file contents of a spool directory.

Parameters:

file a pointer to a null-terminated string containing the pathname of the spool file.
info a pointer to a stat object containing the filesystem info of the specified file.
the ftw type flag of the specified file (only FTW_F is handled).

Returns:

This function always returns 0.

Definition at line 186 of file spool.c.

References `buflen`, `bufptr`, `log_error`, `mutex_lock()`, `mutex_unlock()`, `NULLER`, `PLACER`, and `st_cmp_cs_eq()`.

Referenced by `spool_cleanup()`, and `spool_stop()`.

int_t spool_cleanup (void)

Clean up the file contents in the spool base directory.

Returns:

-1 on error or the number of files that were purged from the spool base directory.

Definition at line 214 of file spool.c.

References `log_error`, `log_pedantic`, `MAGMA_SPOOL_BASE`, `rwlock_lock_write()`, `rwlock_unlock()`, `spool_check_file()`, `spool_path()`, `st_char_get()`, `st_free()`, and `st_length_int()`.

Referenced by `spool_start()`.

uint64_t spool_error_stats (void)

Get the number of spool errors encountered.

Returns:

the total number of spool errors.

Definition at line 30 of file spool.c.

References `mutex_lock()`, and `mutex_unlock()`.

Referenced by `derived_value()`.

int_t spool_mktemp (int_t spool, chr_t * prefix)

Create a temporary file in a specified spool directory.

Note:

The temp file is automatically unlinked as soon as it is created.

Parameters:

spool the spool directory id in which the temp file will be stored (`MAGMA_SPOOL_BASE`, `MAGMA_SPOOL_DATA`, or `MAGMA_SPOOL_SCAN`).
prefix an optional prefix for the temp file name ("magma" will be used if prefix is NULL0).

Returns:

-1 on failure or the file descriptor to the newly created temp file on success.

Definition at line 114 of file spool.c.

References `log_critical`, `log_pedantic`, `MAGMA_SPOOL_BASE`, `MEMORYBUF`, `mutex_lock()`, `mutex_unlock()`, `rand_get_uint64()`, `rwlock_lock_read()`, `rwlock_unlock()`, `spool_check()`, `spool_path()`, `st_aprint()`, `st_char_get()`, `st_free()`, `st_length_int()`, and `thread_get_thread_id()`.

Referenced by `st_alloc_opts()`, and `virus_check()`.

stringer_t* spool_path (int_t spool)

Get the full path to a requested spool directory.

Parameters:

the spool id (`MAGMA_SPOOL_BASE`, `MAGMA_SPOOL_DATA`, or `MAGMA_SPOOL_SCAN`);
defaults to `MAGMA_SPOOL_DATA`.

Returns:

NULL on failure, or a managed string pointing to the requested path inside the magma spool parent directory, or `/tmp/magma` if the spool isn't configured.

Definition at line 44 of file `spool.c`.

References `CONTIGUOUS`, `HEAP`, `magma`, `MAGMA_SPOOL_BASE`, `MAGMA_SPOOL_SCAN`, `ns_length_get()`, `NULLER_T`, `magma_t::spool`, and `st_merge_opts()`.

Referenced by `dspam_check()`, `dspam_train()`, `get_temp_file_handle()`, `spool_cleanup()`, `spool_mktemp()`, `spool_start()`, and `virus_start()`.

bool_t spool_start (void)

Create and check the spool directories, and clean them for use.

Returns:

true on success or false on failure.

Definition at line 278 of file `spool.c`.

References `log_critical`, `MAGMA_SPOOL_BASE`, `MAGMA_SPOOL_DATA`, `MAGMA_SPOOL_SCAN`, `mutex_lock()`, `mutex_unlock()`, `spool_check()`, `spool_cleanup()`, `spool_path()`, `st_char_get()`, `st_free()`, and `st_length_int()`.

Referenced by `process_start()`.

void spool_stop (void)

Clean up the file contents in the spool base directory.

Note:

This is like a fast version of `spool_cleanup()`.

Returns:

This function returns no value.

Definition at line 253 of file `spool.c`.

References `log_pedantic`, `spool_check_file()`, `st_char_get()`, `st_free()`, and `st_length_int()`.

Referenced by `process_stop()`.

magma/core/indexes/cursors.c File Reference

The generic index interface for handling cursors.

```
#include "magma.h"
```

Functions

- void **inx_cursor_reset** (inx_cursor_t *cursor)
- *Reset the position of an inx cursor.* void **inx_cursor_free** (inx_cursor_t *cursor)
- *Free an inx cursor and the inx object it points to, if the inx reference count hits zero.* inx_cursor_t * **inx_cursor_alloc** (inx_t *index)
- *Create a new cursor to iterate through an inx object.* multi_t **inx_cursor_key_next** (inx_cursor_t *cursor)
- *Get the key at the next inx cursor position.* multi_t **inx_cursor_key_active** (inx_cursor_t *cursor)
- *Get the key at the current inx cursor position.* void * **inx_cursor_value_next** (inx_cursor_t *cursor)
- *Get the key at the next inx cursor position.* void * **inx_cursor_value_active** (inx_cursor_t *cursor)

Get the value at the current inx cursor position.

Detailed Description

The generic index interface for handling cursors.

Definition in file **cursors.c**.

Function Documentation

inx_cursor_t* inx_cursor_alloc (inx_t * index)

Create a new cursor to iterate through an inx object.

cursors.c

Parameters:

index a pointer to the inx object to be traversed.

Returns:

NULL on failure, or a pointer to a new inx cursor for the specified inx object on success.

Definition at line 57 of file cursors.c.

References inx_t::cursor_alloc, inx_auto_unlock(), inx_auto_write(), and inx_t::references.

Referenced by config_load_cmdline_settings(), config_load_file_settings(), contact_find_detail(), contact_find_name(), contacts_fetch(), contacts_update(), decrypt_user_messages(), encrypt_user_messages(), http_data_get(), imap_append(), imap_close(), imap_copy(), imap_duplicate_messages(), imap_examine(), imap_expunge(), imap_fetch(), imap_folder_remove(), imap_folder_status(), imap_list(), imap_lsub(), imap_narrow_folders(), imap_narrow_messages(), imap_next_folder_order(), imap_search(), imap_search_messages(), imap_select(), imap_session_destroy(), imap_session_update(), imap_store(), imap_update_flags(), magma_folder_children(), magma_folder_find_full_name(), magma_folder_find_name(), messages_update(), meta_check_message_encryption(), meta_data_fetch_messages(), meta_data_flags_add(), meta_data_flags_remove(), meta_data_flags_replace(), meta_folders_by_name(), meta_folders_by_number(), meta_folders_children(), meta_folders_stats_tags(), meta_message_by_number(), meta_messages_update_sequences(), obj_cache_prune(), pattern_check(), pop_get_last(), pop_get_message(), pop_list(), pop_session_destroy(), pop_session_reset(), pop_total_messages(), pop_total_size(), pop_uidl(),

portal_config_collection(), portal_contact_details(), portal_endpoint_alert_list(), portal_endpoint_aliases(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_list(), portal_endpoint_folders_list(), portal_endpoint_folders_tags(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(), portal_endpoint_messages_tag(), portal_endpoint_messages_tags(), portal_smtp_create_data(), portal_smtp_merge_headers(), portal_smtp_relay_message(), portal_upload(), register_abuse_check_blocklist(), smtp_bypass_check(), smtp_check_filters(), and teacher_print_message().

void inx_cursor_free (inx_cursor_t * cursor)

Free an inx cursor and the inx object it points to, if the inx reference count hits zero.

Parameters:

cursor the inx cursor to be freed.

Returns:

This function returns no value.

Definition at line 34 of file cursors.c.

References inx_auto_unlock(), inx_auto_write(), and inx_free().

Referenced by config_load_cmdline_settings(), config_load_file_settings(), contact_find_detail(), contact_find_name(), contacts_fetch(), contacts_update(), decrypt_user_messages(), encrypt_user_messages(), http_data_get(), imap_append(), imap_close(), imap_copy(), imap_duplicate_messages(), imap_examine(), imap_expunge(), imap_fetch(), imap_folder_remove(), imap_folder_status(), imap_list(), imap_lsub(), imap_narrow_folders(), imap_narrow_messages(), imap_next_folder_order(), imap_search_messages(), imap_select(), imap_session_destroy(), imap_session_update(), imap_store(), imap_update_flags(), magma_folder_children(), magma_folder_find_full_name(), magma_folder_find_name(), messages_update(), meta_check_message_encryption(), meta_data_fetch_messages(), meta_data_flags_add(), meta_data_flags_remove(), meta_data_flags_replace(), meta_folders_by_name(), meta_folders_by_number(), meta_folders_children(), meta_folders_stats_tags(), meta_message_by_number(), meta_messages_update_sequences(), obj_cache_prune(), pattern_check(), pop_get_last(), pop_get_message(), pop_list(), pop_session_destroy(), pop_session_reset(), pop_total_messages(), pop_total_size(), pop_uidl(), portal_config_collection(), portal_contact_details(), portal_endpoint_alert_list(), portal_endpoint_aliases(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_list(), portal_endpoint_folders_list(), portal_endpoint_folders_tags(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(), portal_endpoint_messages_tag(), portal_endpoint_messages_tags(), portal_smtp_create_data(), portal_smtp_merge_headers(), portal_smtp_relay_message(), portal_upload(), register_abuse_check_blocklist(), smtp_bypass_check(), smtp_check_filters(), and teacher_print_message().

multi_t inx_cursor_key_active (inx_cursor_t * cursor)

Get the key at the current inx cursor position.

Parameters:

cursor the inx cursor to be examined.

Returns:

NULL on failure, or a multi-type data object containing the key of the specified inx cursor.

Definition at line 97 of file cursors.c.

References inx_auto_read(), inx_auto_unlock(), and mt_get_null().

Referenced by imap_search_messages(), and obj_cache_prune().

multi_t inx_cursor_key_next (inx_cursor_t * *cursor*)

Get the key at the next inx cursor position.

Parameters:

cursor the inx cursor to be examined.

Returns:

NULL on failure, or a multi-type data object containing the key of the next inx cursor position.

Definition at line 79 of file cursors.c.

References inx_auto_read(), inx_auto_unlock(), and mt_get_null().

Referenced by config_load_cmdline_settings(), and config_load_file_settings().

void inx_cursor_reset (inx_cursor_t * *cursor*)

Reset the position of an inx cursor.

Parameters:

cursor a pointer to the inx cursor to be reset.

Returns:

This function returns no value.

Definition at line 20 of file cursors.c.

Referenced by imap_fetch(), imap_list(), imap_lsub(), obj_cache_prune(), and portal_smtp_create_data().

void* inx_cursor_value_active (inx_cursor_t * *cursor*)

Get the value at the current inx cursor position.

Parameters:

cursor the inx cursor to be examined.

Returns:

NULL on failure, or the value at the current position of the specified inx cursor.

Definition at line 133 of file cursors.c.

References inx_auto_read(), and inx_auto_unlock().

Referenced by config_load_cmdline_settings(), and config_load_file_settings().

void* inx_cursor_value_next (inx_cursor_t * *cursor*)

Get the key at the next inx cursor position.

Parameters:

cursor the inx cursor to be examined.

Returns:

NULL on failure, or a multi-type data object containing the key of the next inx cursor position.

Definition at line 115 of file cursors.c.

References `inx_auto_read()`, and `inx_auto_unlock()`.

Referenced by `contact_find_detail()`, `contact_find_name()`, `contacts_fetch()`, `contacts_update()`, `decrypt_user_messages()`, `encrypt_user_messages()`, `http_data_get()`, `imap_append()`, `imap_close()`, `imap_copy()`, `imap_duplicate_messages()`, `imap_examine()`, `imap_expunge()`, `imap_fetch()`, `imap_folder_remove()`, `imap_folder_status()`, `imap_list()`, `imap_lsub()`, `imap_narrow_folders()`, `imap_narrow_messages()`, `imap_next_folder_order()`, `imap_search()`, `imap_search_messages()`, `imap_select()`, `imap_session_destroy()`, `imap_session_update()`, `imap_store()`, `imap_update_flags()`, `magma_folder_children()`, `magma_folder_find_full_name()`, `magma_folder_find_name()`, `messages_update()`, `meta_check_message_encryption()`, `meta_data_fetch_messages()`, `meta_data_flags_add()`, `meta_data_flags_remove()`, `meta_data_flags_replace()`, `meta_folders_by_name()`, `meta_folders_by_number()`, `meta_folders_children()`, `meta_folders_stats_tags()`, `meta_message_by_number()`, `meta_messages_update_sequences()`, `obj_cache_prune()`, `pattern_check()`, `pop_get_last()`, `pop_get_message()`, `pop_list()`, `pop_session_destroy()`, `pop_session_reset()`, `pop_total_messages()`, `pop_total_size()`, `pop_uidl()`, `portal_config_collection()`, `portal_contact_details()`, `portal_endpoint_alert_list()`, `portal_endpoint_aliases()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_contacts_list()`, `portal_endpoint_folders_list()`, `portal_endpoint_folders_tags()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_tag()`, `portal_endpoint_messages_tags()`, `portal_smtp_create_data()`, `portal_smtp_merge_headers()`, `portal_smtp_relay_message()`, `portal_upload()`, `register_abuse_check_blocklist()`, `smtp_bypass_check()`, `smtp_check_filters()`, and `teacher_print_message()`.

magma/core/indexes/hashed.c File Reference

Function declarations and types for the hashed list.

```
#include "magma.h"
```

Data Structures

- struct **hashed_bucket_t**
- struct **__attribute__**
- struct **__attribute__**

Defines

- #define **MAGMA_HASHED_BUCKETS** 1024

Functions

- uint32_t **hashed_bucket** (uint32_t buckets, **multi_t** key)
- *Get the hash bucket for a key.* **hashed_bucket_t * hashed_bucket_alloc** (**multi_t** key, void ***data**)
- *Allocate and initialize hashed bucket object.* **hashed_bucket_t * hashed_bucket_get_ptr** (hashed_index_t *hashed, uint32_t num)
- void **hashed_bucket_set_ptr** (hashed_index_t *hashed, uint32_t num, **hashed_bucket_t** *bucket)
- void **hashed_bucket_add** (**hashed_bucket_t** *bucket, **hashed_bucket_t** *new)
- **bool_t** **hashed_insert** (void *inx, **multi_t** key, void ***data**)
- void * **hashed_bucket_find_key** (**hashed_bucket_t** *bucket, **multi_t** key)
- void * **hashed_find** (void *inx, **multi_t** key)
- **bool_t** **hashed_delete** (void *inx, **multi_t** key)
- **hashed_bucket_t * hashed_cursor_active** (hashed_cursor_t *cursor)
- **hashed_bucket_t * hashed_cursor_next** (hashed_cursor_t *cursor)
- void * **hashed_cursor_value_next** (hashed_cursor_t *cursor)
- void * **hashed_cursor_value_active** (hashed_cursor_t *cursor)
- **multi_t** **hashed_cursor_key_next** (hashed_cursor_t *cursor)
- **multi_t** **hashed_cursor_key_active** (hashed_cursor_t *cursor)
- void **hashed_cursor_reset** (hashed_cursor_t *cursor)
- void **hashed_cursor_free** (hashed_cursor_t *cursor)
- void * **hashed_cursor_alloc** (**inx_t** *inx)
- void **hashed_free** (void *inx)
- void **hashed_truncate** (void *inx)
- **inx_t * hashed_alloc** (uint64_t options, void *data_free)

Allocate a new hash table.

Detailed Description

Function declarations and types for the hashed list.

Definition in file **hashed.c**.

Define Documentation

#define MAGMA_HASHED_BUCKETS 1024

Definition at line 15 of file hashed.c.

Referenced by hashed_alloc().

Function Documentation

inx_t* hashed_alloc (uint64_t *options*, void * *data_free*)

Allocate a new hash table.

hashed.c

Parameters:

options an options value for the hash table.

data_free a pointer to a function

Returns:

NULL on failure, or a pointer to the newly allocated hash table object on success.

Definition at line 498 of file hashed.c.

References inx_t::cursor_alloc, inx_t::cursor_free, inx_t::cursor_key_active, inx_t::cursor_key_next, inx_t::cursor_reset, inx_t::cursor_value_active, inx_t::cursor_value_next, inx_t::data_free, inx_t::delete, inx_t::find, hashed_cursor_alloc(), hashed_cursor_free(), hashed_cursor_key_active(), hashed_cursor_key_next(), hashed_cursor_reset(), hashed_cursor_value_active(), hashed_cursor_value_next(), hashed_delete(), hashed_find(), hashed_free(), hashed_insert(), hashed_truncate(), inx_t::index, inx_t::index_free, inx_t::index_truncate, inx_t::insert, MAGMA_HASHED_BUCKETS, mm_alloc(), mm_free(), and inx_t::options.

Referenced by inx_alloc().

uint32_t hashed_bucket (uint32_t *buckets*, multi_t *key*)

Get the hash bucket for a key.

Note:

If the key is passed as a string, it will be mapped to a bucket using the Fletcher32 hash.

Parameters:

buckets the total number of buckets for grouping items.

key a multi-type key with the value to be looked up; numbers and strings are supported.

Returns:

the number of the hash bucket corresponding to the specified key.

Definition at line 45 of file hashed.c.

References bits_count(), count, hash_fletcher32(), log_check, mt_get_char(), mt_get_length(), mt_get_number(), and mt_is_number().

Referenced by hashed_delete(), hashed_find(), and hashed_insert().

void hashed_bucket_add (hashed_bucket_t * *bucket*, hashed_bucket_t * *new*)

Definition at line 124 of file hashed.c.

References hashed_bucket_t::next.

Referenced by hashed_insert().

hashed_bucket_t* hashed_bucket_alloc (multi_t *key*, void * *data*)

Allocate and initialize hashed bucket object.

Parameters:

key a multi-type key that will be hashed on lookup.

data a pointer to a data buffer associated with the key.

Returns:

NULL on failure, or a pointer to the newly allocated hashed bucket object on success.

Definition at line 75 of file hashed.c.

References hashed_bucket_t::data, hashed_bucket_t::key, mm_alloc(), and mt_dupe().

Referenced by hashed_insert().

void* hashed_bucket_find_key (hashed_bucket_t * *bucket*, multi_t *key*)

Definition at line 178 of file hashed.c.

References hashed_bucket_t::data, data, ident_mt_mt(), hashed_bucket_t::key, and hashed_bucket_t::next.

Referenced by hashed_find().

hashed_bucket_t* hashed_bucket_get_ptr (hashed_index_t * *hashed*, uint32_t *num*)

Definition at line 90 of file hashed.c.

References log_pedantic.

Referenced by hashed_cursor_active(), hashed_delete(), hashed_find(), hashed_free(), hashed_insert(), and hashed_truncate().

void hashed_bucket_set_ptr (hashed_index_t * *hashed*, uint32_t *num*, hashed_bucket_t * *bucket*)

Definition at line 108 of file hashed.c.

References log_pedantic.

Referenced by hashed_delete(), hashed_insert(), and hashed_truncate().

hashed_bucket_t* hashed_cursor_active (hashed_cursor_t * *cursor*)

BUG: It's a little unclear what's happening, but when this function gets called, the serials don't match, and the count is greater than the number of available data buckets, because a bucket has been removed, the cursor will get stuck returning the last item endlessly (I think).

Definition at line 279 of file hashed.c.

References count, hashed_bucket_get_ptr(), and hashed_bucket_t::next.

Referenced by hashed_cursor_key_active(), hashed_cursor_next(), and hashed_cursor_value_active().

void* hashed_cursor_alloc (inx_t * inx)

Definition at line 411 of file hashed.c.

References log_pedantic, and mm_alloc().

Referenced by hashed_alloc().

void hashed_cursor_free (hashed_cursor_t * cursor)

Definition at line 402 of file hashed.c.

References mm_free().

Referenced by hashed_alloc().

multi_t hashed_cursor_key_active (hashed_cursor_t * cursor)

Definition at line 382 of file hashed.c.

References hashed_cursor_active(), hashed_bucket_t::key, and mt_get_null().

Referenced by hashed_alloc().

multi_t hashed_cursor_key_next (hashed_cursor_t * cursor)

Definition at line 372 of file hashed.c.

References hashed_cursor_next(), hashed_bucket_t::key, and mt_get_null().

Referenced by hashed_alloc().

hashed_bucket_t* hashed_cursor_next (hashed_cursor_t * cursor)

Definition at line 327 of file hashed.c.

References hashed_cursor_active().

Referenced by hashed_cursor_key_next(), and hashed_cursor_value_next().

void hashed_cursor_reset (hashed_cursor_t * cursor)

Definition at line 392 of file hashed.c.

Referenced by hashed_alloc().

void* hashed_cursor_value_active (hashed_cursor_t * cursor)

Definition at line 362 of file hashed.c.

References hashed_bucket_t::data, and hashed_cursor_active().

Referenced by hashed_alloc().

void* hashed_cursor_value_next (hashed_cursor_t * cursor)

Definition at line 351 of file hashed.c.

References hashed_bucket_t::data, and hashed_cursor_next().

Referenced by hashed_alloc().

bool_t hashed_delete (void * inx, multi_t key)

Definition at line 221 of file hashed.c.

References inx_t::count, hashed_bucket_t::data, inx_t::data_free, hashed_bucket(), hashed_bucket_get_ptr(), hashed_bucket_set_ptr(), ident_mt_mt(), inx_t::index, hashed_bucket_t::key, mm_free(), mt_free(), hashed_bucket_t::next, and inx_t::serial.

Referenced by hashed_alloc().

void* hashed_find (void * inx, multi_t key)

Definition at line 193 of file hashed.c.

References data, hashed_bucket(), hashed_bucket_find_key(), hashed_bucket_get_ptr(), and inx_t::index.

Referenced by hashed_alloc().

void hashed_free (void * inx)

Definition at line 425 of file hashed.c.

References hashed_bucket_t::data, inx_t::data_free, hashed_bucket_get_ptr(), inx_t::index, hashed_bucket_t::key, mm_free(), mt_free(), and hashed_bucket_t::next.

Referenced by hashed_alloc().

bool_t hashed_insert (void * inx, multi_t key, void * data)

Definition at line 141 of file hashed.c.

References inx_t::count, hashed_bucket(), hashed_bucket_add(), hashed_bucket_alloc(), hashed_bucket_get_ptr(), hashed_bucket_set_ptr(), inx_t::index, and inx_t::serial.

Referenced by hashed_alloc().

void hashed_truncate (void * inx)

Definition at line 458 of file hashed.c.

References hashed_bucket_t::data, inx_t::data_free, hashed_bucket_get_ptr(), hashed_bucket_set_ptr(), inx_t::index, hashed_bucket_t::key, mm_free(), mt_free(), and hashed_bucket_t::next.

Referenced by hashed_alloc().

magma/core/indexes/indexes.h File Reference

Function declarations and types for the generic index interface.

Data Structures

- struct **inx_t**
- struct **__attribute__**

Defines

- #define **MAGMA_INDEX_TYPE** (M_INX_TREE | M_INX_LINKED | M_INX_HASHED)
- #define **MAGMA_INDEX_OPTION** (M_INX_INDEX_LOCK)

Enumerations

- enum **MAGMA_INDEX** { **M_INX_TREE** = 1, **M_INX_HASHED** = 2, **M_INX_LINKED** = 4, **M_INX_LOCK_MANUAL** = 16 }

Functions

- **inx_cursor_t** * **inx_cursor_alloc** (**inx_t** *index)
- *cursor.c* void **inx_cursor_free** (**inx_cursor_t** *cursor)
- *Free an inx cursor and the inx object it points to, if the inx reference count hits zero.* **multi_t** **inx_cursor_key_active** (**inx_cursor_t** *cursor)
- *Get the key at the current inx cursor position.* **multi_t** **inx_cursor_key_next** (**inx_cursor_t** *cursor)
- *Get the key at the next inx cursor position.* void **inx_cursor_reset** (**inx_cursor_t** *cursor)
- *Reset the position of an inx cursor.* void * **inx_cursor_value_active** (**inx_cursor_t** *cursor)
- *Get the value at the current inx cursor position.* void * **inx_cursor_value_next** (**inx_cursor_t** *cursor)
- *Get the key at the next inx cursor position.* **inx_t** * **inx_alloc** (uint64_t options, void *data_free)
- *inx.c* void **inx_auto_read** (**inx_t** *inx)
- void **inx_auto_unlock** (**inx_t** *inx)
- void **inx_auto_write** (**inx_t** *inx)
- void **inx_cleanup** (**inx_t** *inx)
- *Perform a checked free of an inx object.* uint64_t **inx_count** (**inx_t** *inx)
- *Return the total number of items held by an inx object.* **bool_t** **inx_delete** (**inx_t** *inx, **multi_t** key)
- *Delete a child of an inx object with a specified key.* void * **inx_find** (**inx_t** *inx, **multi_t** key)
- *Find the value associated with a particular key within the children of an inx object.* void **inx_free** (**inx_t** *inx)
- **bool_t** **inx_insert** (**inx_t** *inx, **multi_t** key, void *data)
- *Insert a new record into an inx holder.* void **inx_lock_read** (**inx_t** *inx)
- *Acquire a reader's lock for an inx object.* void **inx_lock_write** (**inx_t** *inx)
- *Acquire a writer's lock for an inx object.* uint64_t **inx_options** (**inx_t** *inx)
- *Return the options value of an inx object.* **bool_t** **inx_replace** (**inx_t** *inx, **multi_t** key, void *data)
- *Replace the value of a key in an inx holder.* uint64_t **inx_serial** (**inx_t** *inx)
- void **inx_truncate** (**inx_t** *inx)
- void **inx_unlock** (**inx_t** *inx)
- *Unlock an inx object.* **inx_t** * **linked_alloc** (uint64_t options, void *data_free)
- *linked.c* **inx_t** * **hashed_alloc** (uint64_t options, void *data_free)

hashed.c

Detailed Description

Function declarations and types for the generic index interface.

Definition in file **indexes.h**.

Define Documentation

#define MAGMA_INDEX_OPTION (M_INX_INDEX_LOCK)

The different index options.

Definition at line 37 of file indexes.h.

#define MAGMA_INDEX_TYPE (M_INX_TREE | M_INX_LINKED | M_INX_HASHED)

The different types of indexes.

Definition at line 32 of file indexes.h.

Referenced by `inx_alloc()`.

Enumeration Type Documentation

enum MAGMA_INDEX

Index types and options.

Enumerator:

M_INX_TREE M_INX_BTREE.

M_INX_HASHED M_INX_HASHED.

M_INX_LINKED M_INX_LINKED.

M_INX_LOCK_MANUAL M_INX_LOCK_MANUAL.

Definition at line 20 of file indexes.h.

Function Documentation

inx_t* hashed_alloc (uint64_t options, void * data_free)

hashed.c

hashed.c

Parameters:

options an options value for the hash table.

data_free a pointer to a function

Returns:

NULL on failure, or a pointer to the newly allocated hash table object on success.

Definition at line 498 of file hashed.c.

References `inx_t::cursor_alloc`, `inx_t::cursor_free`, `inx_t::cursor_key_active`, `inx_t::cursor_key_next`, `inx_t::cursor_reset`, `inx_t::cursor_value_active`, `inx_t::cursor_value_next`, `inx_t::data_free`, `inx_t::delete`, `inx_t::find`, `hashed_cursor_alloc()`, `hashed_cursor_free()`, `hashed_cursor_key_active()`, `hashed_cursor_key_next()`, `hashed_cursor_reset()`, `hashed_cursor_value_active()`, `hashed_cursor_value_next()`,

hashed_delete(), hashed_find(), hashed_free(), hashed_insert(), hashed_truncate(), inx_t::index, inx_t::index_free, inx_t::index_truncate, inx_t::insert, MAGMA_HASHED_BUCKETS, mm_alloc(), mm_free(), and inx_t::options.

Referenced by inx_alloc().

inx_t* inx_alloc (uint64_t options, void * data_free)

inx.c

inx.c

Parameters:

options a value indicating the inx type. Can be M_INX_TREE for a binary tree, M_INX_LINKED for a linked list, or M_INX_HASHED for a hash tree.

data_free a function pointer to a routine to free the data associated with an inx record.

Returns:

NULL on failure or a pointer to the newly created inx object on success.

Definition at line 298 of file inx.c.

References inx_t::automatic, hashed_alloc(), linked_alloc(), inx_t::lock, log_options, M_INX_HASHED, M_INX_LINKED, M_INX_LOCK_MANUAL, M_INX_TREE, M_LOG_ERROR, M_LOG_STACK_TRACE, MAGMA_INDEX_TYPE, inx_t::references, rwlock_init(), and tree_alloc().

Referenced by adjust_message_encryption(), contact_alloc(), contact_folder_alloc(), http_content_refresh(), http_content_start(), http_parse_header(), http_parse_pairs(), imap_duplicate_messages(), imap_narrow_folders(), imap_narrow_messages(), imap_search_messages(), magma_folder_fetch(), message_folder_alloc(), meta_data_fetch_alerts(), meta_data_fetch_all_tags(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_messages(), meta_folders_stats_tags(), nvp_alloc(), obj_cache_start(), portal_endpoint_messages_compose(), portal_endpoint_messages_flag(), portal_endpoint_messages_tag(), portal_parse_json_str_array(), register_data_fetch_blocklist(), sess_create(), smtp_add_bypass_entry(), smtp_fetch_inbound(), teacher_add_cookie(), user_config_alloc(), warehouse_fetch_domains(), and warehouse_fetch_patterns().

void inx_auto_read (inx_t * inx)

Definition at line 47 of file inx.c.

References inx_t::automatic, inx_t::lock, and rwlock_lock_read().

Referenced by inx_count(), inx_cursor_key_active(), inx_cursor_key_next(), inx_cursor_value_active(), inx_cursor_value_next(), inx_find(), inx_options(), and inx_serial().

void inx_auto_unlock (inx_t * inx)

Definition at line 28 of file inx.c.

References inx_t::automatic, inx_t::lock, and rwlock_unlock().

Referenced by inx_count(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_key_active(), inx_cursor_key_next(), inx_cursor_value_active(), inx_cursor_value_next(), inx_delete(), inx_find(), inx_free(), inx_insert(), inx_options(), inx_replace(), inx_serial(), and inx_truncate().

void inx_auto_write (inx_t * inx)

Definition at line 66 of file inx.c.

References `inx_t::automatic`, `inx_t::lock`, and `rwlock_lock_write()`.

Referenced by `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_delete()`, `inx_free()`, `inx_insert()`, `inx_replace()`, and `inx_truncate()`.

`void inx_cleanup (inx_t * inx)`

Perform a checked free of an `inx` object.

See also:

`inx_free()`

Parameters:

inx the `inx` instance to be freed.

Definition at line 285 of file `inx.c`.

References `inx_free()`.

Referenced by `contact_free()`, `contacts_update()`, `domain_stop()`, `http_content_stop()`, `http_session_reset()`, `imap_narrow_messages()`, `magma_folder_free()`, `messages_update()`, `meta_data_fetch_folders()`, `meta_data_fetch_mailbox_aliases()`, `meta_data_fetch_messages()`, `meta_user_destroy()`, `pattern_stop()`, `pattern_update()`, `sess_destroy()`, `sess_release_composition()`, `smtp_free_inbound()`, and `user_config_free()`.

`uint64_t inx_count (inx_t * inx)`

Return the total number of items held by an `inx` object.

Parameters:

inx a pointer to the `inx` object to be examined.

Returns:

the total number of items held by the `inx` object.

Definition at line 101 of file `inx.c`.

References `inx_t::count`, `count`, `inx_auto_read()`, `inx_auto_unlock()`, and `log_pedantic`.

Referenced by `contact_folder_remove()`, `imap_copy()`, `imap_list()`, `imap_login()`, `imap_narrow_messages()`, `message_folder_remove()`, `meta_user_prune()`, `obj_cache_prune()`, and `pop_pass()`.

`inx_cursor_t* inx_cursor_alloc (inx_t * index)`

`cursors.c`

`cursors.c`

Parameters:

index a pointer to the `inx` object to be traversed.

Returns:

NULL on failure, or a pointer to a new `inx` cursor for the specified `inx` object on success.

Definition at line 57 of file `cursors.c`.

References `inx_t::cursor_alloc`, `inx_auto_unlock()`, `inx_auto_write()`, and `inx_t::references`.

Referenced by `config_load_cmdline_settings()`, `config_load_file_settings()`, `contact_find_detail()`, `contact_find_name()`, `contacts_fetch()`, `contacts_update()`, `decrypt_user_messages()`, `encrypt_user_messages()`,

http_data_get(), imap_append(), imap_close(), imap_copy(), imap_duplicate_messages(), imap_examine(), imap_expunge(), imap_fetch(), imap_folder_remove(), imap_folder_status(), imap_list(), imap_lsub(), imap_narrow_folders(), imap_narrow_messages(), imap_next_folder_order(), imap_search(), imap_search_messages(), imap_select(), imap_session_destroy(), imap_session_update(), imap_store(), imap_update_flags(), magma_folder_children(), magma_folder_find_full_name(), magma_folder_find_name(), messages_update(), meta_check_message_encryption(), meta_data_fetch_messages(), meta_data_flags_add(), meta_data_flags_remove(), meta_data_flags_replace(), meta_folders_by_name(), meta_folders_by_number(), meta_folders_children(), meta_folders_stats_tags(), meta_message_by_number(), meta_messages_update_sequences(), obj_cache_prune(), pattern_check(), pop_get_last(), pop_get_message(), pop_list(), pop_session_destroy(), pop_session_reset(), pop_total_messages(), pop_total_size(), pop_uidl(), portal_config_collection(), portal_contact_details(), portal_endpoint_alert_list(), portal_endpoint_aliases(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_list(), portal_endpoint_folders_list(), portal_endpoint_folders_tags(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(), portal_endpoint_messages_tag(), portal_endpoint_messages_tags(), portal_smtp_create_data(), portal_smtp_merge_headers(), portal_smtp_relay_message(), portal_upload(), register_abuse_check_blocklist(), smtp_bypass_check(), smtp_check_filters(), and teacher_print_message().

void inx_cursor_free (inx_cursor_t * cursor)

Free an inx cursor and the inx object it points to, if the inx reference count hits zero.

Parameters:

cursor the inx cursor to be freed.

Returns:

This function returns no value.

Definition at line 34 of file cursors.c.

References inx_auto_unlock(), inx_auto_write(), and inx_free().

Referenced by config_load_cmdline_settings(), config_load_file_settings(), contact_find_detail(), contact_find_name(), contacts_fetch(), contacts_update(), decrypt_user_messages(), encrypt_user_messages(), http_data_get(), imap_append(), imap_close(), imap_copy(), imap_duplicate_messages(), imap_examine(), imap_expunge(), imap_fetch(), imap_folder_remove(), imap_folder_status(), imap_list(), imap_lsub(), imap_narrow_folders(), imap_narrow_messages(), imap_next_folder_order(), imap_search_messages(), imap_select(), imap_session_destroy(), imap_session_update(), imap_store(), imap_update_flags(), magma_folder_children(), magma_folder_find_full_name(), magma_folder_find_name(), messages_update(), meta_check_message_encryption(), meta_data_fetch_messages(), meta_data_flags_add(), meta_data_flags_remove(), meta_data_flags_replace(), meta_folders_by_name(), meta_folders_by_number(), meta_folders_children(), meta_folders_stats_tags(), meta_message_by_number(), meta_messages_update_sequences(), obj_cache_prune(), pattern_check(), pop_get_last(), pop_get_message(), pop_list(), pop_session_destroy(), pop_session_reset(), pop_total_messages(), pop_total_size(), pop_uidl(), portal_config_collection(), portal_contact_details(), portal_endpoint_alert_list(), portal_endpoint_aliases(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_list(), portal_endpoint_folders_list(), portal_endpoint_folders_tags(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(), portal_endpoint_messages_tag(), portal_endpoint_messages_tags(), portal_smtp_create_data(), portal_smtp_merge_headers(), portal_smtp_relay_message(), portal_upload(), register_abuse_check_blocklist(), smtp_bypass_check(), smtp_check_filters(), and teacher_print_message().

multi_t inx_cursor_key_active (inx_cursor_t * cursor)

Get the key at the current inx cursor position.

Parameters:

cursor the inx cursor to be examined.

Returns:

NULL on failure, or a multi-type data object containing the key of the specified inx cursor.

Definition at line 97 of file cursors.c.

References `inx_auto_read()`, `inx_auto_unlock()`, and `mt_get_null()`.

Referenced by `imap_search_messages()`, and `obj_cache_prune()`.

multi_t inx_cursor_key_next (inx_cursor_t * *cursor*)

Get the key at the next inx cursor position.

Parameters:

cursor the inx cursor to be examined.

Returns:

NULL on failure, or a multi-type data object containing the key of the next inx cursor position.

Definition at line 79 of file cursors.c.

References `inx_auto_read()`, `inx_auto_unlock()`, and `mt_get_null()`.

Referenced by `config_load_cmdline_settings()`, and `config_load_file_settings()`.

void inx_cursor_reset (inx_cursor_t * *cursor*)

Reset the position of an inx cursor.

Parameters:

cursor a pointer to the inx cursor to be reset.

Returns:

This function returns no value.

Definition at line 20 of file cursors.c.

Referenced by `imap_fetch()`, `imap_list()`, `imap_lsub()`, `obj_cache_prune()`, and `portal_smtp_create_data()`.

void* inx_cursor_value_active (inx_cursor_t * *cursor*)

Get the value at the current inx cursor position.

Parameters:

cursor the inx cursor to be examined.

Returns:

NULL on failure, or the value at the current position of the specified inx cursor.

Definition at line 133 of file cursors.c.

References `inx_auto_read()`, and `inx_auto_unlock()`.

Referenced by `config_load_cmdline_settings()`, and `config_load_file_settings()`.

void* inx_cursor_value_next (inx_cursor_t * cursor)

Get the key at the next inx cursor position.

Parameters:

cursor the inx cursor to be examined.

Returns:

NULL on failure, or a multi-type data object containing the key of the next inx cursor position.

Definition at line 115 of file cursors.c.

References inx_auto_read(), and inx_auto_unlock().

Referenced by contact_find_detail(), contact_find_name(), contacts_fetch(), contacts_update(), decrypt_user_messages(), encrypt_user_messages(), http_data_get(), imap_append(), imap_close(), imap_copy(), imap_duplicate_messages(), imap_examine(), imap_expunge(), imap_fetch(), imap_folder_remove(), imap_folder_status(), imap_list(), imap_lsub(), imap_narrow_folders(), imap_narrow_messages(), imap_next_folder_order(), imap_search(), imap_search_messages(), imap_select(), imap_session_destroy(), imap_session_update(), imap_store(), imap_update_flags(), magma_folder_children(), magma_folder_find_full_name(), magma_folder_find_name(), messages_update(), meta_check_message_encryption(), meta_data_fetch_messages(), meta_data_flags_add(), meta_data_flags_remove(), meta_data_flags_replace(), meta_folders_by_name(), meta_folders_by_number(), meta_folders_children(), meta_folders_stats_tags(), meta_message_by_number(), meta_messages_update_sequences(), obj_cache_prune(), pattern_check(), pop_get_last(), pop_get_message(), pop_list(), pop_session_destroy(), pop_session_reset(), pop_total_messages(), pop_total_size(), pop_uidl(), portal_config_collection(), portal_contact_details(), portal_endpoint_alert_list(), portal_endpoint_aliases(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_list(), portal_endpoint_folders_list(), portal_endpoint_folders_tags(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(), portal_endpoint_messages_tag(), portal_endpoint_messages_tags(), portal_session_create_data(), portal_smtp_merge_headers(), portal_smtp_relay_message(), portal_upload(), register_abuse_check_blocklist(), smtp_bypass_check(), smtp_check_filters(), and teacher_print_message().

bool_t inx_delete (inx_t * inx, multi_t key)

Delete a child of an inx object with a specified key.

Parameters:

inx a pointer to the inx object to be searched.

key the target key of the object to be deleted.

Returns:

true if the delete operation succeeded or false if it did not.

Definition at line 197 of file inx.c.

References inx_t::delete, inx_auto_unlock(), inx_auto_write(), and log_pedantic.

Referenced by contact_edit(), contact_folder_remove(), contact_move(), imap_folder_remove(), imap_message_expunge(), message_folder_remove(), meta_user_prune(), obj_cache_prune(), pop_session_destroy(), portal_endpoint_attachments_remove(), portal_endpoint_contacts_remove(), portal_endpoint_messages_remove(), portal_endpoint_messages_send(), sess_get(), and user_config_edit().

void* inx_find (inx_t * inx, multi_t key)

Find the value associated with a particular key within the children of an `inx` object.

Parameters:

inx a pointer to the `inx` object to be searched.
key the target key to be found.

Returns:

NULL on failure, or the value associated with the requested key on success.

Definition at line 221 of file `inx.c`.

References `inx_t::find`, `inx_auto_read()`, `inx_auto_unlock()`, and `log_pedantic`.

Referenced by `contact_find_detail()`, `contact_find_number()`, `contact_folder_remove()`, `contact_folder_rename()`, `domain_dkim()`, `domain_mailboxes()`, `domain_restricted()`, `domain_spf()`, `domain_wildcard()`, `http_get_static()`, `http_get_template()`, `imap_search_messages()`, `magma_folder_find_number()`, `message_folder_remove()`, `meta_folders_stats_tags()`, `meta_get()`, `meta_messages_mover()`, `meta_remove()`, `meta_user_prune()`, `portal_endpoint_attachments_add()`, `portal_endpoint_attachments_remove()`, `portal_endpoint_contacts_load()`, `portal_endpoint_messages_compose()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_load()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_send()`, `portal_endpoint_messages_tag()`, `portal_get_upload_attachment()`, `sess_get()`, and `user_config_edit()`.

`void inx_free (inx_t * inx)`

Definition at line 239 of file `inx.c`.

References `inx_t::index_free`, `inx_auto_unlock()`, `inx_auto_write()`, `inx_t::lock`, `log_pedantic`, `mm_free()`, `inx_t::references`, and `rwlock_destroy()`.

Referenced by `adjust_message_encryption()`, `contacts_update()`, `http_content_refresh()`, `imap_copy()`, `imap_duplicate_messages()`, `imap_fetch()`, `imap_list()`, `imap_lsub()`, `imap_narrow_messages()`, `imap_search()`, `imap_store()`, `inx_cleanup()`, `inx_cursor_free()`, `magma_folder_fetch()`, `messages_update()`, `meta_contacts_update()`, `meta_data_fetch_alerts()`, `meta_data_fetch_all_tags()`, `meta_message_folders_update()`, `nvp_free()`, `obj_cache_stop()`, `portal_endpoint_alert_list()`, `portal_endpoint_folders_tags()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_send()`, `portal_endpoint_messages_tag()`, `portal_endpoint_messages_tags()`, `portal_parse_json_str_array()`, `register_blocklist_free()`, `register_blocklist_update()`, `register_data_fetch_blocklist()`, `warehouse_fetch_domains()`, and `warehouse_fetch_patterns()`.

`bool_t inx_insert (inx_t * inx, multi_t key, void * data)`

Insert a new record into an `inx` holder.

Parameters:

inx a pointer to the `inx` object that will hold the record.
key a multi-type value specifying the identifier of the record to be inserted.
data a pointer to the data associated with the new key.

Returns:

true if the new record was inserted successfully or false on failure.

Definition at line 144 of file `inx.c`.

References `inx_t::insert`, `inx_auto_unlock()`, `inx_auto_write()`, and `log_pedantic`.

Referenced by `adjust_message_encryption()`, `contact_details_fetch()`, `contact_folder_create()`, `contact_move()`, `contacts_fetch()`, `http_content_load_fonts()`, `http_data_value_parse()`, `http_load_file()`, `http_parse_header()`, `imap_append_message()`, `imap_duplicate_messages()`, `imap_folder_create()`, `imap_folder_rename()`, `imap_message_copier()`, `imap_narrow_folders()`, `imap_narrow_messages()`, `imap_search_messages()`, `magma_folder_fetch()`, `message_folder_create()`, `messages_fetch()`, `meta_data_fetch_alerts()`, `meta_data_fetch_all_tags()`, `meta_data_fetch_folders()`, `meta_data_fetch_mailbox_aliases()`, `meta_data_fetch_messages()`, `meta_folders_stats_tags()`, `meta_get()`, `meta_messages_copier()`, `nvp_parse()`, `portal_endpoint_attachments_add()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_messages_compose()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_tag()`, `portal_parse_json_str_array()`, `register_data_fetch_blocklist()`, `sess_create()`, `smtp_add_bypass_entry()`, `smtp_fetch_inbound()`, `teacher_add_cookie()`, `user_config_fetch()`, `warehouse_fetch_domains()`, and `warehouse_fetch_patterns()`.

void `inx_lock_read` (`inx_t * inx`)

Acquire a reader's lock for an `inx` object.

Parameters:

inx a pointer to the `inx` object to be locked.

Returns:

This function returns no value.

Definition at line 40 of file `inx.c`.

References `inx_t::automatic`, `inx_t::lock`, and `rwlock_lock_read()`.

Referenced by `meta_remove()`, `obj_cache_prune()`, and `sess_get()`.

void `inx_lock_write` (`inx_t * inx`)

Acquire a writer's lock for an `inx` object.

Parameters:

inx a pointer to the `inx` object to be locked.

Returns:

This function returns no value.

Definition at line 58 of file `inx.c`.

References `inx_t::automatic`, `inx_t::lock`, and `rwlock_lock_write()`.

Referenced by `meta_get()`, `meta_user_prune()`, and `obj_cache_prune()`.

uint64_t `inx_options` (`inx_t * inx`)

Return the options value of an `inx` object.

Parameters:

inx a pointer to the `inx` object to be examined.

Returns:

0 on failure, or the options value of the `inx` object on success.

Definition at line 78 of file `inx.c`.

References `inx_auto_read()`, `inx_auto_unlock()`, `log_pedantic`, and `inx_t::options`.

`bool_t inx_replace (inx_t * inx, multi_t key, void * data)`

Replace the value of a key in an `inx` holder.

Parameters:

inx a pointer to the `inx` object where the specified key will be replaced.

key a multi-type value specifying the identifier of the record to be replaced.

data a pointer to the new data to be associated with the specified key.

Returns:

true if the specified key's value was replaced successfully, or false on failure.

Definition at line 169 of file `inx.c`.

References `inx_t::delete`, `inx_t::insert`, `inx_auto_unlock()`, `inx_auto_write()`, and `log_pedantic`.

Referenced by `contact_edit()`, and `user_config_edit()`.

`uint64_t inx_serial (inx_t * inx)`

Definition at line 119 of file `inx.c`.

References `inx_t::count`, `inx_auto_read()`, `inx_auto_unlock()`, and `log_pedantic`.

`void inx_truncate (inx_t * inx)`

Definition at line 264 of file `inx.c`.

References `inx_t::index_truncate`, `inx_auto_unlock()`, `inx_auto_write()`, and `log_pedantic`.

Referenced by `user_config_update()`.

`void inx_unlock (inx_t * inx)`

Unlock an `inx` object.

Parameters:

inx a pointer to the `inx` object to be unlocked.

Returns:

This function returns no value.

Definition at line 20 of file `inx.c`.

References `inx_t::automatic`, `inx_t::lock`, and `rwlock_unlock()`.

Referenced by `meta_get()`, `meta_remove()`, `meta_user_prune()`, `obj_cache_prune()`, and `sess_get()`.

`inx_t* linked_alloc (uint64_t options, void * data_free)`

`linked.c`

linked.c

Parameters:

options an options value for the newly created linked list object.

data_free a pointer to a function used to free linked list items.

Returns:

NULL on failure or a pointer to the newly allocated linked list object on success.

Definition at line 466 of file linked.c.

References `inx_t::cursor_alloc`, `inx_t::cursor_free`, `inx_t::cursor_key_active`, `inx_t::cursor_key_next`, `inx_t::cursor_reset`, `inx_t::cursor_value_active`, `inx_t::cursor_value_next`, `inx_t::data_free`, `inx_t::delete`, `inx_t::find`, `inx_t::index_free`, `inx_t::index_truncate`, `inx_t::insert`, `linked_cursor_alloc()`, `linked_cursor_free()`, `linked_cursor_key_active()`, `linked_cursor_key_next()`, `linked_cursor_reset()`, `linked_cursor_value_active()`, `linked_cursor_value_next()`, `linked_delete()`, `linked_find()`, `linked_free()`, `linked_insert()`, `linked_truncate()`, `mm_alloc()`, and `inx_t::options`.

Referenced by `inx_alloc()`.

magma/core/indexes/inx.c File Reference

The generic index interface used to abstract away the underlying data structure used for storage.

```
#include "magma.h"
```

Functions

- void **inx_unlock** (**inx_t** *inx)
- *Unlock an inx object.* void **inx_auto_unlock** (**inx_t** *inx)
- void **inx_lock_read** (**inx_t** *inx)
- *Acquire a reader's lock for an inx object.* void **inx_auto_read** (**inx_t** *inx)
- void **inx_lock_write** (**inx_t** *inx)
- *Acquire a writer's lock for an inx object.* void **inx_auto_write** (**inx_t** *inx)
- uint64_t **inx_options** (**inx_t** *inx)
- *Return the options value of an inx object.* uint64_t **inx_count** (**inx_t** *inx)
- *Return the total number of items held by an inx object.* uint64_t **inx_serial** (**inx_t** *inx)
- **bool_t** **inx_insert** (**inx_t** *inx, **multi_t** key, void *data)
- *Insert a new record into an inx holder.* **bool_t** **inx_replace** (**inx_t** *inx, **multi_t** key, void *data)
- *Replace the value of a key in an inx holder.* **bool_t** **inx_delete** (**inx_t** *inx, **multi_t** key)
- *Delete a child of an inx object with a specified key.* void * **inx_find** (**inx_t** *inx, **multi_t** key)
- *Find the value associated with a particular key within the children of an inx object.* void **inx_free** (**inx_t** *inx)
- void **inx_truncate** (**inx_t** *inx)
- void **inx_cleanup** (**inx_t** *inx)
- *Perform a checked free of an inx object.* **inx_t** * **inx_alloc** (uint64_t options, void *data_free)

Allocate a new inx instance.

Detailed Description

The generic index interface used to abstract away the underlying data structure used for storage.

Definition in file **inx.c**.

Function Documentation

inx_t* **inx_alloc** (uint64_t *options*, void * *data_free*)

Allocate a new inx instance.

inx.c

Parameters:

options a value indicating the inx type. Can be M_INX_TREE for a binary tree, M_INX_LINKED for a linked list, or M_INX_HASHED for a hash tree.

data_free a function pointer to a routine to free the data associated with an inx record.

Returns:

NULL on failure or a pointer to the newly created inx object on success.

Definition at line 298 of file inx.c.

References `inx_t::automatic`, `hashed_alloc()`, `linked_alloc()`, `inx_t::lock`, `log_options`, `M_INX_HASHED`, `M_INX_LINKED`, `M_INX_LOCK_MANUAL`, `M_INX_TREE`, `M_LOG_ERROR`, `M_LOG_STACK_TRACE`, `MAGMA_INDEX_TYPE`, `inx_t::references`, `rwlock_init()`, and `tree_alloc()`.

Referenced by `adjust_message_encryption()`, `contact_alloc()`, `contact_folder_alloc()`, `http_content_refresh()`, `http_content_start()`, `http_parse_header()`, `http_parse_pairs()`, `imap_duplicate_messages()`, `imap_narrow_folders()`, `imap_narrow_messages()`, `imap_search_messages()`, `magma_folder_fetch()`, `message_folder_alloc()`, `meta_data_fetch_alerts()`, `meta_data_fetch_all_tags()`, `meta_data_fetch_folders()`, `meta_data_fetch_mailbox_aliases()`, `meta_data_fetch_messages()`, `meta_folders_stats_tags()`, `nvp_alloc()`, `obj_cache_start()`, `portal_endpoint_messages_compose()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_tag()`, `portal_parse_json_str_array()`, `register_data_fetch_blocklist()`, `sess_create()`, `smtp_add_bypass_entry()`, `smtp_fetch_inbound()`, `teacher_add_cookie()`, `user_config_alloc()`, `warehouse_fetch_domains()`, and `warehouse_fetch_patterns()`.

void inx_auto_read (inx_t * inx)

Definition at line 47 of file `inx.c`.

References `inx_t::automatic`, `inx_t::lock`, and `rwlock_lock_read()`.

Referenced by `inx_count()`, `inx_cursor_key_active()`, `inx_cursor_key_next()`, `inx_cursor_value_active()`, `inx_cursor_value_next()`, `inx_find()`, `inx_options()`, and `inx_serial()`.

void inx_auto_unlock (inx_t * inx)

Definition at line 28 of file `inx.c`.

References `inx_t::automatic`, `inx_t::lock`, and `rwlock_unlock()`.

Referenced by `inx_count()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_key_active()`, `inx_cursor_key_next()`, `inx_cursor_value_active()`, `inx_cursor_value_next()`, `inx_delete()`, `inx_find()`, `inx_free()`, `inx_insert()`, `inx_options()`, `inx_replace()`, `inx_serial()`, and `inx_truncate()`.

void inx_auto_write (inx_t * inx)

Definition at line 66 of file `inx.c`.

References `inx_t::automatic`, `inx_t::lock`, and `rwlock_lock_write()`.

Referenced by `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_delete()`, `inx_free()`, `inx_insert()`, `inx_replace()`, and `inx_truncate()`.

void inx_cleanup (inx_t * inx)

Perform a checked free of an `inx` object.

See also:

`inx_free()`

Parameters:

inx the `inx` instance to be freed.

Definition at line 285 of file `inx.c`.

References `inx_free()`.

Referenced by `contact_free()`, `contacts_update()`, `domain_stop()`, `http_content_stop()`, `http_session_reset()`, `imap_narrow_messages()`, `magma_folder_free()`, `messages_update()`, `meta_data_fetch_folders()`, `meta_data_fetch_mailbox_aliases()`, `meta_data_fetch_messages()`, `meta_user_destroy()`, `pattern_stop()`, `pattern_update()`, `sess_destroy()`, `sess_release_composition()`, `smtp_free_inbound()`, and `user_config_free()`.

uint64_t inx_count (inx_t * inx)

Return the total number of items held by an inx object.

Parameters:

inx a pointer to the inx object to be examined.

Returns:

the total number of items held by the inx object.

Definition at line 101 of file `inx.c`.

References `inx_t::count`, `count`, `inx_auto_read()`, `inx_auto_unlock()`, and `log_pedantic`.

Referenced by `contact_folder_remove()`, `imap_copy()`, `imap_list()`, `imap_login()`, `imap_narrow_messages()`, `message_folder_remove()`, `meta_user_prune()`, `obj_cache_prune()`, and `pop_pass()`.

bool_t inx_delete (inx_t * inx, multi_t key)

Delete a child of an inx object with a specified key.

Parameters:

inx a pointer to the inx object to be searched.

key the target key of the object to be deleted.

Returns:

true if the delete operation succeeded or false if it did not.

Definition at line 197 of file `inx.c`.

References `inx_t::delete`, `inx_auto_unlock()`, `inx_auto_write()`, and `log_pedantic`.

Referenced by `contact_edit()`, `contact_folder_remove()`, `contact_move()`, `imap_folder_remove()`, `imap_message_expunge()`, `message_folder_remove()`, `meta_user_prune()`, `obj_cache_prune()`, `pop_session_destroy()`, `portal_endpoint_attachments_remove()`, `portal_endpoint_contacts_remove()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_send()`, `sess_get()`, and `user_config_edit()`.

void* inx_find (inx_t * inx, multi_t key)

Find the value associated with a particular key within the children of an inx object.

Parameters:

inx a pointer to the inx object to be searched.

key the target key to be found.

Returns:

NULL on failure, or the value associated with the requested key on success.

Definition at line 221 of file `inx.c`.

References `inx_t::find`, `inx_auto_read()`, `inx_auto_unlock()`, and `log_pedantic`.

Referenced by contact_find_detail(), contact_find_number(), contact_folder_remove(), contact_folder_rename(), domain_dkim(), domain_mailboxes(), domain_restricted(), domain_spf(), domain_wildcard(), http_get_static(), http_get_template(), imap_search_messages(), magma_folder_find_number(), message_folder_remove(), meta_folders_stats_tags(), meta_get(), meta_messages_mover(), meta_remove(), meta_user_prune(), portal_endpoint_attachments_add(), portal_endpoint_attachments_remove(), portal_endpoint_contacts_load(), portal_endpoint_messages_compose(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_load(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(), portal_endpoint_messages_send(), portal_endpoint_messages_tag(), portal_get_upload_attachment(), sess_get(), and user_config_edit().

void inx_free (inx_t * inx)

Definition at line 239 of file inx.c.

References inx_t::index_free, inx_auto_unlock(), inx_auto_write(), inx_t::lock, log_pedantic, mm_free(), inx_t::references, and rwlock_destroy().

Referenced by adjust_message_encryption(), contacts_update(), http_content_refresh(), imap_copy(), imap_duplicate_messages(), imap_fetch(), imap_list(), imap_lsub(), imap_narrow_messages(), imap_search(), imap_store(), inx_cleanup(), inx_cursor_free(), magma_folder_fetch(), messages_update(), meta_contacts_update(), meta_data_fetch_alerts(), meta_data_fetch_all_tags(), meta_message_folders_update(), nvp_free(), obj_cache_stop(), portal_endpoint_alert_list(), portal_endpoint_folders_tags(), portal_endpoint_messages_flag(), portal_endpoint_messages_send(), portal_endpoint_messages_tag(), portal_endpoint_messages_tags(), portal_parse_json_str_array(), register_blocklist_free(), register_blocklist_update(), register_data_fetch_blocklist(), warehouse_fetch_domains(), and warehouse_fetch_patterns().

bool_t inx_insert (inx_t * inx, multi_t key, void * data)

Insert a new record into an inx holder.

Parameters:

inx a pointer to the inx object that will hold the record.
key a multi-type value specifying the identifier of the record to be inserted.
data a pointer to the data associated with the new key.

Returns:

true if the new record was inserted successfully or false on failure.

Definition at line 144 of file inx.c.

References inx_t::insert, inx_auto_unlock(), inx_auto_write(), and log_pedantic.

Referenced by adjust_message_encryption(), contact_details_fetch(), contact_folder_create(), contact_move(), contacts_fetch(), http_content_load_fonts(), http_data_value_parse(), http_load_file(), http_parse_header(), imap_append_message(), imap_duplicate_messages(), imap_folder_create(), imap_folder_rename(), imap_message_copier(), imap_narrow_folders(), imap_narrow_messages(), imap_search_messages(), magma_folder_fetch(), message_folder_create(), messages_fetch(), meta_data_fetch_alerts(), meta_data_fetch_all_tags(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_messages(), meta_folders_stats_tags(), meta_get(), meta_messages_copier(), nvp_parse(), portal_endpoint_attachments_add(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_messages_compose(), portal_endpoint_messages_flag(), portal_endpoint_messages_tag(), portal_parse_json_str_array(), register_data_fetch_blocklist(), sess_create(), smtp_add_bypass_entry(), smtp_fetch_inbound(), teacher_add_cookie(), user_config_fetch(), warehouse_fetch_domains(), and warehouse_fetch_patterns().

void inx_lock_read (inx_t * *inx*)

Acquire a reader's lock for an inx object.

Parameters:

inx a pointer to the inx object to be locked.

Returns:

This function returns no value.

Definition at line 40 of file inx.c.

References inx_t::automatic, inx_t::lock, and rwlock_lock_read().

Referenced by meta_remove(), obj_cache_prune(), and sess_get().

void inx_lock_write (inx_t * *inx*)

Acquire a writer's lock for an inx object.

Parameters:

inx a pointer to the inx object to be locked.

Returns:

This function returns no value.

Definition at line 58 of file inx.c.

References inx_t::automatic, inx_t::lock, and rwlock_lock_write().

Referenced by meta_get(), meta_user_prune(), and obj_cache_prune().

uint64_t inx_options (inx_t * *inx*)

Return the options value of an inx object.

Parameters:

inx a pointer to the inx object to be examined.

Returns:

0 on failure, or the options value of the inx object on success.

Definition at line 78 of file inx.c.

References inx_auto_read(), inx_auto_unlock(), log_pedantic, and inx_t::options.

bool_t inx_replace (inx_t * *inx*, multi_t *key*, void * *data*)

Replace the value of a key in an inx holder.

Parameters:

inx a pointer to the inx object where the specified key will be replaced.

key a multi-type value specifying the identifier of the record to be replaced.

data a pointer to the new data to be associated with the specified key.

Returns:

true if the specified key's value was replaced successfully, or false on failure.

Definition at line 169 of file `inx.c`.

References `inx_t::delete`, `inx_t::insert`, `inx_auto_unlock()`, `inx_auto_write()`, and `log_pedantic`.

Referenced by `contact_edit()`, and `user_config_edit()`.

uint64_t inx_serial (inx_t * *inx*)

Definition at line 119 of file `inx.c`.

References `inx_t::count`, `inx_auto_read()`, `inx_auto_unlock()`, and `log_pedantic`.

void inx_truncate (inx_t * *inx*)

Definition at line 264 of file `inx.c`.

References `inx_t::index_truncate`, `inx_auto_unlock()`, `inx_auto_write()`, and `log_pedantic`.

Referenced by `user_config_update()`.

void inx_unlock (inx_t * *inx*)

Unlock an `inx` object.

Parameters:

inx a pointer to the `inx` object to be unlocked.

Returns:

This function returns no value.

Definition at line 20 of file `inx.c`.

References `inx_t::automatic`, `inx_t::lock`, and `rwlock_unlock()`.

Referenced by `meta_get()`, `meta_remove()`, `meta_user_prune()`, `obj_cache_prune()`, and `sess_get()`.

magma/core/indexes/linked.c File Reference

The linked list implementation functions utilized by the generic index interface.

```
#include "magma.h"
```

Data Structures

- struct **linked_record_t**
- struct **linked_node_t**
- struct **__attribute__**

Functions

- void * **linked_record_get_data** (**linked_record_t** *record, size_t element)
 - *Get the data associated with a linked list record. multi_t* **linked_record_get_key** (**linked_record_t** *record)
 - *Get the multi-type key of a linked list record. void* **linked_record_free** (**inx_t** *index, **linked_record_t** *record)
 - *Free a linked list record object and its underlying data. linked_record_t ** **linked_record_alloc** (**multi_t** key, void *data)
 - *Create a new linked list record object. void ** **linked_find** (void *inx, **multi_t** key)
 - *Find a record in a linked list by key. bool_t* **linked_delete** (void *inx, **multi_t** key)
 - *Remove a record from a linked list and free it and its underlying data. bool_t* **linked_insert** (void *inx, **multi_t** key, void *data)
 - *Create and append a new record to the end of a linked list. linked_node_t ** **linked_cursor_active** (**linked_cursor_t** *cursor)
 - *Get the current node pointed to by a linked list cursor. linked_node_t ** **linked_cursor_next** (**linked_cursor_t** *cursor)
 - *Get the next node of a linked list cursor. void ** **linked_cursor_value_next** (**linked_cursor_t** *cursor)
 - *Get the data of a linked list cursor's next record, and update the cursor. void ** **linked_cursor_value_active** (**linked_cursor_t** *cursor)
 - *Get the data of a linked list cursor's current record. multi_t* **linked_cursor_key_next** (**linked_cursor_t** *cursor)
 - *Get the multi-type key value of a linked list cursor's next record, and update the cursor. multi_t* **linked_cursor_key_active** (**linked_cursor_t** *cursor)
 - *Get the multi-type key value at the current linked list cursor position. void* **linked_cursor_reset** (**linked_cursor_t** *cursor)
 - *Reset the position of a linked list cursor. void* **linked_cursor_free** (**linked_cursor_t** *cursor)
 - *Free a linked list cursor. void ** **linked_cursor_alloc** (**inx_t** *inx)
 - *Allocate a cursor to traverse a linked list. void* **linked_truncate** (void *inx)
 - *Truncate all the nodes in a linked list, but do not free it. void* **linked_free** (void *inx)
 - *Free a linked list object and all of its records. inx_t ** **linked_alloc** (uint64_t options, void *data_free)
- Allocate a new linked list instance.*
-

Detailed Description

The linked list implementation functions utilized by the generic index interface.

Definition in file **linked.c**.

Function Documentation

inx_t* linked_alloc (uint64_t *options*, void * *data_free*)

Allocate a new linked list instance.

linked.c

Parameters:

options an options value for the newly created linked list object.

data_free a pointer to a function used to free linked list items.

Returns:

NULL on failure or a pointer to the newly allocated linked list object on success.

Definition at line 466 of file linked.c.

References inx_t::cursor_alloc, inx_t::cursor_free, inx_t::cursor_key_active, inx_t::cursor_key_next, inx_t::cursor_reset, inx_t::cursor_value_active, inx_t::cursor_value_next, inx_t::data_free, inx_t::delete, inx_t::find, inx_t::index_free, inx_t::index_truncate, inx_t::insert, linked_cursor_alloc(), linked_cursor_free(), linked_cursor_key_active(), linked_cursor_key_next(), linked_cursor_reset(), linked_cursor_value_active(), linked_cursor_value_next(), linked_delete(), linked_find(), linked_free(), linked_insert(), linked_truncate(), mm_alloc(), and inx_t::options.

Referenced by inx_alloc().

linked_node_t* linked_cursor_active (linked_cursor_t * *cursor*)

Get the current node pointed to by a linked list cursor.

Parameters:

cursor a pointer to the linked list cursor to be queried.

Returns:

a pointer to the current node indicated by the specified linked list cursor.

LOW: The logic used to find our place in an index that has been modified could be improved.

- If the index we sort the index keys, we could look for the next highest key value.

We could store the previous node (or even the previous 10 nodes), and try searching for them instead.

- We could also add a delete path through the cursor that would include positional updates.
- We could simply make a deep copy of the entire list so that updates to the original index don't affect the iteration, or use the copy to reconcile.

Definition at line 229 of file linked.c.

References linked_node_t::next.

Referenced by linked_cursor_key_active(), linked_cursor_next(), and linked_cursor_value_active().

void* linked_cursor_alloc (inx_t * *inx*)

Allocate a cursor to traverse a linked list.

Parameters:

inx a pointer the linked list object to be traversed by the cursor.

Returns:

NULL on failure, or a cursor pointing to the head of the linked list on success.

Definition at line 399 of file linked.c.

References log_pedantic, and mm_alloc().

Referenced by linked_alloc().

void linked_cursor_free (linked_cursor_t * cursor)

Free a linked list cursor.

Parameters:

cursor a pointer to the linked list cursor object to be freed.

Returns:

This function returns no value.

Definition at line 385 of file linked.c.

References mm_free().

Referenced by linked_alloc().

multi_t linked_cursor_key_active (linked_cursor_t * cursor)

Get the multi-type key value at the current linked list cursor position.

Parameters:

cursor a pointer to the linked list cursor to be queried.

Returns:

an empty multi-type key on failure, or the current linked list cursor's record key on success.

Definition at line 354 of file linked.c.

References linked_cursor_active(), linked_record_get_key(), mt_get_null(), and linked_node_t::record.

Referenced by linked_alloc().

multi_t linked_cursor_key_next (linked_cursor_t * cursor)

Get the multi-type key value of a linked list cursor's next record, and update the cursor.

Parameters:

cursor a pointer to the linked list cursor to be queried.

Returns:

an empty multi-type key on failure, or the linked list cursor's next record key on success.

Definition at line 338 of file linked.c.

References linked_cursor_next(), linked_record_get_key(), mt_get_null(), and linked_node_t::record.

Referenced by linked_alloc().

linked_node_t* linked_cursor_next (linked_cursor_t * *cursor*)

Get the next node of a linked list cursor.

Note:

The cursor position will be reset if the end of the linked list has been reached.

Parameters:

cursor a pointer to the linked list cursor to be queried.

Returns:

a pointer to the next node of the linked list cursor.

Definition at line 286 of file linked.c.

References linked_cursor_active(), and linked_node_t::next.

Referenced by linked_cursor_key_next(), and linked_cursor_value_next().

void linked_cursor_reset (linked_cursor_t * *cursor*)

Reset the position of a linked list cursor.

Parameters:

cursor a pointer to the linked list cursor object to be reset.

Returns:

This function returns no value.

Definition at line 370 of file linked.c.

Referenced by linked_alloc().

void* linked_cursor_value_active (linked_cursor_t * *cursor*)

Get the data of a linked list cursor's current record.

Parameters:

cursor a pointer to the linked list cursor to be queried.

Returns:

NULL on failure, or a pointer to the data of the linked list cursor's current record.

Definition at line 323 of file linked.c.

References linked_cursor_active(), linked_record_get_data(), and linked_node_t::record.

Referenced by linked_alloc().

void* linked_cursor_value_next (linked_cursor_t * *cursor*)

Get the data of a linked list cursor's next record, and update the cursor.

Parameters:

cursor a pointer to the linked list cursor to be queried.

Returns:

NULL on failure, or the data of the linked list cursor's next record on success.

Definition at line 308 of file linked.c.

References `linked_cursor_next()`, `linked_record_get_data()`, and `linked_node_t::record`.

Referenced by `linked_alloc()`.

bool_t linked_delete (void * *inx*, multi_t *key*)

Remove a record from a linked list and free it and its underlying data.

Parameters:

inx a pointer to the linked list to be searched for the specified key.

key a multi-type key value to lookup the record that will be deleted from the linked list.

Returns:

true on success or false on failure.

Definition at line 142 of file linked.c.

References `inx_t::count`, `ident_mt_mt()`, `inx_t::index`, `linked_record_t::key`, `linked_record_free()`, `mm_free()`, `linked_node_t::next`, `linked_node_t::prev`, `linked_node_t::record`, and `inx_t::serial`.

Referenced by `linked_alloc()`.

void* linked_find (void * *inx*, multi_t *key*)

Find a record in a linked list by key.

Parameters:

inx a pointer to the linked list to be searched.

key a multi-type key value to be searched against the contents of the *inx* object.

Returns:

NULL on failure or if the record cannot be found, or a pointer to the data of the matching record on success.

Definition at line 112 of file linked.c.

References `inx_t::count`, `ident_mt_mt()`, `inx_t::index`, `linked_record_t::key`, `linked_record_get_data()`, `linked_node_t::next`, and `linked_node_t::record`.

Referenced by `linked_alloc()`.

void linked_free (void * *inx*)

Free a linked list object and all of its records.

See also:

`linked_truncate()`

Parameters:

inx a pointer to the linked list object to be freed.

Returns:

This function returns no value.

Definition at line 446 of file linked.c.

References `inx_t::index`, and `linked_truncate()`.

Referenced by `linked_alloc()`.

`bool_t linked_insert (void * inx, multi_t key, void * data)`

Create and append a new record to the end of a linked list.

Parameters:

inx a pointer to the linked list that will store the new record.

key a multi-type key value that will be associated with the newly created record.

data a pointer to the data that will be associated with the new record.

Returns:

true on success or false on failure.

Definition at line 189 of file linked.c.

References `inx_t::count`, `inx_t::index`, `linked_record_alloc()`, `log_info`, `mm_alloc()`, `mm_free()`, `linked_node_t::next`, `linked_node_t::prev`, `linked_node_t::record`, and `inx_t::serial`.

Referenced by `linked_alloc()`.

`linked_record_t* linked_record_alloc (multi_t key, void * data)`

Create a new linked list record object.

Parameters:

key the multi-type key value of the record to be used for data searches.

data a pointer to the data to be associated with the record.

Returns:

a pointer to the newly allocated and initialized linked record object.

Definition at line 93 of file linked.c.

References `linked_record_t::count`, `linked_record_t::data`, `linked_record_t::key`, `mm_alloc()`, and `mt_dupe()`.

Referenced by `linked_insert()`.

`void linked_record_free (inx_t * index, linked_record_t * record)`

Free a linked list record object and its underlying data.

Parameters:

index a pointer to the linked list containing the specified record.

record a pointer to the linked list record to be freed.

Returns:

This function returns no value.

Definition at line 72 of file linked.c.

References `linked_record_t::data`, `inx_t::data_free`, `linked_record_t::key`, `log_pedantic`, `mm_free()`, and `mt_free()`.

Referenced by `linked_delete()`, and `linked_truncate()`.

`void* linked_record_get_data (linked_record_t * record, size_t element)`

Get the data associated with a linked list record.

Parameters:

record a pointer to the linked list record to be queried.

element the index of the data element to be retrieved. Must be set to zero.

Returns:

NULL on failure, or a pointer to the specified linked list record's data on success.

Definition at line 39 of file linked.c.

References `log_pedantic`.

Referenced by `linked_cursor_value_active()`, `linked_cursor_value_next()`, and `linked_find()`.

`multi_t linked_record_get_key (linked_record_t * record)`

Get the multi-type key of a linked list record.

Parameters:

record a pointer to the linked list record to be queried.

Returns:

an empty multi-type key on failure, or the specified record's multi-type key value on success.

Definition at line 54 of file linked.c.

References `linked_record_t::key`, `log_pedantic`, and `mt_get_null()`.

Referenced by `linked_cursor_key_active()`, and `linked_cursor_key_next()`.

`void linked_truncate (void * inx)`

Truncate all the nodes in a linked list, but do not free it.

Parameters:

inx a pointer to the linked list object to have all of its records truncated.

Returns:

This function returns no value.

Definition at line 418 of file linked.c.

References `inx_t::index`, `linked_record_free()`, `mm_free()`, `linked_node_t::next`, and `linked_node_t::record`.

Referenced by `linked_alloc()`, and `linked_free()`.

magma/core/log/log.c File Reference

Internal logging functions. This function should be accessed using the appropriate macro.

```
#include "magma.h"
```

Functions

- void **log_disable** (void)
- *Disable logging.* void **log_enable** (void)
- *Enable logging.* int_t **print_backtrace** ()
- *Print the current stack backtrace to stdout.* void **log_internal** (const char *file, const char *function, const int line, **M_LOG_OPTIONS** options, const char *format,...)
- void **log_rotate** (void)
- bool_t **log_start** (void)
- void **debug_hook** (void)

A stub routine where it is convenient to set a breakpoint in a debugger. Variables

- uint64_t **log_date**
- bool_t **log_enabled** = true
- pthread_mutex_t **log_mutex** = PTHREAD_MUTEX_INITIALIZER

Detailed Description

Internal logging functions. This function should be accessed using the appropriate macro.

Definition in file **log.c**.

Function Documentation

void debug_hook (void)

A stub routine where it is convenient to set a breakpoint in a debugger.

Note:

This function should be called by code paths that detect sanity check failures, but that are not fatal.

Returns:

This function returns no value.

Definition at line 272 of file log.c.

References log_pedantic.

Referenced by base64_encode_mod(), http_print_500(), res_field_generic(), st_data_get(), and st_free().

void log_disable (void)

Disable logging.

Returns:

This function returns no value.

Definition at line 23 of file log.c.

References log_enabled, log_mutex, mutex_lock(), and mutex_unlock().

void log_enable (void)

Enable logging.

Returns:

This function returns no value.

Definition at line 34 of file log.c.

References log_enabled, log_mutex, mutex_lock(), and mutex_unlock().

void log_internal (const char * file, const char * function, const int line, M_LOG_OPTIONS options, const char * format, ...)

Logs the message provided by *format* . The global configuration dictates whether the *file* , *function* and *line* are also logged. The global configuration can be overridden on a per call basis using the *options* parameter.

Definition at line 105 of file log.c.

References magma_t::file, magma_t::function, magma_t::line, magma_t::log, log_enabled, log_mutex, M_LOG_FILE, M_LOG_FILE_DISABLE, M_LOG_FUNCTION, M_LOG_FUNCTION_DISABLE, M_LOG_LINE, M_LOG_LINE_DISABLE, M_LOG_LINE_FEED_DISABLE, M_LOG_STACK_TRACE, M_LOG_STACK_TRACE_DISABLE, M_LOG_TIME, M_LOG_TIME_DISABLE, magma, mutex_lock(), mutex_unlock(), ns_length_get(), print_backtrace(), magma_t::stack, and magma_t::time.

void log_rotate (void)

Definition at line 184 of file log.c.

References magma_t::file, log_critical, log_date, log_mutex, magma, MEMORYBUF, ns_length_get(), magma_t::output, magma_t::path, and time_datestamp().

Referenced by process_maint().

bool_t log_start (void)

Definition at line 226 of file log.c.

References magma_t::file, folder_exists(), log_critical, log_date, magma, MEMORYBUF, ns_length_get(), NULLER, magma_t::output, magma_t::path, and time_datestamp().

Referenced by process_start().

int_t print_backtrace ()

Print the current stack backtrace to stdout.

log.c

Note:

This function was created because `backtrace_symbols()` can fail due to heap corruption.

Returns:

-1 if the backtrace failed, or 0 on success.

Definition at line 46 of file `log.c`.

Referenced by `log_internal()`.

Variable Documentation

uint64_t log_date

Definition at line 15 of file `log.c`.

Referenced by `log_rotate()`, and `log_start()`.

bool_t log_enabled = true

Definition at line 16 of file `log.c`.

Referenced by `log_disable()`, `log_enable()`, and `log_internal()`.

pthread_mutex_t log_mutex = PTHREAD_MUTEX_INITIALIZER

Definition at line 17 of file `log.c`.

Referenced by `log_disable()`, `log_enable()`, `log_internal()`, `log_rotate()`, and `xml_error()`.

magma/core/log/log.h File Reference

The function declarations and macros needed to access the error log subsystem.

Defines

- `#define MAGMA_LOG_LEVELS (M_LOG_PEDANTIC | M_LOG_INFO | M_LOG_INFO | M_LOG_CRITICAL)`
- `#define log_pedantic(...)`
- `#define log_check(expr)`
- `#define log_info(...) log_internal (__FILE__, __FUNCTION__, __LINE__, M_LOG_INFO, __VA_ARGS__)`
- `#define log_error(...) log_internal (__FILE__, __FUNCTION__, __LINE__, M_LOG_CRITICAL, __VA_ARGS__)`
- `#define log_critical(...) log_internal (__FILE__, __FUNCTION__, __LINE__, M_LOG_CRITICAL, __VA_ARGS__)`
- `#define log_options(options,...) log_internal (__FILE__, __FUNCTION__, __LINE__, options, __VA_ARGS__)`

Enumerations

- `enum M_LOG_OPTIONS { M_LOG_PEDANTIC = 1, M_LOG_INFO, M_LOG_ERROR, M_LOG_CRITICAL, M_LOG_TIME, M_LOG_FILE, M_LOG_LINE, M_LOG_FUNCTION, M_LOG_STACK_TRACE, M_LOG_PEDANTIC_DISABLE, M_LOG_INFO_DISABLE, M_LOG_ERROR_DISABLE, M_LOG_CRITICAL_DISABLE, M_LOG_LINE_FEED_DISABLE, M_LOG_TIME_DISABLE, M_LOG_FILE_DISABLE, M_LOG_LINE_DISABLE, M_LOG_FUNCTION_DISABLE, M_LOG_STACK_TRACE_DISABLE }`

Functions

- `int_t print_backtrace ()`
- *log.c* `void log_internal (const char *file, const char *function, const int line, M_LOG_OPTIONS options, const char *format,...) __attribute__((format(printf`
- `void log_disable (void)`
- *Disable logging.* `void log_enable (void)`
- *Enable logging.* `void log_rotate (void)`
- `bool_t log_start (void)`
- `void debug_hook (void)`

A stub routine where it is convenient to set a breakpoint in a debugger.

Detailed Description

The function declarations and macros needed to access the error log subsystem.

Definition in file **log.h**.

Define Documentation

#define log_check(expr)

Definition at line 68 of file log.h.

Referenced by `contact_delete()`, `contact_detail_delete()`, `contact_detail_upsert()`, `contact_update()`, `contact_update_stamp()`, `hashed_bucket()`, `pool_get_status()`, `pool_set_status()`, `protocol_enqueue()`, `protocol_secure()`, `st_swap()`, `tank_load()`, `user_config_delete()`, `user_config_upsert()`, `virus_engine_destroy()`, and `warehouse_fetch_domains()`.

#define log_critical(...) log_internal (__FILE__, __FUNCTION__, __LINE__, M_LOG_CRITICAL, __VA_ARGS__)

Used to record errors that could cause system instability.

Definition at line 85 of file `log.h`.

Referenced by `args_parse()`, `cache_alloc()`, `cache_config()`, `cache_set_value()`, `cache_start()`, `cache_validate()`, `config_load_cmdline_settings()`, `config_load_database_settings()`, `config_load_file_settings()`, `config_validate_settings()`, `config_value_set()`, `lib_load()`, `lib_load_lzo()`, `lib_symbols()`, `log_rotate()`, `log_start()`, `net_init()`, `net_listen()`, `obj_cache_start()`, `process_start()`, `process_stop()`, `relay_alloc()`, `relay_config()`, `relay_set_value()`, `relay_validate()`, `requeue()`, `servers_alloc()`, `servers_config()`, `servers_set_value()`, `servers_validate()`, `signal_shutdown()`, `spool_check()`, `spool_mktemp()`, `spool_start()`, `sql_open()`, `sql_start()`, `ssl_server_create()`, `ssl_start()`, `stats_init()`, `stmt_bind_param()`, `stmt_close()`, `stmt_open()`, `stmt_prepare()`, `stmt_rebuild()`, `stmt_reset()`, `stmt_start()`, `tank_delete()`, `tank_open()`, `tank_start()`, and `virus_start()`.

#define log_error(...) log_internal (__FILE__, __FUNCTION__, __LINE__, M_LOG_CRITICAL, __VA_ARGS__)

Used to log errors that may indicate a problem requiring user intervention to solve.

Definition at line 80 of file `log.h`.

Referenced by `ar_alloc()`, `base64_encode()`, `config_fetch_host_number()`, `contact_business()`, `dequeue()`, `dspam_check()`, `ecies_group()`, `ecies_start()`, `hex_encode_st_debug()`, `imap_build_array()`, `imap_duplicate_messages()`, `imap_fetch_response_add()`, `imap_folder_remove()`, `imap_folder_rename()`, `imap_narrow_messages()`, `imap_parse_dataitems()`, `imap_parse_nstring()`, `imap_update_flags()`, `mail_copy_message()`, `mail_create_directory()`, `mail_db_delete_message()`, `mail_mime_boundary()`, `mail_mime_generate_boundary()`, `mail_mime_get_smtp_envelope()`, `mail_move_message()`, `mail_store_message()`, `mail_store_message_data()`, `meta_data_fetch_alerts()`, `meta_data_fetch_all_tags()`, `meta_data_fetch_folders()`, `meta_data_fetch_mailbox_aliases()`, `meta_data_fetch_messages()`, `meta_get()`, `meta_messages_copier()`, `meta_user_prune()`, `net_set_non_blocking()`, `portal_config_collection()`, `portal_contact_details()`, `portal_endpoint_attachments_add()`, `portal_endpoint_attachments_remove()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_messages_compose()`, `portal_endpoint_messages_send()`, `portal_endpoint_messages_tags()`, `portal_parse_json_str_array()`, `portal_upload()`, `queue_init()`, `sess_get()`, `signal_refresh()`, `smtp_accept_message()`, `smtp_add_bypass_entry()`, `smtp_fetch_authorization()`, `smtp_fetch_inbound()`, `smtp_get_action()`, `smtp_store_message()`, `spool_check_file()`, `spool_cleanup()`, `sql_ping()`, `sql_thread_start()`, `ssl_dh_exchange_callback()`, `ssl_ecdh_exchange_callback()`, `stacker_push()`, `tank_close()`, `tank_delete()`, `tank_load()`, `tank_store()`, `virus_check()`, `virus_engine_create()`, `virus_engine_refresh()`, and `virus_sigs_total()`.

#define log_info(...) log_internal (__FILE__, __FUNCTION__, __LINE__, M_LOG_INFO, __VA_ARGS__)

Used to record information related to daemon performance.

Definition at line 75 of file `log.h`.

Referenced by `args_parse()`, `cache_add()`, `cache_append()`, `cache_delete()`, `cache_flush()`, `cache_get()`, `cache_get_u64()`, `cache_output_help()`, `cache_output_settings()`, `cache_set()`, `cache_set_u64()`, `compress_bzip()`, `compress_lzo()`, `compress_zlib()`, `con_init_network_buffer()`, `config_load_defaults()`, `config_output_help()`, `config_output_settings()`, `config_output_value()`, `config_output_value_generic()`, `contact_details_fetch()`, `contacts_fetch()`, `decompress_block_lzo()`, `decompress_bzip()`, `decompress_lzo()`, `decompress_zlib()`, `decrypt_user_messages()`, `display_usage()`, `dspam_check()`, `dspam_train()`, `ecies_decrypt()`, `ecies_encrypt()`,

ecies_key_alloc(), ecies_key_create(), ecies_key_private(), ecies_key_private_bin(), ecies_key_private_hex(), ecies_key_public(), ecies_key_public_bin(), ecies_key_public_hex(), encrypt_user_messages(), file_load(), file_read(), http_content_load_directory(), http_content_load_fonts(), http_load_file(), http_parse_header(), http_parse_method(), imap_command_log_safe(), imap_id(), linked_insert(), magma_folder_fetch(), mail_load_message(), messages_fetch(), meta_data_fetch_alerts(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_messages(), meta_data_user_build_storage_keys(), meta_data_user_save_storage_keys(), net_listen(), ns_dupe(), nvp_alloc(), pool_alloc(), process_start(), process_stop(), queue_signal(), register_print_captcha(), relay_output_help(), relay_output_settings(), res_bind_create(), res_field_count(), res_field_generic(), res_field_length(), res_field_string(), res_row_count(), res_row_get(), res_row_set(), res_row_store(), res_stmt_store(), res_table_alloc(), res_table_free(), scramble_decrypt(), servers_output_help(), servers_output_settings(), signal_refresh(), signal_shutdown(), signal_start(), signal_status(), signal_thread_start(), spool_check(), sql_query(), stats_get_name_pos(), stmt_exec(), stmt_exec_affected(), stmt_exec_affected_conn(), stmt_exec_conn(), stmt_get_result(), stmt_get_result_conn(), stmt_insert(), stmt_insert_conn(), system_change_root_directory(), system_fork_daemon(), system_init_core_dumps(), system_init_impersonation(), system_init_resource_limits(), system_init_umask(), system_limit_cur(), system_limit_max(), tran_commit(), tran_rollback(), tran_start(), tree_insert(), user_config_fetch(), virus_engine_refresh(), and warehouse_fetch_domains().

#define log_options(options, ...) log_internal (__FILE__, __FUNCTION__, __LINE__, options, __VA_ARGS__)

Used to override the globally configured log options for a specific entry.

Definition at line 90 of file log.h.

Referenced by cache_output_help(), cmp_mt_mt(), config_output_help(), http_print_500(), http_print_500_log(), ident_mt_mt(), inx_alloc(), mm_sec_alloc(), mm_sec_free(), mt_get_char(), mt_get_length(), mt_get_number(), mt_get_type(), mt_is_empty(), mt_is_number(), mt_set_type(), mutex_lock(), mutex_unlock(), nvp_parse(), portal_config_collection(), portal_endpoint_error(), relay_output_help(), servers_output_help(), signal_segfault(), tok_get_count_bl(), and tok_get_ns().

#define log_pedantic(...)

Definition at line 67 of file log.h.

Referenced by adjust_message_encryption(), alert_alloc(), alias_alloc(), ar_alloc(), ar_append(), ar_avail_get(), ar_field_ar(), ar_field_ns(), ar_field_pl(), ar_field_ptr(), ar_field_st(), ar_field_type(), ar_free(), ar_length_get(), ar_length_set(), base64_decode(), base64_decode_mod(), base64_decode_opts(), base64_encode(), base64_encode_mod(), base64_encode_opts(), bracket_extract_pl(), cache_free(), cache_increment(), cache_output_settings(), cache_validate(), cipher_block_length(), cipher_id(), cipher_key_length(), cipher_name(), cipher_numeric_id(), cipher_vector_length(), client_connect(), client_write(), cmp_mt_mt(), compress_import(), con_addr(), con_write_bl(), config_free(), config_output_value(), config_output_value_generic(), contact_alloc(), contact_create(), contact_delete(), contact_detail_alloc(), contact_detail_delete(), contact_detail_upsert(), contact_details_fetch(), contact_edit(), contact_find_number(), contact_folder_create(), contact_folder_remove(), contact_folder_rename(), contact_move(), contact_remove(), contact_update(), contact_update_stamp(), contacts_fetch(), contacts_update(), cryptex_alloc(), debug_hook(), decrypt_user_messages(), derived_value(), deserialize_int16(), deserialize_int32(), deserialize_int64(), deserialize_ns(), deserialize_st(), deserialize_uint16(), deserialize_uint32(), deserialize_uint64(), digest_hash(), digest_id(), digest_name(), dkim_check(), dkim_create(), dkim_start(), domain_alloc(), double_conv(), dspam_check(), dspam_train(), encrypt_user_messages(), engine_compress(), engine_decompress(), errno_name(), file_read(), float_conv(), get_temp_file_handle(), hashed_bucket_get_ptr(), hashed_bucket_set_ptr(), hashed_cursor_alloc(), hex_decode_chr(), hex_decode_st(), hex_encode_st(), host_platform(), host_version(), http_body(), http_content_load_fonts(), http_content_start(), http_data_value_parse(), http_load_file(), http_page_get(), http_print_301(), http_process(), http_response_status(), ident_mt_mt(), imap_append(), imap_append_message(), imap_build_array(), imap_command_parser(), imap_copy(), imap_count_folder_levels(), imap_fetch_message(), imap_folder_create(), imap_folder_remove(),

imap_folder_rename(), imap_folder_status(), imap_get_ar_ar(), imap_get_ptr(), imap_get_st_ar(),
imap_get_type_ar(), imap_list(), imap_login(), imap_message_copier(), imap_narrow_folders(),
imap_narrow_messages(), imap_parse_array(), imap_parse_astring(), imap_parse_literal(),
imap_parse_qstring(), imap_range_build(), imap_search_messages_inner(), imap_starttls(),
imap_valid_folder_name(), int16_conv_bl(), int32_conv_bl(), int64_conv_bl(), int8_conv_bl(), inx_count(),
inx_delete(), inx_find(), inx_free(), inx_insert(), inx_options(), inx_replace(), inx_serial(), inx_truncate(),
ip_presentation(), ip_reversed(), ip_standard(), ip_subnet(), lib_load(), lib_load_bzip(), line_pl_bl(),
linked_cursor_alloc(), linked_record_free(), linked_record_get_data(), linked_record_get_key(), lock_get(),
lock_release(), lower_st(), magma_folder_alloc(), magma_folder_fetch(), magma_folder_name(),
mail_add_forward_headers(), mail_add_inbound_headers(), mail_add_outbound_headers(),
mail_add_required_headers(), mail_build_signature(), mail_cache_set(), mail_cache_start(), mail_cache_stop(),
mail_copy_message(), mail_count_received(), mail_create_message(), mail_db_insert_duplicate_message(),
mail_db_insert_message(), mail_db_update_message_folder(), mail_extract_address(), mail_extract_tag(),
mail_get_boundary(), mail_get_chunk(), mail_header_end(), mail_headers(), mail_insert_chunk_base64(),
mail_insert_chunk_text(), mail_load_header(), mail_load_message(), mail_message(),
mail_message_cleanup(), mail_message_path(), mail_mime_boundary(), mail_mime_child(),
mail_mime_count(), mail_mime_encode_part(), mail_mime_generate_boundary(),
mail_mime_get_smtp_envelope(), mail_mime_part(), mail_mime_split(), mail_mod_subject(),
mail_modify_part(), mail_move_message(), mail_remove_message(), mail_signature_add(),
mail_store_message(), message_alloc(), message_folder_create(), message_folder_remove(), messages_fetch(),
messages_update(), meta_data_acknowledge_alert(), meta_data_check_mailbox(), meta_data_fetch_alerts(),
meta_data_fetch_all_tags(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(),
meta_data_fetch_messages(), meta_data_flags_add(), meta_data_flags_remove(), meta_data_flags_replace(),
meta_data_update_lock(), meta_data_update_log(), meta_data_user_build(),
meta_data_user_build_storage_keys(), meta_folder_stats_tag_alloc(), meta_folders_name(),
meta_folders_stats_tags(), meta_get(), meta_messages_copier(), meta_messages_mover(), meta_user_create(),
mm_alloc(), mm_dupe(), mm_free(), mm_sec_alloc(), mm_sec_realloc(), mm_sec_start(), mm_wipe(),
mt_dupe(), mutex_destroy(), mutex_init(), net_set_buffer_length(), net_set_keepalive(), net_set_linger(),
net_set_nodelay(), net_set_non_blocking(), net_set_reuseable_address(), net_set_timeout(), ns_alloc(),
ns_append(), ns_dupe(), ns_free(), ns_import(), ns_length_int(), ns_wipe(), pool_get_obj(), pool_get_status(),
pool_set_obj(), pool_set_status(), pop_num_parse(), pop_pass(), pop_pass_parse(), pop_starttls(),
pop_top_parse(), pop_user_parse(), portal_config_entry(), portal_contact_details(), portal_endpoint(),
portal_endpoint_alert_acknowledge(), portal_endpoint_alert_list(), portal_endpoint_aliases(),
portal_endpoint_attachments_add(), portal_endpoint_attachments_remove(), portal_endpoint_auth(),
portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(),
portal_endpoint_contacts_list(), portal_endpoint_contacts_load(), portal_endpoint_contacts_move(),
portal_endpoint_contacts_remove(), portal_endpoint_error(), portal_endpoint_folders_add(),
portal_endpoint_folders_list(), portal_endpoint_folders_remove(), portal_endpoint_folders_rename(),
portal_endpoint_folders_tags(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(),
portal_endpoint_messages_list(), portal_endpoint_messages_load(), portal_endpoint_messages_move(),
portal_endpoint_messages_remove(), portal_endpoint_messages_send(), portal_endpoint_messages_tag(),
portal_endpoint_messages_tags(), portal_endpoint_response(), portal_get_upload_attachment(),
portal_message_attachments(), portal_message_body(), portal_message_header(), portal_message_info(),
portal_message_meta(), portal_message_security(), portal_message_server(), portal_message_source(),
portal_meta(), portal_parse_json_str_array(), portal_settings_changepass(), portal_settings_identity(),
portal_smtp_create_data(), portal_upload(), portal_validate_request(), process_kill(), process_stop(),
protocol_enqueue(), protocol_process(), protocol_secure(), qp_decode(), qp_encode(), rand_choices(),
rand_get_int16(), rand_get_int32(), rand_get_int64(), rand_get_int8(), rand_get_uint16(), rand_get_uint32(),
rand_get_uint64(), rand_get_uint8(), rand_write(), register_business_step1(), register_business_step2(),
register_captcha_generate(), register_captcha_random_font(), register_data_fetch_blocklist(),
register_data_insert_user(), register_process(), register_session_cache(), register_session_generate(),
register_session_get(), relay_free(), relay_output_settings(), relay_validate(), rwlock_attr_destroy(),
rwlock_attr_getkind(), rwlock_attr_init(), rwlock_attr_setkind(), rwlock_destroy(), rwlock_init(),
rwlock_lock_read(), rwlock_lock_write(), rwlock_unlock(), scramble_decrypt(), scramble_encrypt(),
scramble_import(), serial_get(), serial_increment(), serial_prefix(), serial_reset(), servers_free(),
servers_output_settings(), servers_validate(), sess_create(), sess_get(), sess_key(), sess_token(), signal_name(),

```

smtp_accept_message(), smtp_add_bypass_entry(), smtp_add_recipient(), smtp_bounce(), smtp_check_filters(),
smtp_check_greylis(),  smtp_check_rbl(),  smtp_check_receive_quota(),  smtp_check_transmit_quota(),
smtp_client_connect(),  smtp_client_send_data(),  smtp_client_send_helo(),  smtp_client_send_mailfrom(),
smtp_client_send_nullfrom(),  smtp_client_send_rcptto(),  smtp_data_read(),  smtp_fetch_authorization(),
smtp_fetch_autoreply(),  smtp_fetch_inbound(),  smtp_fetch_rollmessages(),  smtp_forward_message(),
smtp_insert_spamsig(),  smtp_parse_auth(),  smtp_parse_helo_domain(),  smtp_parse_mail_from_path(),
smtp_parse_rcpt_to(),  smtp_relay_message(),  smtp_reply(),  smtp_rollout(),  smtp_send_message(),
smtp_starttls(),  smtp_store_message(),  smtp_store_spamsig(),  smtp_update_receive_stats(),
smtp_update_transmission_stats(), spf_check(), spf_start(), spool_cleanup(), spool_mktemp(), spool_stop(),
sql_query_conn(), ssl_alloc(), ssl_client_create(), ssl_dh_exchange_callback(), ssl_error_string(), ssl_free(),
ssl_read(),  ssl_server_destroy(),  ssl_verify_privkey(),  ssl_write(),  st_alloc_opts(),  st_append_opts(),
st_avail_get(),  st_avail_set(),  st_copy_in(),  st_data_get(),  st_data_set(),  st_dupe_opts(),  st_free(),
st_length_get(),  st_length_int(),  st_length_set(),  st_merge_opts(),  st_opt_get(),  st_opt_set(),  st_output(),
st_realloc(),  st_replace(),  st_search_chr(),  st_search_ci(),  st_search_cs(),  st_swap(),  st_vaprint_opts(),
st_vsprint(), st_wipe(), stacker_alloc(), stacker_push(), stamp_counter_check(), stamp_counter_increment(),
statistics_process(),  statistics_refresh(),  stmt_bind_param(),  stmt_reset(),  symmetric_decrypt(),
symmetric_encrypt(),  symmetric_vector(),  system_limit_cur(),  system_limit_max(),  tank_count(),
tank_delete_object(),  tank_load(),  teacher_add_cookie(),  teacher_data_delete(),  teacher_data_get(),
teacher_data_save(),  thread_alloc(),  thread_cancel(),  thread_join(),  thread_launch(),  thread_result(),
time_print_gmt(), time_print_local(), tkey_init(), tkey_set(), tree_alloc(), tree_cursor_alloc(), uint16_conv_bl(),
uint32_conv_bl(),  uint64_conv_bl(),  uint8_conv_bl(),  upper_st(),  url_decode(),  url_encode(),
user_config_alloc(), user_config_create(), user_config_delete(), user_config_edit(), user_config_entry_alloc(),
user_config_fetch(),  user_config_upsert(),  virus_check(),  virus_start(),  warehouse_fetch_domains(),
warehouse_fetch_patterns(),  xml_create_doc(),  xml_create_parser_ctx(),  xml_create_xpath_ctx(),
xml_free_doc(), xml_free_parser_ctx(), xml_free_xpath_ctx(), xml_free_xpath_obj(), xml_get_xpath_int16(),
xml_get_xpath_int32(),  xml_get_xpath_int64(),  xml_get_xpath_int8(),  xml_get_xpath_node_count(),
xml_get_xpath_ns(),  xml_get_xpath_st(),  xml_get_xpath_uint16(),  xml_get_xpath_uint32(),
xml_get_xpath_uint64(),  xml_get_xpath_uint8(),  xml_node_get_content_st(),  xml_set_xpath_ns(),
xml_set_xpath_property(),  xml_set_xpath_uint64(),  xml_xpath_eval(),  zbase32_decode(),  and
zbase32_encode().

```

```

#define MAGMA_LOG_LEVELS (M_LOG_PEDANTIC | M_LOG_INFO | M_LOG_INFO |
M_LOG_CRITICAL)

```

Definition at line 42 of file log.h.

Enumeration Type Documentation

enum M_LOG_OPTIONS

Options used to control the behavior of the log subsystem.

Enumerator:

```

M_LOG_PEDANTIC
M_LOG_INFO
M_LOG_ERROR
M_LOG_CRITICAL
M_LOG_TIME
M_LOG_FILE
M_LOG_LINE
M_LOG_FUNCTION
M_LOG_STACK_TRACE
M_LOG_PEDANTIC_DISABLE
M_LOG_INFO_DISABLE

```

M_LOG_ERROR_DISABLE
M_LOG_CRITICAL_DISABLE
M_LOG_LINE_FEED_DISABLE
M_LOG_TIME_DISABLE
M_LOG_FILE_DISABLE
M_LOG_LINE_DISABLE
M_LOG_FUNCTION_DISABLE
M_LOG_STACK_TRACE_DISABLE

Definition at line 19 of file log.h.

Function Documentation

void debug_hook (void)

A stub routine where it is convenient to set a breakpoint in a debugger.

Note:

This function should be called by code paths that detect sanity check failures, but that are not fatal.

Returns:

This function returns no value.

Definition at line 272 of file log.c.

References log_pedantic.

Referenced by base64_encode_mod(), http_print_500(), res_field_generic(), st_data_get(), and st_free().

void void log_disable (void)

Disable logging.

Returns:

This function returns no value.

Definition at line 23 of file log.c.

References log_enabled, log_mutex, mutex_lock(), and mutex_unlock().

void log_enable (void)

Enable logging.

Returns:

This function returns no value.

Definition at line 34 of file log.c.

References log_enabled, log_mutex, mutex_lock(), and mutex_unlock().

void log_internal (const char * *file*, const char * *function*, const int *line*, M_LOG_OPTIONS *options*, const char * *format*, ...)

void log_rotate (void)

Definition at line 184 of file log.c.

References magma_t::file, log_critical, log_date, log_mutex, magma, MEMORYBUF, ns_length_get(), magma_t::output, magma_t::path, and time_datestamp().

Referenced by process_maint().

bool_t log_start (void)

Definition at line 226 of file log.c.

References magma_t::file, folder_exists(), log_critical, log_date, magma, MEMORYBUF, ns_length_get(), NULLER, magma_t::output, magma_t::path, and time_datestamp().

Referenced by process_start().

int_t print_backtrace ()

log.c

log.c

Note:

This function was created because backtrace_symbols() can fail due to heap corruption.

Returns:

-1 if the backtrace failed, or 0 on success.

Definition at line 46 of file log.c.

Referenced by log_internal().

magma/core/memory/align.c File Reference

Functions for memory alignment operations.

#include "magma.h"

Functions

- `size_t align` (`size_t alignment`, `size_t length`)

align.c

Detailed Description

Functions for memory alignment operations.

Definition in file **align.c**.

Function Documentation

size_t align (`size_t alignment`, `size_t length`)

align.c

Definition at line 21 of file align.c.

Referenced by `alert_alloc()`, `alias_alloc()`, `contact_alloc()`, `contact_detail_alloc()`, `domain_alloc()`, `magma_folder_alloc()`, `message_alloc()`, `meta_folder_stats_tag_alloc()`, `mm_sec_alloc()`, `st_alloc_opts()`, `user_config_alloc()`, and `user_config_entry_alloc()`.

magma/core/memory/bits.c File Reference

A collection of functions used handle common bit manipulation tasks.

#include "magma.h"

Functions

- **uint_t bits_count** (uint64_t value)

Count the number of bits that are set in a 64-bit number.

Detailed Description

A collection of functions used handle common bit manipulation tasks.

Definition in file **bits.c**.

Function Documentation

uint_t bits_count (uint64_t value)

Count the number of bits that are set in a 64-bit number.

Parameters:

value an unsigned 64-bit value to be checked.

Returns:

the number of bits in value that are set.

Definition at line 20 of file bits.c.

Referenced by hashed_bucket(), and st_valid_opts().

magma/core/memory/memory.c File Reference

The functions used to handle Magma memory buffers.

```
#include "magma.h"
```

Functions

- void **mm_cleanup** (void *block)
- *Performed a checked memory free.* **bool_t mm_empty** (void *block, size_t len)
- *Determine whether the memory buffer and/or its length encompass an empty block.* void * **mm_copy** (void *dst, const void *src, size_t len)
- *Copy data from one buffer to another.* void * **mm_move** (void *dst, void *src, size_t len)
- *Copy the contents of one buffer into another one, allowing for overlapping data.* void * **mm_set** (void *block, **int_t** set, size_t len)
- *Sets a block of memory to a specified value.* void * **mm_wipe** (void *block, size_t len)
- *Zero out a block of memory.* void **mm_free** (void *block)
- *Free a block of memory.* void * **mm_dupe** (void *block, size_t len)
- *Duplicate a block of memory.* void * **mm_alloc** (size_t len)

Allocate a chunk of memory with the system allocator and zero-wipe it.

Detailed Description

The functions used to handle Magma memory buffers.

Definition in file **memory.c**.

Function Documentation

void* mm_alloc (size_t len)

Allocate a chunk of memory with the system allocator and zero-wipe it.

memory.c

Note:

Uses the 'malloc' function attribute to indicate any non-NULL return value is not an alias for any other valid pointer.

The buffer length must be non-zero.

See also:

<http://gcc.gnu.org/onlinedocs/gcc-4.4.4/gcc/Function-Attributes.html>

Parameters:

len the amount of memory to allocate.

Returns:

a valid pointer to the allocated memory on success, or NULL on error.

Definition at line 181 of file memory.c.

References `log_pedantic`, and `mm_set()`.

Referenced by `alert_alloc()`, `alias_alloc()`, `cache_alloc()`, `client_connect()`, `compress_alloc()`, `compress_lzo()`, `con_addr()`, `con_init()`, `con_print()`, `contact_alloc()`, `contact_detail_alloc()`, `credential_alloc_auth()`,

credential_alloc_mail(), cryptex_alloc(), dkim_memory_alloc(), domain_alloc(), ecies_decrypt(), ecies_key_public_bin(), hashed_alloc(), hashed_bucket_alloc(), hashed_cursor_alloc(), http_data_header_parse_line(), http_data_value_parse(), http_load_file(), http_page_get(), imap_append_message(), imap_copy(), imap_fetch_response_add(), imap_folder_create(), imap_folder_rename(), imap_parse_dataitems(), linked_alloc(), linked_cursor_alloc(), linked_insert(), linked_record_alloc(), magma_folder_alloc(), mail_cache_set(), mail_create_message(), mail_load_header(), mail_message(), mail_mime_part(), message_alloc(), meta_data_fetch_folders(), meta_data_fetch_messages(), meta_folder_stats_tag_alloc(), meta_user_create(), mm_dupe(), ns_alloc(), ns_dupe(), ns_import(), nvp_alloc(), pool_alloc(), portal_endpoint_attachments_add(), portal_endpoint_messages_compose(), process_start(), queue_init(), register_session_generate(), register_session_get(), relay_alloc(), requeue(), res_bind_create(), res_row_store(), res_table_alloc(), scramble_alloc(), servers_alloc(), sess_create(), smtp_add_bypass_entry(), smtp_add_recipient(), smtp_fetch_authorization(), smtp_fetch_inbound(), ssl_print(), ssl_start(), st_alloc_opts(), st_realloc(), stacker_alloc(), stacker_push(), stmt_start(), system_init_impersonation(), tank_start(), tank_store(), teacher_data_fetch(), teacher_data_get(), thread_alloc(), tree_alloc(), tree_cursor_alloc(), user_config_alloc(), and user_config_entry_alloc().

void mm_cleanup (void * *block*)

Performed a checked memory free.

See also:

mm_cleup

Parameters:

block the block of memory to be freed.

Returns:

This function returns no value.

Definition at line 21 of file memory.c.

References mm_free().

Referenced by contact_detail_free(), contact_folder_alloc(), contact_print_form(), imap_fetch_free_items(), message_folder_alloc(), message_free(), queue_shutdown(), and res_bind_free().

void* mm_copy (void * *dst*, const void * *src*, size_t *len*)

Copy data from one buffer to another.

Note:

For overlapping data regions, bl_move() must be used.

Warning:

This function performs an aligned copy so data directly after the src buffer may end up inside the dst buffer.

Parameters:

dst the destination buffer of the copy operation.

src the source buffer of the copy operation.

len the length, in bytes, of the data to be copied.

Returns:

a pointer to the destination buffer.

Definition at line 55 of file memory.c.

Referenced by alert_alloc(), alias_alloc(), con_addr(), contact_alloc(), contact_detail_alloc(), deserialize_ns(), deserialize_st(), domain_alloc(), imap_folder_create(), imap_folder_rename(), imap_parse_literal(), ip_copy(), magma_folder_alloc(), message_alloc(), meta_data_fetch_folders(), meta_data_fetch_messages(), meta_data_user_save_storage_keys(), meta_folder_stats_tag_alloc(), mm_dupe(), mm_sec_realloc(), ns_append(), ns_dupe(), ns_import(), res_row_store(), scramble_encrypt(), serialize_ns(), serialize_st(), signal_shutdown(), smtp_auth_plain(), st_append_opts(), st_copy_in(), st_dupe_opts(), st_import(), st_merge_opts(), st_nullify(), st_realloc(), tank_load(), tank_store(), tree_cursor_next(), tree_delete(), and user_config_entry_alloc().

void* mm_dupe (void * *block*, size_t *len*)

Duplicate a block of memory.

Parameters:

block a pointer to the block of memory to be duplicated.
len the length, in bytes, of the buffer to be duplicated.

Returns:

a freshly allocated buffer containing a copy of the input data.

Definition at line 152 of file memory.c.

References log_pedantic, mm_alloc(), and mm_copy().

Referenced by contact_name(), imap_command_log_safe(), imap_narrow_folders(), and meta_message_dupe().

bool_t mm_empty (void * *block*, size_t *len*)

Determine whether the memory buffer and/or its length encompass an empty block.

Parameters:

block a pointer to the block of memory to be assessed.
len the length, in bytes, of the memory block.

Returns:

false if block is NULL or len is 0; true otherwise.

Definition at line 37 of file memory.c.

Referenced by line_pl_bl(), mm_cmp_ci_eq(), mm_cmp_cs_eq(), str_tok_get_bl(), str_tok_get_count_bl(), tok_get_count_bl(), and tok_get_ns().

void mm_free (void * *block*)

Free a block of memory.

Note:

block can point to either a secure or insecure memory block.

Parameters:

block a pointer to the block to be freed.

Returns:

This function does not return any value.

Definition at line 128 of file memory.c.

References log_pedantic, and mm_sec_secured().

Referenced by ar_append(), ar_free(), cache_free(), cache_get(), cache_get_u64(), client_close(), client_connect(), compress_free(), compress_lzo(), con_addr(), con_destroy(), con_init(), con_print(), config_free(), contact_alloc(), contact_free(), contact_name(), credential_free(), dequeue(), dkim_memory_free(), ecies_key_public_bin(), hashed_alloc(), hashed_cursor_free(), hashed_delete(), hashed_free(), hashed_truncate(), http_data_free(), http_free_content(), http_page_free(), imap_append_message(), imap_command_log_safe(), imap_copy(), imap_fetch_free_items(), imap_fetch_response_add(), imap_fetch_response_free(), imap_folder_create(), imap_folder_rename(), imap_message_copier(), imap_narrow_folders(), inx_free(), linked_cursor_free(), linked_delete(), linked_insert(), linked_record_free(), linked_truncate(), magma_folder_alloc(), magma_folder_free(), mail_cache_destroy(), mail_cache_set(), mail_create_message(), mail_destroy(), mail_destroy_message(), mail_load_message(), mail_message(), mail_mime_free(), message_alloc(), meta_data_fetch_alerts(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_messages(), meta_data_user_build_storage_keys(), meta_folders_stats_tags(), meta_message_free(), meta_messages_copier(), meta_user_create(), meta_user_destroy(), mm_cleanup(), ns_free(), nvp_alloc(), nvp_free(), pool_alloc(), pool_free(), process_start(), process_stop(), protocol_secure(), register_session_free(), relay_free(), res_bind_free(), res_table_free(), scramble_free(), servers_free(), sess_create(), sess_destroy(), sess_release_attachment(), sess_release_composition(), smtp_add_bypass_entry(), smtp_add_recipient(), smtp_fetch_authorization(), smtp_fetch_inbound(), smtp_free_inbound(), smtp_free_outbound(), smtp_free_recipients(), smtp_list_free_filter(), ssl_print(), ssl_stop(), st_alloc_opts(), st_data_set(), st_free(), st_realloc(), stacker_alloc(), stacker_free(), stacker_pop(), stmt_stop(), system_init_impersonation(), tank_stop(), tank_store(), teacher_data_free(), thread_alloc(), tree_alloc(), tree_cursor_free(), user_config_alloc(), user_config_entry_free(), user_config_free(), warehouse_fetch_domains(), and xml_dump_doc().

void* mm_move (void * *dst*, void * *src*, size_t *len*)

Copy the contents of one buffer into another one, allowing for overlapping data.

See also:

memmove()

Parameters:

dst a pointer to the destination buffer to receive the data to be copied.

src a pointer to the source buffer supplying the data to be copied.

len the length, in bytes, of the data buffer to be copied.

Returns:

a pointer to the specified destination buffer.

Definition at line 68 of file memory.c.

Referenced by ar_append(), client_read(), client_read_line(), con_read(), con_read_line(), imap_parse_literal(), and st_trim().

void* mm_set (void * *block*, int_t *set*, size_t *len*)

Sets a block of memory to a specified value.

Note:

Uses the 'optimize (0)' and 'noinline' function attributes to prevent compiler optimization from removing logic it might consider unnecessary.

Parameters:

block the block of memory to be set.
set the byte value to be written to block.
len the number of times to write the value of the byte repeatedly to block.

Returns:

a pointer to the block of memory passed to the function.

See also:

<http://gcc.gnu.org/onlinedocs/gcc-4.4.4/gcc/Function-Attributes.html>

Definition at line 84 of file memory.c.

Referenced by mm_alloc(), mm_sec_free(), mm_sec_stop(), mm_wipe(), ns_wipe(), and st_alloc_opts().

void* mm_wipe (void * *block*, size_t *len*)

Zero out a block of memory.

Note:

Uses the 'optimize (0)' and 'noinline' function attributes to prevent compiler optimization from removing logic it might consider unnecessary.

Parameters:

block the block of memory to be zeroed.
len the number of zero bytes to write to memory.

See also:

<http://gcc.gnu.org/onlinedocs/gcc-4.4.4/gcc/Function-Attributes.html>

Definition at line 106 of file memory.c.

References log_pedantic, and mm_set().

Referenced by ar_alloc(), config_fetch_host_number(), config_fetch_settings(), contact_delete(), contact_detail_delete(), contact_detail_upsert(), contact_details_fetch(), contact_insert(), contact_update(), contact_update_stamp(), contacts_fetch(), ecies_key_private_hex(), hex_encode_chr(), http_response_header(), http_response_options(), imap_folder_rename(), imap_folder_status(), magma_folder_delete(), magma_folder_fetch(), magma_folder_insert(), magma_folder_rename(), mail_db_delete_message(), mail_db_hide_message(), mail_db_insert_duplicate_message(), mail_db_insert_message(), mail_db_update_message_folder(), messages_fetch(), meta_data_acknowledge_alert(), meta_data_check_mailbox(), meta_data_delete_folder(), meta_data_delete_tag(), meta_data_fetch_alerts(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_message_tags(), meta_data_fetch_messages(), meta_data_fetch_user(), meta_data_flags_add(), meta_data_flags_remove(), meta_data_flags_replace(), meta_data_insert_folder(), meta_data_insert_tag(), meta_data_truncate_tags(), meta_data_update_folder_name(), meta_data_update_lock(), meta_data_update_log(), meta_data_user_build(), meta_data_user_build_storage_keys(), meta_data_user_save_storage_keys(), mm_sec_alloc(), mm_sec_start(), net_init(), net_listen(), net_set_timeout(), portal_message_header(), register_captcha_generate(), register_data_check_username(), register_data_insert_user(), register_session_get(), serv_charset_mysql(), serv_schema_mysql(), serv_version_mysql(), signal_segfault(), signal_shutdown(), signal_start(), signal_thread_start(), smtp_check_authorized_from(), smtp_check_filters(), smtp_check_rbl(), smtp_check_receive_quota(), smtp_check_transmit_quota(), smtp_fetch_authorization(), smtp_fetch_autoreply(), smtp_fetch_inbound(), smtp_fetch_rollmessages(), smtp_insert_spamsig(), smtp_update_receive_stats(), smtp_update_transmission_stats(), st_trim(), st_wipe(), statistics_init(), stats_init(), stmt_start(), system_init_impersonation(), tank_delete_object(), tank_insert_object(), tank_store(), teacher_data_delete(), teacher_data_fetch(), teacher_data_get(), time_print_gmt(), time_print_local(), user_config_delete(), user_config_fetch(), user_config_upsert(), virus_engine_refresh(), and virus_start().

magma/core/memory/memory.h File Reference

The functions used to allocate and manipulate blocks of memory off the heap and inside the secured address space.

Defines

- `#define MM_SEC_REQUEST_ALIGNMENT 12`
- `#define MM_SEC_PAGE_ALIGNMENT_MIN 1024`
- `#define MM_SEC_POOL_LENGTH_MIN 4096`
- `#define MEMORYBUF(l) (void *)&((chr_t []){ [0 ... l] = 0 })`

Functions

- `size_t align` (`size_t alignment, size_t len`)
- *align.c* `uint_t bits_count` (`uint64_t value`)
- *Count the number of bits that are set in a 64-bit number.* `void mm_sec_stop` (`void`)
- *Deallocate and perform a multi-stage secure wipe of the secure memory region.* `bool_t mm_sec_start` (`void`)
- *If enabled, allocate and initialize the secure memory slab.* `void mm_sec_free` (`void *block`)
- *Free a secure memory block and perform a multi-pass wipe of its contents.* `bool_t mm_sec_secured` (`void *block`)
- *Determine whether the data pointer falls within the secure memory block.* `void * mm_sec_alloc` (`size_t len`)
- *Allocate a chunk of memory from the secure memory slab.* `void * mm_sec_realloc` (`void *orig, size_t len`)
- `bool_t mm_sec_stats` (`size_t *total, size_t *bytes, size_t *items`) `__attribute__((nonnull(1`
- `bool_t void * mm_alloc` (`size_t len`)
- *memory.c* `void mm_cleanup` (`void *block`)
- *Performed a checked memory free.* `void * mm_copy` (`void *dst, const void *src, size_t len`)
- *Copy data from one buffer to another.* `void * mm_dupe` (`void *block, size_t len`)
- *Duplicate a block of memory.* `bool_t mm_empty` (`void *block, size_t len`)
- *Determine whether the memory buffer and/or its length encompass an empty block.* `void mm_free` (`void *block`)
- *Free a block of memory.* `void * mm_move` (`void *dst, void *src, size_t len`)
- *Copy the contents of one buffer into another one, allowing for overlapping data.* `void * mm_set` (`void *block, int_t set, size_t len`)
- *Sets a block of memory to a specified value.* `void * mm_wipe` (`void *block, size_t len`)

Zero out a block of memory.

Detailed Description

The functions used to allocate and manipulate blocks of memory off the heap and inside the secured address space.

Definition in file **memory.h**.

Define Documentation

#define MEMORYBUF(l) (void *)&((chr_t []){ [0 ... l] = 0 })

Definition at line 52 of file **memory.h**.

Referenced by client_connect(), file_read(), http_content_load_directory(), http_content_load_fonts(), http_load_file(), log_rotate(), log_start(), mutex_destroy(), mutex_init(), mutex_lock(), mutex_unlock(), process_kill(), sess_create(), sess_get(), smtp_rcpt_to(), spool_mktemp(), ssl_client_create(), ssl_verify_privkey(), st_alloc_opts(), st_avail_get(), st_avail_set(), st_data_get(), st_data_set(), st_dupe_opts(), st_free(), st_length_get(), st_length_set(), st_merge_opts(), st_opt_get(), st_opt_set(), st_realloc(), st_vaprint_opts(), st_vsprint(), and st_wipe().

#define MM_SEC_PAGE_ALIGNMENT_MIN 1024

Definition at line 46 of file memory.h.

Referenced by mm_sec_start().

#define MM_SEC_POOL_LENGTH_MIN 4096

Definition at line 49 of file memory.h.

Referenced by mm_sec_start().

#define MM_SEC_REQUEST_ALIGNMENT 12

Definition at line 43 of file memory.h.

Function Documentation

size_t align (size_t *alignment*, size_t *len*)

align.c

Definition at line 21 of file align.c.

Referenced by alert_alloc(), alias_alloc(), contact_alloc(), contact_detail_alloc(), domain_alloc(), magma_folder_alloc(), message_alloc(), meta_folder_stats_tag_alloc(), mm_sec_alloc(), st_alloc_opts(), user_config_alloc(), and user_config_entry_alloc().

uint_t bits_count (uint64_t *value*)

Count the number of bits that are set in a 64-bit number.

Parameters:

value an unsigned 64-bit value to be checked.

Returns:

the number of bits in *value* that are set.

Definition at line 20 of file bits.c.

Referenced by hashed_bucket(), and st_valid_opts().

bool_t void* mm_alloc (size_t *len*)

memory.c

memory.c

Note:

Uses the 'malloc' function attribute to indicate any non-NULL return value is not an alias for any other valid pointer.

The buffer length must be non-zero.

See also:

<http://gcc.gnu.org/onlinedocs/gcc-4.4.4/gcc/Function-Attributes.html>

Parameters:

len the amount of memory to allocate.

Returns:

a valid pointer to the allocated memory on success, or NULL on error.

Definition at line 181 of file memory.c.

References log_pedantic, and mm_set().

Referenced by alert_alloc(), alias_alloc(), cache_alloc(), client_connect(), compress_alloc(), compress_lzo(), con_addr(), con_init(), con_print(), contact_alloc(), contact_detail_alloc(), credential_alloc_auth(), credential_alloc_mail(), cryptex_alloc(), dkim_memory_alloc(), domain_alloc(), ecies_decrypt(), ecies_key_public_bin(), hashed_alloc(), hashed_bucket_alloc(), hashed_cursor_alloc(), http_data_header_parse_line(), http_data_value_parse(), http_load_file(), http_page_get(), imap_append_message(), imap_copy(), imap_fetch_response_add(), imap_folder_create(), imap_folder_rename(), imap_parse_dataitems(), linked_alloc(), linked_cursor_alloc(), linked_insert(), linked_record_alloc(), magma_folder_alloc(), mail_cache_set(), mail_create_message(), mail_load_header(), mail_message(), mail_mime_part(), message_alloc(), meta_data_fetch_folders(), meta_data_fetch_messages(), meta_folder_stats_tag_alloc(), meta_user_create(), mm_dupe(), ns_alloc(), ns_dupe(), ns_import(), nvp_alloc(), pool_alloc(), portal_endpoint_attachments_add(), portal_endpoint_messages_compose(), process_start(), queue_init(), register_session_generate(), register_session_get(), relay_alloc(), requeue(), res_bind_create(), res_row_store(), res_table_alloc(), scramble_alloc(), servers_alloc(), sess_create(), smtp_add_bypass_entry(), smtp_add_recipient(), smtp_fetch_authorization(), smtp_fetch_inbound(), ssl_print(), ssl_start(), st_alloc_opts(), st_realloc(), stacker_alloc(), stacker_push(), stmt_start(), system_init_impersonation(), tank_start(), tank_store(), teacher_data_fetch(), teacher_data_get(), thread_alloc(), tree_alloc(), tree_cursor_alloc(), user_config_alloc(), and user_config_entry_alloc().

void mm_cleanup (void * *block*)

Performed a checked memory free.

See also:

mm_cleanup

Parameters:

block the block of memory to be freed.

Returns:

This function returns no value.

Definition at line 21 of file memory.c.

References mm_free().

Referenced by contact_detail_free(), contact_folder_alloc(), contact_print_form(), imap_fetch_free_items(), message_folder_alloc(), message_free(), queue_shutdown(), and res_bind_free().

void* mm_copy (void * *dst*, const void * *src*, size_t *len*)

Copy data from one buffer to another.

Note:

For overlapping data regions, `bl_move()` must be used.

Warning:

This function performs an aligned copy so data directly after the `src` buffer may end up inside the `dst` buffer.

Parameters:

dst the destination buffer of the copy operation.

src the source buffer of the copy operation.

len the length, in bytes, of the data to be copied.

Returns:

a pointer to the destination buffer.

Definition at line 55 of file `memory.c`.

Referenced by `alert_alloc()`, `alias_alloc()`, `con_addr()`, `contact_alloc()`, `contact_detail_alloc()`, `deserialize_ns()`, `deserialize_st()`, `domain_alloc()`, `imap_folder_create()`, `imap_folder_rename()`, `imap_parse_literal()`, `ip_copy()`, `magma_folder_alloc()`, `message_alloc()`, `meta_data_fetch_folders()`, `meta_data_fetch_messages()`, `meta_data_user_save_storage_keys()`, `meta_folder_stats_tag_alloc()`, `mm_dupe()`, `mm_sec_realloc()`, `ns_append()`, `ns_dupe()`, `ns_import()`, `res_row_store()`, `scramble_encrypt()`, `serialize_ns()`, `serialize_st()`, `signal_shutdown()`, `smtp_auth_plain()`, `st_append_opts()`, `st_copy_in()`, `st_dupe_opts()`, `st_import()`, `st_merge_opts()`, `st_nullify()`, `st_realloc()`, `tank_load()`, `tank_store()`, `tree_cursor_next()`, `tree_delete()`, and `user_config_entry_alloc()`.

void* mm_dupe (void * *block*, size_t *len*)

Duplicate a block of memory.

Parameters:

block a pointer to the block of memory to be duplicated.

len the length, in bytes, of the buffer to be duplicated.

Returns:

a freshly allocated buffer containing a copy of the input data.

Definition at line 152 of file `memory.c`.

References `log_pedantic`, `mm_alloc()`, and `mm_copy()`.

Referenced by `contact_name()`, `imap_command_log_safe()`, `imap_narrow_folders()`, and `meta_message_dupe()`.

bool_t mm_empty (void * *block*, size_t *len*)

Determine whether the memory buffer and/or its length encompass an empty block.

Parameters:

block a pointer to the block of memory to be assessed.

len the length, in bytes, of the memory block.

Returns:

false if block is NULL or len is 0; true otherwise.

Definition at line 37 of file memory.c.

Referenced by line_pl_bl(), mm_cmp_ci_eq(), mm_cmp_cs_eq(), str_tok_get_bl(), str_tok_get_count_bl(), tok_get_count_bl(), and tok_get_ns().

void mm_free (void * *block*)

Free a block of memory.

Note:

block can point to either a secure or insecure memory block.

Parameters:

block a pointer to the block to be freed.

Returns:

This function does not return any value.

Definition at line 128 of file memory.c.

References log_pedantic, and mm_sec_secured().

Referenced by ar_append(), ar_free(), cache_free(), cache_get(), cache_get_u64(), client_close(), client_connect(), compress_free(), compress_lzo(), con_addr(), con_destroy(), con_init(), con_print(), config_free(), contact_alloc(), contact_free(), contact_name(), credential_free(), dequeue(), dkim_memory_free(), ecies_key_public_bin(), hashed_alloc(), hashed_cursor_free(), hashed_delete(), hashed_free(), hashed_truncate(), http_data_free(), http_free_content(), http_page_free(), imap_append_message(), imap_command_log_safe(), imap_copy(), imap_fetch_free_items(), imap_fetch_response_add(), imap_fetch_response_free(), imap_folder_create(), imap_folder_rename(), imap_message_copier(), imap_narrow_folders(), inx_free(), linked_cursor_free(), linked_delete(), linked_insert(), linked_record_free(), linked_truncate(), magma_folder_alloc(), magma_folder_free(), mail_cache_destroy(), mail_cache_set(), mail_create_message(), mail_destroy(), mail_destroy_message(), mail_load_message(), mail_message(), mail_mime_free(), message_alloc(), meta_data_fetch_alerts(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_messages(), meta_data_user_build_storage_keys(), meta_folders_stats_tags(), meta_message_free(), meta_messages_copier(), meta_user_create(), meta_user_destroy(), mm_cleanup(), ns_free(), nvp_alloc(), nvp_free(), pool_alloc(), pool_free(), process_start(), process_stop(), protocol_secure(), register_session_free(), relay_free(), res_bind_free(), res_table_free(), scramble_free(), servers_free(), sess_create(), sess_destroy(), sess_release_attachment(), sess_release_composition(), smtp_add_bypass_entry(), smtp_add_recipient(), smtp_fetch_authorization(), smtp_fetch_inbound(), smtp_free_inbound(), smtp_free_outbound(), smtp_free_recipients(), smtp_list_free_filter(), ssl_print(), ssl_stop(), st_alloc_opts(), st_data_set(), st_free(), st_realloc(), stacker_alloc(), stacker_free(), stacker_pop(), stmt_stop(), system_init_impersonation(), tank_stop(), tank_store(), teacher_data_free(), thread_alloc(), tree_alloc(), tree_cursor_free(), user_config_alloc(), user_config_entry_free(), user_config_free(), warehouse_fetch_domains(), and xml_dump_doc().

void* mm_move (void * *dst*, void * *src*, size_t *len*)

Copy the contents of one buffer into another one, allowing for overlapping data.

See also:

memmove()

Parameters:

dst a pointer to the destination buffer to receive the data to be copied.
src a pointer to the source buffer supplying the data to be copied.
len the length, in bytes, of the data buffer to be copied.

Returns:

a pointer to the specified destination buffer.

Definition at line 68 of file memory.c.

Referenced by ar_append(), client_read(), client_read_line(), con_read(), con_read_line(), imap_parse_literal(), and st_trim().

void* mm_sec_alloc (size_t len)

Allocate a chunk of memory from the secure memory slab.

See also:

mm_sec_chunk_new()

Parameters:

len the length, in bytes, of the secure memory chunk to be allocated.

Returns:

NULL on failure, or a pointer to the freshly allocated chunk of secure memory on success.

Definition at line 238 of file secure.c.

References align(), bytes, items, log_options, log_pedantic, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, mm_sec_chunk_new(), mm_sec_stats(), mm_wipe(), mutex_lock(), and mutex_unlock().

Referenced by ecies_key_private_bin(), mm_sec_realloc(), st_alloc_opts(), and st_realloc().

void mm_sec_free (void * block)

Free a secure memory block and perform a multi-pass wipe of its contents.

Returns:

This function returns no value.

Definition at line 198 of file secure.c.

References log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, MM_SEC_CHUNK_ALLOCATED, mm_sec_chunk_merge(), mm_sec_secured(), mm_set(), mutex_lock(), and mutex_unlock().

Referenced by ecies_key_private_bin(), meta_data_user_build_storage_keys(), mm_sec_realloc(), st_alloc_opts(), st_data_set(), st_free(), and st_realloc().

void* mm_sec_realloc (void * orig, size_t len)

Definition at line 281 of file secure.c.

References log_pedantic, mm_copy(), mm_sec_alloc(), and mm_sec_free().

bool_t mm_sec_secured (void * block)

Determine whether the data pointer falls within the secure memory block.

Parameters:

block the data pointer to be tested.

Returns:

true if block points to secure data; false if not, or if block is invalid or secure memory is disabled.

Definition at line 85 of file secure.c.

References slab.

Referenced by mm_free(), mm_sec_chunk_new(), mm_sec_chunk_next(), mm_sec_free(), and st_data_set().

bool_t mm_sec_start (void)

If enabled, allocate and initialize the secure memory slab.

Note:

This function will mmap a page-aligned secure memory slab (defaults to 32768 bytes long), mlock() it into memory, and zero-wipe it. Guard pages with empty permissions are created on the boundaries of the slab to prevent memory bungling.

Returns:

true if the secure memory slab has been initialized, or false if the process fails.

Definition at line 344 of file secure.c.

References log_pedantic, magma, magma_t::memory, MM_SEC_PAGE_ALIGNMENT_MIN, MM_SEC_POOL_LENGTH_MIN, mm_wipe(), magma_t::page_length, and magma_t::secure.

Referenced by process_start().

bool_t mm_sec_stats (size_t * total, size_t * bytes, size_t * items)

Referenced by derived_value(), and mm_sec_alloc().

void mm_sec_stop (void)

Deallocate and perform a multi-stage secure wipe of the secure memory region.

Returns:

This function returns no value.

Definition at line 313 of file secure.c.

References mm_set().

Referenced by process_stop().

void* mm_set (void * block, int_t set, size_t len)

Sets a block of memory to a specified value.

Note:

Uses the 'optimize (0)' and 'noinline' function attributes to prevent compiler optimization from removing logic it might consider unnecessary.

Parameters:

block the block of memory to be set.

set the byte value to be written to block.

len the number of times to write the value of the byte repeatedly to block.

Returns:

a pointer to the block of memory passed to the function.

See also:

<http://gcc.gnu.org/onlinedocs/gcc-4.4.4/gcc/Function-Attributes.html>

Definition at line 84 of file memory.c.

Referenced by mm_alloc(), mm_sec_free(), mm_sec_stop(), mm_wipe(), ns_wipe(), and st_alloc_opts().

void* mm_wipe (void * *block*, size_t *len*)

Zero out a block of memory.

Note:

Uses the 'optimize (0)' and 'noinline' function attributes to prevent compiler optimization from removing logic it might consider unnecessary.

Parameters:

block the block of memory to be zeroed.

len the number of zero bytes to write to memory.

See also:

<http://gcc.gnu.org/onlinedocs/gcc-4.4.4/gcc/Function-Attributes.html>

Definition at line 106 of file memory.c.

References log_pedantic, and mm_set().

Referenced by ar_alloc(), config_fetch_host_number(), config_fetch_settings(), contact_delete(), contact_detail_delete(), contact_detail_upsert(), contact_details_fetch(), contact_insert(), contact_update(), contact_update_stamp(), contacts_fetch(), ecies_key_private_hex(), hex_encode_chr(), http_response_header(), http_response_options(), imap_folder_rename(), imap_folder_status(), magma_folder_delete(), magma_folder_fetch(), magma_folder_insert(), magma_folder_rename(), mail_db_delete_message(), mail_db_hide_message(), mail_db_insert_duplicate_message(), mail_db_insert_message(), mail_db_update_message_folder(), messages_fetch(), meta_data_acknowledge_alert(), meta_data_check_mailbox(), meta_data_delete_folder(), meta_data_delete_tag(), meta_data_fetch_alerts(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_message_tags(), meta_data_fetch_messages(), meta_data_fetch_user(), meta_data_flags_add(), meta_data_flags_remove(), meta_data_flags_replace(), meta_data_insert_folder(), meta_data_insert_tag(), meta_data_truncate_tags(), meta_data_update_folder_name(), meta_data_update_lock(), meta_data_update_log(), meta_data_user_build(), meta_data_user_build_storage_keys(), meta_data_user_save_storage_keys(), mm_sec_alloc(), mm_sec_start(), net_init(), net_listen(), net_set_timeout(), portal_message_header(), register_captcha_generate(), register_data_check_username(), register_data_insert_user(), register_session_get(), serv_charset_mysql(), serv_schema_mysql(), serv_version_mysql(), signal_segfault(), signal_shutdown(), signal_start(), signal_thread_start(), smtp_check_authorized_from(), smtp_check_filters(), smtp_check_rbl(), smtp_check_receive_quota(), smtp_check_transmit_quota(), smtp_fetch_authorization(), smtp_fetch_autoreply(), smtp_fetch_inbound(), smtp_fetch_rollmessages(), smtp_insert_spamsig(), smtp_update_receive_stats(), smtp_update_transmission_stats(), st_trim(), st_wipe(), statistics_init(), stats_init(), stmt_start(), system_init_impersonation(), tank_delete_object(), tank_insert_object(), tank_store(),

teacher_data_delete(), teacher_data_fetch(), teacher_data_get(), time_print_gmt(), time_print_local(), user_config_delete(), user_config_fetch(), user_config_upsert(), virus_engine_refresh(), and virus_start().

magma/core/memory/secure.c File Reference

Functions for allocating secure memory. Secure buffers should always be used to hold sensitive information.
`#include "magma.h"`

Data Structures

- `struct __attribute__`

Enumerations

- `enum { MM_SEC_CHUNK_AVAILABLE = 0, MM_SEC_CHUNK_ALLOCATED = 1 }`

Functions

- `bool_t mm_sec_stats` (`size_t *total`, `size_t *bytes`, `size_t *items`)
 - *Get the collected secure memory statistics for the caller.* `bool_t mm_sec_secured` (`void *block`)
 - *Determine whether the data pointer falls within the secure memory block.* `secured_t * mm_sec_chunk_next` (`secured_t *chunk`)
 - *Get the next chunk of secure memory.* `secured_t * mm_sec_chunk_prev` (`secured_t *chunk`)
 - *Get the previous chunk of secure memory.* `void mm_sec_chunk_merge` (`secured_t *chunk`)
 - `secured_t * mm_sec_chunk_new` (`secured_t *block`, `size_t size`)
 - `void mm_sec_free` (`void *block`)
 - *Free a secure memory block and perform a multi-pass wipe of its contents.* `void * mm_sec_alloc` (`size_t len`)
 - *Allocate a chunk of memory from the secure memory slab.* `void * mm_sec_realloc` (`void *orig`, `size_t len`)
 - `void mm_sec_stop` (`void`)
 - *Deallocate and perform a multi-stage secure wipe of the secure memory region.* `bool_t mm_sec_start` (`void`)
- If enabled, allocate and initialize the secure memory slab.*
-

Detailed Description

Functions for allocating secure memory. Secure buffers should always be used to hold sensitive information.

Definition in file `secure.c`.

Enumeration Type Documentation

anonymous enum

Enumerator:

`MM_SEC_CHUNK_AVAILABLE`
`MM_SEC_CHUNK_ALLOCATED`

Definition at line 15 of file `secure.c`.

Function Documentation

void* mm_sec_alloc (size_t *len*)

Allocate a chunk of memory from the secure memory slab.

See also:

mm_sec_chunk_new()

Parameters:

len the length, in bytes, of the secure memory chunk to be allocated.

Returns:

NULL on failure, or a pointer to the freshly allocated chunk of secure memory on success.

Definition at line 238 of file secure.c.

References align(), bytes, items, log_options, log_pedantic, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, mm_sec_chunk_new(), mm_sec_stats(), mm_wipe(), mutex_lock(), and mutex_unlock().

Referenced by ecies_key_private_bin(), mm_sec_realloc(), st_alloc_opts(), and st_realloc().

void mm_sec_chunk_merge (secured_t * *chunk*)

Definition at line 139 of file secure.c.

References MM_SEC_CHUNK_ALLOCATED, mm_sec_chunk_next(), and mm_sec_chunk_prev().

Referenced by mm_sec_chunk_new(), and mm_sec_free().

secured_t* mm_sec_chunk_new (secured_t * *block*, size_t *size*)

Definition at line 159 of file secure.c.

References MM_SEC_CHUNK_ALLOCATED, mm_sec_chunk_merge(), mm_sec_chunk_next(), and mm_sec_secured().

Referenced by mm_sec_alloc().

secured_t* mm_sec_chunk_next (secured_t * *chunk*)

Get the next chunk of secure memory.

Parameters:

chunk the input secure chunk.

Returns:

a pointer to the next chunk of secure memory, or NULL on failure or if the end of the slab is reached.

Definition at line 107 of file secure.c.

References mm_sec_secured().

Referenced by mm_sec_chunk_merge(), mm_sec_chunk_new(), and mm_sec_chunk_prev().

secured_t* mm_sec_chunk_prev (secured_t * *chunk*)

Get the previous chunk of secure memory.

Parameters:

chunk the input secure chunk.

Returns:

a pointer to the previous chunk of secure memory, or NULL on failure or if the beginning of the slab is reached.

Definition at line 123 of file secure.c.

References mm_sec_chunk_next().

Referenced by mm_sec_chunk_merge().

void mm_sec_free (void * *block*)

Free a secure memory block and perform a multi-pass wipe of its contents.

Returns:

This function returns no value.

Definition at line 198 of file secure.c.

References log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, MM_SEC_CHUNK_ALLOCATED, mm_sec_chunk_merge(), mm_sec_secured(), mm_set(), mutex_lock(), and mutex_unlock().

Referenced by ecies_key_private_bin(), meta_data_user_build_storage_keys(), mm_sec_realloc(), st_alloc_opts(), st_data_set(), st_free(), and st_realloc().

void* mm_sec_realloc (void * *orig*, size_t *len*)

Definition at line 281 of file secure.c.

References log_pedantic, mm_copy(), mm_sec_alloc(), and mm_sec_free().

bool_t mm_sec_secured (void * *block*)

Determine whether the data pointer falls within the secure memory block.

Parameters:

block the data pointer to be tested.

Returns:

true if block points to secure data; false if not, or if block is invalid or secure memory is disabled.

Definition at line 85 of file secure.c.

References slab.

Referenced by mm_free(), mm_sec_chunk_new(), mm_sec_chunk_next(), mm_sec_free(), and st_data_set().

bool_t mm_sec_start (void)

If enabled, allocate and initialize the secure memory slab.

Note:

This function will mmap a page-aligned secure memory slab (defaults to 32768 bytes long), mlock() it into memory, and zero-wipe it. Guard pages with empty permissions are created on the boundaries of the slab to prevent memory bungling.

Returns:

true if the secure memory slab has been initialized, or false if the process fails.

Definition at line 344 of file secure.c.

References log_pedantic, magma, magma_t::memory, MM_SEC_PAGE_ALIGNMENT_MIN, MM_SEC_POOL_LENGTH_MIN, mm_wipe(), magma_t::page_length, and magma_t::secure.

Referenced by process_start().

bool_t mm_sec_stats (size_t * total, size_t * bytes, size_t * items)

Get the collected secure memory statistics for the caller.

Parameters:

total a pointer to a size_t variable that will store the secure memory region length, in bytes.

bytes a pointer to a size_t variable that will store the number of secure bytes allocated by magma.

items a pointer to a size_t variable that will store the number of secure memory allocations requested by magma.

Returns:

true on success or false on failure.

Definition at line 65 of file secure.c.

References mutex_lock(), and mutex_unlock().

void mm_sec_stop (void)

Deallocate and perform a multi-stage secure wipe of the secure memory region.

Returns:

This function returns no value.

Definition at line 313 of file secure.c.

References mm_set().

Referenced by process_stop().

Variable Documentation

struct { ... } allocated

size_t bytes

Definition at line 37 of file secure.c.

Referenced by client_read(), client_read_line(), con_read(), con_read_line(), derived_value(), deserialize_ns(), and mm_sec_alloc().

void* data

Definition at line 29 of file secure.c.

Referenced by bracket_extract_pl(), cache_get(), cache_get_u64(), contact_name(), contact_print_form(), hash_murmur32(), hash_murmur64(), hashed_bucket_find_key(), hashed_find(), http_body(), http_data_get(), http_data_value_parse(), http_load_file(), http_parse_header(), http_parse_pairs(), int16_conv_bl(), int32_conv_bl(), int64_conv_bl(), int8_conv_bl(), net_listen(), portal_endpoint_messages_load(), portal_upload(), register_business_step1(), register_business_step2(), register_session_cache(), register_session_get(), smtp_check_filters(), st_free(), st_length_get(), st_realloc(), teacher_data_get(), teacher_data_save(), uint16_conv_bl(), uint32_conv_bl(), uint64_conv_bl(), and uint8_conv_bl().

void* data_true

Definition at line 28 of file secure.c.

bool_t enabled

Definition at line 40 of file secure.c.

size_t items

Definition at line 36 of file secure.c.

Referenced by derived_value(), imap_fetch(), imap_fetch_bodystructure(), and mm_sec_alloc().

size_t length

Definition at line 30 of file secure.c.

Referenced by cache_get(), cache_get_u64(), client_write(), con_print(), contact_business_valid_email(), deserialize_int16(), deserialize_int32(), deserialize_int64(), deserialize_st(), deserialize_uint16(), deserialize_uint32(), deserialize_uint64(), http_body(), http_data_value_decode(), imap_build_array(), imap_build_array_isliteral(), imap_command_parser(), imap_fetch_body(), imap_fetch_body_portion(), imap_fetch_bodystructure(), imap_parse_address_breaker(), imap_parse_address_part(), imap_valid_sequence(), int16_digits(), int32_digits(), int64_digits(), int8_digits(), mail_add_required_headers(), mail_build_signature(), mail_discover_insertion_point(), mail_extract_address(), mail_get_boundary(), mail_get_chunk(), mail_header_end(), mail_headers(), mail_insert_chunk_base64(), mail_load_message_top(), mail_message(), mail_message_cleanup(), mail_mime_boundary(), mail_mime_child(), mail_mime_count(), mail_mime_header(), mail_mime_type_parameters_key(),

mail_mime_type_parameters_value(), mail_modify_part(), mail_path_finder(), mail_setup_basic(), molten_stats(), ns_dupe(), pop_num_parse(), pop_pass_parse(), pop_top_parse(), pop_user_parse(), register_business_validate_password(), register_business_validate_username(), res_bind_create(), res_bind_free(), res_field_string(), res_row_store(), serialize_int16(), serialize_int32(), serialize_int64(), serialize_st(), serialize_uint16(), serialize_uint32(), serialize_uint64(), smtp_check_filters(), smtp_client_send_data(), smtp_parse_auth(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(), smtp_parse_rcpt_to(), ssl_print(), st_free(), st_merge_opts(), st_vaprint_opts(), st_vsprint(), stamp_counter_check(), tran_commit(), tran_rollback(), tran_start(), tree_truncate(), uint16_digits(), uint32_digits(), uint64_digits(), and uint8_digits().

size_t length_true

Definition at line 31 of file secure.c.

pthread_mutex_t lock

Definition at line 32 of file secure.c.

Referenced by __attribute__(), lock_get(), and lock_release().

struct { ... } slab

Referenced by mm_sec_secured().

magma/core/parsers/case.c File Reference

A collection of functions used for manipulating the capitalization of characters.

```
#include "magma.h"
```

Functions

- **uchr_t upper_chr (uchr_t c)**
- *Return the uppercase representation of a character.* **uchr_t lower_chr (uchr_t c)**
- *Return the lowercase representation of a character.* **stringer_t * upper_st (stringer_t *s)**
- *Transform a managed string (in-place) into uppercase.* **stringer_t * lower_st (stringer_t *s)**

Transform a managed string (in-place) into lowercase.

Detailed Description

A collection of functions used for manipulating the capitalization of characters.

Definition in file **case.c**.

Function Documentation

uchr_t lower_chr (uchr_t c)

Return the lowercase representation of a character.

Parameters:

c the character to be transformed.

Returns:

the input character, in lowercase.

Definition at line 30 of file case.c.

Referenced by credential_address(), hex_decode_chr(), mm_cmp_ci_eq(), pop_user_parse(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(), smtp_parse_rcpt_to(), st_cmp_ci_ends(), st_cmp_ci_eq(), st_cmp_ci_starts(), and st_search_ci().

stringer_t* lower_st (stringer_t * s)

Transform a managed string (in-place) into lowercase.

Parameters:

s the managed string to be modified.

Returns:

NULL on error, or a pointer to the input managed string, in lowercase.

Definition at line 63 of file case.c.

References log_pedantic, and st_empty_out().

Referenced by `contact_business_valid_email()`, `http_parse_header()`, and `smtp_rcpt_to()`.

`uchar_t upper_chr (uchar_t c)`

Return the uppercase representation of a character.

Parameters:

`c` the character to be transformed.

Returns:

the input character, in uppercase.

Definition at line 20 of file `case.c`.

`stringer_t* upper_st (stringer_t * s)`

Transform a managed string (in-place) into uppercase.

Parameters:

`s` the managed string to be modified.

Returns:

NULL on error, or a pointer to the input managed string, in uppercase.

Definition at line 40 of file `case.c`.

References `log_pedantic`, and `st_empty_out()`.

Referenced by `http_response_allow_cross()`, `imap_fetch_body_tag()`, `imap_fetch_bodystructure()`, and `mail_mime_type_parameters()`.

magma/core/parsers/formats/formats.h File Reference

Function declarations and types used by the structured format parsers.

Data Structures

- struct **nvp_t**

Functions

- **nvp_t * nvp_alloc ()**
- *Allocate a new name/value pair and initialize it with the default settings.* void **nvp_free (nvp_t *nvp)**
- *Free a name/value pair object from memory.* void **nvp_init (nvp_t *nvp)**
- int **nvp_parse (nvp_t *nvp, stringer_t *data)**

Detailed Description

Function declarations and types used by the structured format parsers.

Definition in file **formats.h**.

Function Documentation

nvp_t* nvp_alloc ()

Allocate a new name/value pair and initialize it with the default settings.

Note:

Defaults use "\n" for a line separator, "#" for a comment starting character, and "=" as the assignment character.

Returns:

NULL on failure, or a pointer to the newly allocated name/value pair object.

Definition at line 77 of file nvp.c.

References `nvp_t::comment`, `inx_alloc()`, `nvp_t::line`, `log_info`, `M_INX_HASHED`, `mm_alloc()`, `mm_free()`, `nvp_t::pairs`, `st_free()`, `nvp_t::tokens`, and `nvp_t::value`.

Referenced by `config_load_cmdline_settings()`, and `config_load_file_settings()`.

void nvp_free (nvp_t * nvp)

Free a name/value pair object from memory.

Parameters:

nvp the name/value pair object to be freed.

Returns:

This function returns no value.

Definition at line 104 of file nvp.c.

References `inx_free()`, `mm_free()`, and `nvp_t::pairs`.

Referenced by `config_load_cmdline_settings()`, and `config_load_file_settings()`.

void nvp_init (nvp_t * *nvp*)

int nvp_parse (nvp_t * *nvp*, stringer_t * *data*)

on success the number of valid pairs is returned on error -1 is returned

Definition at line 19 of file nvp.c.

References `nvp_t::comment`, `count`, `inx_insert()`, `nvp_t::line`, `log_options`, `M_LOG_PEDANTIC`, `M_LOG_STACK_TRACE`, `M_TYPE_STRINGER`, `mt_set_type()`, `nvp_t::pairs`, `pl_char_get()`, `pl_data_get()`, `pl_empty()`, `pl_length_get()`, `pl_null()`, `pl_starts_with_char()`, `pl_trim()`, `placer_t`, `multi_t::st`, `st_free()`, `st_import()`, `st_length_get()`, `tok_get_bl()`, `tok_pop()`, `tok_pop_init_st()`, `nvp_t::tokens`, `multi_t::val`, and `nvp_t::value`.

Referenced by `config_load_cmdline_settings()`, and `config_load_file_settings()`.

magma/core/parsers/formats/nvp.c File Reference

Interface to the name/value pair parser.

```
#include "magma.h"
```

Functions

- `int nvp_parse (nvp_t *nvp, stringer_t *data)`
 - `nvp_t * nvp_alloc ()`
 - *Allocate a new name/value pair and initialize it with the default settings.* `void nvp_free (nvp_t *nvp)`
Free a name/value pair object from memory.
-

Detailed Description

Interface to the name/value pair parser.

Definition in file `nvp.c`.

Function Documentation

`nvp_t* nvp_alloc ()`

Allocate a new name/value pair and initialize it with the default settings.

Note:

Defaults use "\n" for a line separator, "#" for a comment starting character, and "=" as the assignment character.

Returns:

NULL on failure, or a pointer to the newly allocated name/value pair object.

Definition at line 77 of file `nvp.c`.

References `nvp_t::comment`, `inx_alloc()`, `nvp_t::line`, `log_info`, `M_INX_HASHED`, `mm_alloc()`, `mm_free()`, `nvp_t::pairs`, `st_free()`, `nvp_t::tokens`, and `nvp_t::value`.

Referenced by `config_load_cmdline_settings()`, and `config_load_file_settings()`.

`void nvp_free (nvp_t * nvp)`

Free a name/value pair object from memory.

Parameters:

nvp the name/value pair object to be freed.

Returns:

This function returns no value.

Definition at line 104 of file `nvp.c`.

References `inx_free()`, `mm_free()`, and `nvp_t::pairs`.

Referenced by config_load_cmdline_settings(), and config_load_file_settings().

int nvp_parse (nvp_t * *nvp*, stringer_t * *data*)

on success the number of valid pairs is returned on error -1 is returned

Definition at line 19 of file nvp.c.

References nvp_t::comment, count, inx_insert(), nvp_t::line, log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_STRINGER, mt_set_type(), nvp_t::pairs, pl_char_get(), pl_data_get(), pl_empty(), pl_length_get(), pl_null(), pl_starts_with_char(), pl_trim(), placer_t, multi_t::st, st_free(), st_import(), st_length_get(), tok_get_bl(), tok_pop(), tok_pop_init_st(), nvp_t::tokens, multi_t::val, and nvp_t::value.

Referenced by config_load_cmdline_settings(), and config_load_file_settings().

magma/core/parsers/line.c File Reference

Functions used to extract lines from within a larger block of data.

#include "magma.h"

Functions

- **placer_t line_pl_bl** (char **block*, size_t *length*, uint64_t *number*)
Get a placer pointing to the specified line ('
' delimited) of content in a data buffer.
 - **placer_t line_pl_ns** (char **string*, uint64_t *number*)
Get a placer pointing to the specified line ('
' delimited) of a null-terminated string.
 - **placer_t line_pl_st** (stringer_t **string*, uint64_t *number*)
Get a placer pointing to the specified line ('
' delimited) of a managed string.
 - **placer_t line_pl_pl** (placer_t *string*, uint64_t *number*)
Get a placer pointing to the specified line ('
' delimited) of another placer.
-

Detailed Description

Functions used to extract lines from within a larger block of data.

Definition in file **line.c**.

Function Documentation

placer_t line_pl_bl (char * *block*, size_t *length*, uint64_t *number*)

Get a placer pointing to the specified line ('
' delimited) of content in a data buffer.

Parameters:

block a pointer to the block of data to be scanned.
length the length, in bytes, of the specified data buffer.
number the zero-based index of the line to be retrieved from the buffer.

Returns:

a null placer on failure, or a placer pointing to the specified line on success.
Definition at line 22 of file line.c.
References log_pedantic, mm_empty(), pl_init(), and pl_null().
Referenced by line_pl_ns(), line_pl_pl(), and line_pl_st().

placer_t line_pl_ns (char * *string*, uint64_t *number*)

Get a placer pointing to the specified line ('
' delimited) of a null-terminated string.

See also:

line_pl_bl()

Parameters:

string a pointer to a null-terminated string to be scanned.

number the zero-based index of the line to be retrieved from the string.

Returns:

a null placer on failure, or a placer pointing to the specified line on success.

Definition at line 65 of file line.c.

References line_pl_bl(), and ns_length_get().

placer_t line_pl_pl (placer_t *string*, uint64_t *number*)

Get a placer pointing to the specified line ('
' delimited) of another placer.

Parameters:

string a pointer to the placer to be scanned.

number the zero-based index of the line to be retrieved from the placer.

Returns:

a null placer on failure, or a placer pointing to the specified line on success.

Definition at line 87 of file line.c.

References line_pl_bl(), pl_char_get(), and pl_length_get().

placer_t line_pl_st (stringer_t * *string*, uint64_t *number*)

Get a placer pointing to the specified line ('
' delimited) of a managed string.

Parameters:

string a pointer to the managed string to be scanned.

number the zero-based index of the line to be retrieved from the managed string.

Returns:

a null placer on failure, or a placer pointing to the specified line on success.

Definition at line 76 of file line.c.

References line_pl_bl(), st_char_get(), and st_length_get().

Referenced by client_read_line(), con_read_line(), and imap_parse_literal().

magma/core/parsers/numbers/digits.c File Reference

Functions for counting the digit places in a number, including the sign character for signed numbers.

```
#include "magma.h"
```

Functions

- `size_t uint64_digits (uint64_t number)`
- *Count the number of digits needed to represent a base-10 64-bit unsigned integer. `size_t uint32_digits (uint32_t number)`*
- *Count the number of digits needed to represent a base-10 32-bit unsigned integer. `size_t uint16_digits (uint16_t number)`*
- *Count the number of digits needed to represent a base-10 16-bit unsigned integer. `size_t uint8_digits (uint8_t number)`*
- *Count the number of digits needed to represent a base-10 8-bit unsigned integer. `size_t int64_digits (int64_t number)`*
- *Count the number of digits needed to represent a base-10 64-bit signed integer. `size_t int32_digits (int32_t number)`*
- *Count the number of digits needed to represent a base-10 32-bit signed integer. `size_t int16_digits (int16_t number)`*
- *Count the number of digits needed to represent a base-10 16-bit signed integer. `size_t int8_digits (int8_t number)`*

Count the number of digits needed to represent a base-10 8-bit signed integer.

Detailed Description

Functions for counting the digit places in a number, including the sign character for signed numbers.

Definition in file **digits.c**.

Function Documentation

size_t int16_digits (int16_t number)

Count the number of digits needed to represent a base-10 16-bit signed integer.

digits.c

Returns:

the number of digits needed to represent the specified integer, including a negative sign.

Definition at line 118 of file digits.c.

References `length`.

Referenced by `serialize_int16()`.

size_t int32_digits (int32_t number)

Count the number of digits needed to represent a base-10 32-bit signed integer.

Returns:

the number of digits needed to represent the specified integer, including a negative sign.

Definition at line 99 of file digits.c.

References length.

Referenced by serialize_int32().

size_t int64_digits (int64_t *number*)

Count the number of digits needed to represent a base-10 64-bit signed integer.

Returns:

the number of digits needed to represent the specified integer, including a negative sign.

Definition at line 80 of file digits.c.

References length.

Referenced by serialize_int64().

size_t int8_digits (int8_t *number*)

Count the number of digits needed to represent a base-10 8-bit signed integer.

Returns:

the number of digits needed to represent the specified integer, including a negative sign.

Definition at line 137 of file digits.c.

References length.

size_t uint16_digits (uint16_t *number*)

Count the number of digits needed to represent a base-10 16-bit unsigned integer.

Returns:

the number of digits needed to represent the specified integer.

Definition at line 50 of file digits.c.

References length.

Referenced by serialize_uint16().

size_t uint32_digits (uint32_t *number*)

Count the number of digits needed to represent a base-10 32-bit unsigned integer.

Returns:

the number of digits needed to represent the specified integer.

Definition at line 34 of file digits.c.

References length.

Referenced by `serialize_uint32()`.

`size_t uint64_digits (uint64_t number)`

Count the number of digits needed to represent a base-10 64-bit unsigned integer.

Returns:

the number of digits needed to represent specified integer.

Definition at line 19 of file `digits.c`.

References length.

Referenced by `mail_build_signature()`, `serialize_uint64()`, `sess_key()`, and `smtp_store_spamsig()`.

`size_t uint8_digits (uint8_t number)`

Count the number of digits needed to represent a base-10 8-bit unsigned integer.

Returns:

the number of digits needed to represent the specified integer.

Definition at line 65 of file `digits.c`.

References length.

magma/core/parsers/numbers/numbers.c File Reference

Functions for converting different string types into binary numbers.

```
#include "magma.h"
```

Functions

- **bool_t float_conv** (**stringer_t** *s, float_t *number)
- Convert a managed string to a float. **bool_t double_conv** (**stringer_t** *s, double_t *number)
- Convert a managed string to a double. **bool_t size_conv_bl** (void *block, size_t length, size_t *number)
- Convert a numerical string into a size_t value. **bool_t ssize_conv_bl** (void *block, size_t length, ssize_t *number)
- Convert a numerical string into an ssize_t value. **bool_t uint64_conv_bl** (void *block, size_t length, uint64_t *number)
- Convert a numerical string into an unsigned 64-bit integer. **bool_t uint64_conv_ns** (char *string, uint64_t *number)
- Convert a null-terminated string into an unsigned 64-bit integer. **bool_t uint64_conv_st** (**stringer_t** *string, uint64_t *number)
- Convert a managed string into an unsigned 64-bit integer. **bool_t uint64_conv_pl** (**placer_t** string, uint64_t *number)
- Convert a placer into an unsigned 64-bit integer. **bool_t uint32_conv_bl** (void *block, size_t length, uint32_t *number)
- Convert a numerical string into an unsigned 32-bit integer. **bool_t uint32_conv_ns** (char *string, uint32_t *number)
- Convert a null-terminated string into an unsigned 32-bit integer. **bool_t uint32_conv_st** (**stringer_t** *string, uint32_t *number)
- Convert a managed string into an unsigned 32-bit integer. **bool_t uint16_conv_bl** (void *block, size_t length, uint16_t *number)
- Convert a numerical string to a 16-bit unsigned integer. **bool_t uint16_conv_ns** (char *string, uint16_t *number)
- Convert a numerical string to a 16-bit unsigned integer. **bool_t uint16_conv_st** (**stringer_t** *string, uint16_t *number)
- Convert a numerical string to a 16-bit unsigned integer. **bool_t uint8_conv_bl** (void *block, size_t length, uint8_t *number)
- Convert a numerical string to an unsigned 8-bit unsigned integer. **bool_t uint8_conv_ns** (char *string, uint8_t *number)
- Convert a numerical string to an 8-bit unsigned integer. **bool_t uint8_conv_st** (**stringer_t** *string, uint8_t *number)
- Convert a numerical string to an 8-bit unsigned integer. **bool_t int64_conv_bl** (void *block, size_t length, int64_t *number)
- Convert a numerical string into a signed 64-bit integer. **bool_t int64_conv_ns** (char *string, int64_t *number)
- Convert a null-terminated string into a signed 64-bit integer. **bool_t int64_conv_st** (**stringer_t** *string, int64_t *number)
- Convert a managed string into a signed 64-bit integer. **bool_t int32_conv_bl** (void *block, size_t length, int32_t *number)
- Convert a numerical string into a signed 32-bit integer. **bool_t int32_conv_ns** (char *string, int32_t *number)
- Convert a null-terminated string into a signed 32-bit integer. **bool_t int32_conv_st** (**stringer_t** *string, int32_t *number)
- Convert a managed string into a signed 32-bit integer. **bool_t int16_conv_bl** (void *block, size_t length, int16_t *number)
- Convert a numerical string into a signed 16-bit integer. **bool_t int16_conv_ns** (char *string, int16_t *number)
- Convert a null-terminated string into a signed 16-bit integer. **bool_t int16_conv_st** (**stringer_t** *string, int16_t *number)

- Convert a managed string into a signed 16-bit integer. **bool_t int8_conv_bl** (void *block, size_t **length**, int8_t ***number**)
- Convert a numerical string into a signed 8-bit integer. **bool_t int8_conv_ns** (char *string, int8_t ***number**)
- Convert a null-terminated string into a signed 8-bit integer. **bool_t int8_conv_st** (stringer_t *string, int8_t ***number**)

Convert a managed string into a signed 8-bit integer.

Detailed Description

Functions for converting different string types into binary numbers.

Definition in file **numbers.c**.

Function Documentation

bool_t double_conv (stringer_t * **s**, double_t * **number**)

Convert a managed string to a double.

numbers.c

Parameters:

s the managed string to be converted.

number a pointer to a double where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 53 of file numbers.c.

References log_pedantic, st_char_get(), st_empty(), st_length_get(), and st_length_int().

bool_t float_conv (stringer_t * **s**, float_t * **number**)

Convert a managed string to a float.

Parameters:

s the managed string to be converted.

number a pointer to a float where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 21 of file numbers.c.

References log_pedantic, st_char_get(), st_empty(), st_length_get(), and st_length_int().

bool_t int16_conv_bl (void * **block**, size_t **length**, int16_t * **number**)

Convert a numerical string into a signed 16-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to a signed 16-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 564 of file numbers.c.

References data, and log_pedantic.

Referenced by deserialize_int16(), int16_conv_ns(), and int16_conv_st().

bool_t int16_conv_ns (char * *string*, int16_t * *number*)

Convert a null-terminated string into a signed 16-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the null-terminated string to be converted.

number a pointer to a signed 16-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 620 of file numbers.c.

References int16_conv_bl(), and ns_length_get().

bool_t int16_conv_st (stringer_t * *string*, int16_t * *number*)

Convert a managed string into a signed 16-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the managed string to be converted.

number a pointer to a signed 16-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 631 of file numbers.c.

References int16_conv_bl(), st_data_get(), and st_length_get().

Referenced by xml_get_xpath_int16().

bool_t int32_conv_bl (void * *block*, size_t *length*, int32_t * *number*)

Convert a numerical string into a signed 32-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to a signed 32-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 485 of file numbers.c.

References data, and log_pedantic.

Referenced by deserialize_int32(), http_parse_header(), imap_fetch_parse_partial(), int32_conv_ns(), and int32_conv_st().

bool_t int32_conv_ns (char * *string*, int32_t * *number*)

Convert a null-terminated string into a signed 32-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the null-terminated string to be converted.

number a pointer to a signed 32-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 541 of file numbers.c.

References int32_conv_bl(), and ns_length_get().

Referenced by process_kill().

bool_t int32_conv_st (stringer_t * *string*, int32_t * *number*)

Convert a managed string into a signed 32-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the managed string to be converted.

number a pointer to a signed 32-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 552 of file numbers.c.

References int32_conv_bl(), st_data_get(), and st_length_get().

Referenced by cache_set_value(), config_value_set(), relay_set_value(), servers_set_value(), and xml_get_xpath_int32().

bool_t int64_conv_bl (void * *block*, size_t *length*, int64_t * *number*)

Convert a numerical string into a signed 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to a signed 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 406 of file numbers.c.

References data, and log_pedantic.

Referenced by deserialize_int64(), int64_conv_ns(), int64_conv_st(), and ssize_conv_bl().

bool_t int64_conv_ns (char * *string*, int64_t * *number*)

Convert a null-terminated string into a signed 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the null-terminated string to be converted.

number a pointer to a signed 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 462 of file numbers.c.

References int64_conv_bl(), and ns_length_get().

bool_t int64_conv_st (stringer_t * *string*, int64_t * *number*)

Convert a managed string into a signed 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the managed string to be converted.

number a pointer to a signed 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 473 of file numbers.c.

References int64_conv_bl(), st_data_get(), and st_length_get().

Referenced by `cache_set_value()`, `config_value_set()`, `relay_set_value()`, `servers_set_value()`, and `xml_get_xpath_int64()`.

`bool_t int8_conv_bl (void * block, size_t length, int8_t * number)`

Convert a numerical string into a signed 8-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to a signed 8-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 643 of file `numbers.c`.

References `data`, and `log_pedantic`.

Referenced by `int8_conv_ns()`, and `int8_conv_st()`.

`bool_t int8_conv_ns (char * string, int8_t * number)`

Convert a null-terminated string into a signed 8-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the null-terminated string to be converted.

number a pointer to a signed 8-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 699 of file `numbers.c`.

References `int8_conv_bl()`, and `ns_length_get()`.

`bool_t int8_conv_st (stringer_t * string, int8_t * number)`

Convert a managed string into a signed 8-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the managed string to be converted.

number a pointer to a signed 8-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 710 of file numbers.c.

References `int8_conv_bl()`, `st_data_get()`, and `st_length_get()`.

Referenced by `cache_set_value()`, `config_value_set()`, `relay_set_value()`, `servers_set_value()`, and `xml_get_xpath_int8()`.

`bool_t size_conv_bl (void * block, size_t length, size_t * number)`

Convert a numerical string into a `size_t` value.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to an `size_t` value where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 87 of file numbers.c.

References `uint64_conv_bl()`.

Referenced by `http_body()`.

`bool_t ssize_conv_bl (void * block, size_t length, ssize_t * number)`

Convert a numerical string into an `ssize_t` value.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to an `ssize_t` value where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 99 of file numbers.c.

References `int64_conv_bl()`.

`bool_t uint16_conv_bl (void * block, size_t length, uint16_t * number)`

Convert a numerical string to a 16-bit unsigned integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the block of memory to be converted.

length the length, in bytes, of the numerical string.

number a pointer to an unsigned 16-bit integer that will hold the result of the conversion.

Returns:

true on success, or false on failure.

Definition at line 264 of file numbers.c.

References data, and log_pedantic.

Referenced by deserialize_uint16(), uint16_conv_ns(), and uint16_conv_st().

bool_t uint16_conv_ns (char * *string*, uint16_t * *number*)

Convert a numerical string to a 16-bit unsigned integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to a null-terminated string containing the data to be converted.

number a pointer to an unsigned 16-bit integer that will hold the result of the conversion.

Returns:

true on success, or false on failure.

Definition at line 312 of file numbers.c.

References ns_length_get(), and uint16_conv_bl().

bool_t uint16_conv_st (stringer_t * *string*, uint16_t * *number*)

Convert a numerical string to a 16-bit unsigned integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to a managed string containing the data to be converted.

number a pointer to an unsigned 16-bit integer that will hold the result of the conversion.

Returns:

true on success, or false on failure.

Definition at line 323 of file numbers.c.

References st_data_get(), st_length_get(), and uint16_conv_bl().

Referenced by cache_set_value(), config_value_set(), relay_set_value(), servers_set_value(), and xml_get_xpath_uint16().

bool_t uint32_conv_bl (void * *block*, size_t *length*, uint32_t * *number*)

Convert a numerical string into an unsigned 32-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to an unsigned 32-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 193 of file numbers.c.

References data, and log_pedantic.

Referenced by cache_config(), deserialize_uint32(), relay_config(), servers_config(), uint32_conv_ns(), and uint32_conv_st().

bool_t uint32_conv_ns (char * *string*, uint32_t * *number*)

Convert a null-terminated string into an unsigned 32-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the null-terminated string to be converted.

number a pointer to an unsigned 32-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 241 of file numbers.c.

References ns_length_get(), and uint32_conv_bl().

bool_t uint32_conv_st (stringer_t * *string*, uint32_t * *number*)

Convert a managed string into an unsigned 32-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the managed string to be converted.

number a pointer to an unsigned 32-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 252 of file numbers.c.

References st_data_get(), st_length_get(), and uint32_conv_bl().

Referenced by cache_set_value(), config_value_set(), imap_fetch_body_part(), imap_search_messages_date_compare(), relay_set_value(), servers_set_value(), and xml_get_xpath_uint32().

bool_t uint64_conv_bl (void * *block*, size_t *length*, uint64_t * *number*)

Convert a numerical string into an unsigned 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to an unsigned 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 111 of file numbers.c.

References data, and log_pedantic.

Referenced by deserialize_uint64(), imap_parse_literal(), pop_num_parse(), pop_top_parse(), size_conv_bl(), smtp_check_receive_quota(), uint64_conv_ns(), uint64_conv_pl(), and uint64_conv_st().

bool_t uint64_conv_ns (char * *string*, uint64_t * *number*)

Convert a null-terminated string into an unsigned 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the null-terminated string to be converted.

number a pointer to an unsigned 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 159 of file numbers.c.

References ns_length_get(), and uint64_conv_bl().

Referenced by lib_load_xml(), and portal_endpoint().

bool_t uint64_conv_pl (placer_t *string*, uint64_t * *number*)

Convert a placer into an unsigned 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the placer to be converted.

number a pointer to an unsigned 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 181 of file numbers.c.

References pl_data_get(), pl_length_get(), and uint64_conv_bl().

Referenced by smtp_parse_mail_from_path(), and stamp_counter_check().

bool_t uint64_conv_st (stringer_t * *string*, uint64_t * *number*)

Convert a managed string into an unsigned 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the managed string to be converted.

number a pointer to an unsigned 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 170 of file numbers.c.

References st_data_get(), st_length_get(), and uint64_conv_bl().

Referenced by cache_set_value(), config_value_set(), imap_narrow_messages(), imap_search_messages_range(), imap_search_messages_size(), portal_get_upload_attachment(), relay_set_value(), servers_set_value(), teacher_process(), and xml_get_xpath_uint64().

bool_t uint8_conv_bl (void * *block*, size_t *length*, uint8_t * *number*)

Convert a numerical string to an unsigned 8-bit unsigned integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the block of memory to be converted.

length the length, in bytes, of the numerical string.

number a pointer to an unsigned 8-bit integer that will hold the result of the conversion.

Returns:

true on success, or false on failure.

Definition at line 335 of file numbers.c.

References data, and log_pedantic.

Referenced by ip_str_subnet(), uint8_conv_ns(), and uint8_conv_st().

bool_t uint8_conv_ns (char * *string*, uint8_t * *number*)

Convert a numerical string to an 8-bit unsigned integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to a null-terminated string containing the data to be converted.

number a pointer to an unsigned 8-bit integer that will hold the result of the conversion.

Returns:

true on success, or false on failure.

Definition at line 383 of file numbers.c.

References ns_length_get(), and uint8_conv_bl().

bool_t uint8_conv_st (stringer_t * *string*, uint8_t * *number*)

Convert a numerical string to an 8-bit unsigned integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to a managed string containing the data to be converted.

number a pointer to an unsigned 8-bit integer that will hold the result of the conversion.

Returns:

true on success, or false on failure.

Definition at line 394 of file numbers.c.

References st_data_get(), st_length_get(), and uint8_conv_bl().

Referenced by cache_set_value(), config_value_set(), relay_set_value(), servers_set_value(), and xml_get_xpath_uint8().

magma/core/parsers/numbers/numbers.h File Reference

Function declarations for the number conversion functions.

Functions

- `size_t int16_digits (int16_t number)`
- *digits.c* `size_t int32_digits (int32_t number)`
- *Count the number of digits needed to represent a base-10 32-bit signed integer.* `size_t int64_digits (int64_t number)`
- *Count the number of digits needed to represent a base-10 64-bit signed integer.* `size_t int8_digits (int8_t number)`
- *Count the number of digits needed to represent a base-10 8-bit signed integer.* `size_t uint16_digits (uint16_t number)`
- *Count the number of digits needed to represent a base-10 16-bit unsigned integer.* `size_t uint32_digits (uint32_t number)`
- *Count the number of digits needed to represent a base-10 32-bit unsigned integer.* `size_t uint64_digits (uint64_t number)`
- *Count the number of digits needed to represent a base-10 64-bit unsigned integer.* `size_t uint8_digits (uint8_t number)`
- *Count the number of digits needed to represent a base-10 8-bit unsigned integer.* `bool_t double_conv (stringer_t *s, double_t *number)`
- *numbers.c* `bool_t float_conv (stringer_t *s, float_t *number)`
- *Convert a managed string to a float.* `bool_t int16_conv_bl (void *block, size_t length, int16_t *number)`
- *Convert a numerical string into a signed 16-bit integer.* `bool_t int16_conv_ns (char *string, int16_t *number)`
- *Convert a null-terminated string into a signed 16-bit integer.* `bool_t int16_conv_st (stringer_t *string, int16_t *number)`
- *Convert a managed string into a signed 16-bit integer.* `bool_t int32_conv_bl (void *block, size_t length, int32_t *number)`
- *Convert a numerical string into a signed 32-bit integer.* `bool_t int32_conv_ns (char *string, int32_t *number)`
- *Convert a null-terminated string into a signed 32-bit integer.* `bool_t int32_conv_st (stringer_t *string, int32_t *number)`
- *Convert a managed string into a signed 32-bit integer.* `bool_t int64_conv_bl (void *block, size_t length, int64_t *number)`
- *Convert a numerical string into a signed 64-bit integer.* `bool_t int64_conv_ns (char *string, int64_t *number)`
- *Convert a null-terminated string into a signed 64-bit integer.* `bool_t int64_conv_st (stringer_t *string, int64_t *number)`
- *Convert a managed string into a signed 64-bit integer.* `bool_t int8_conv_bl (void *block, size_t length, int8_t *number)`
- *Convert a numerical string into a signed 8-bit integer.* `bool_t int8_conv_ns (char *string, int8_t *number)`
- *Convert a null-terminated string into a signed 8-bit integer.* `bool_t int8_conv_st (stringer_t *string, int8_t *number)`
- *Convert a managed string into a signed 8-bit integer.* `bool_t size_conv_bl (void *block, size_t length, size_t *number)`
- *Convert a numerical string into a size_t value.* `bool_t ssize_conv_bl (void *block, size_t length, ssize_t *number)`
- *Convert a numerical string into an ssize_t value.* `bool_t uint16_conv_bl (void *block, size_t length, uint16_t *number)`
- *Convert a numerical string to a 16-bit unsigned integer.* `bool_t uint16_conv_ns (char *string, uint16_t *number)`
- *Convert a numerical string to a 16-bit unsigned integer.* `bool_t uint16_conv_st (stringer_t *string, uint16_t *number)`

- Convert a numerical string to a 16-bit unsigned integer. **bool_t uint32_conv_bl** (void *block, size_t length, uint32_t *number)
- Convert a numerical string into an unsigned 32-bit integer. **bool_t uint32_conv_ns** (char *string, uint32_t *number)
- Convert a null-terminated string into an unsigned 32-bit integer. **bool_t uint32_conv_st** (stringer_t *string, uint32_t *number)
- Convert a managed string into an unsigned 32-bit integer. **bool_t uint64_conv_bl** (void *block, size_t length, uint64_t *number)
- Convert a numerical string into an unsigned 64-bit integer. **bool_t uint64_conv_ns** (char *string, uint64_t *number)
- Convert a null-terminated string into an unsigned 64-bit integer. **bool_t uint64_conv_pl** (placer_t string, uint64_t *number)
- Convert a placer into an unsigned 64-bit integer. **bool_t uint64_conv_st** (stringer_t *string, uint64_t *number)
- Convert a managed string into an unsigned 64-bit integer. **bool_t uint8_conv_bl** (void *block, size_t length, uint8_t *number)
- Convert a numerical string to an unsigned 8-bit unsigned integer. **bool_t uint8_conv_ns** (char *string, uint8_t *number)
- Convert a numerical string to an 8-bit unsigned integer. **bool_t uint8_conv_st** (stringer_t *string, uint8_t *number)

Convert a numerical string to an 8-bit unsigned integer.

Detailed Description

Function declarations for the number conversion functions.

Definition in file **numbers.h**.

Function Documentation

bool_t double_conv (stringer_t * s, double_t * number)

numbers.c

numbers.c

Parameters:

s the managed string to be converted.

number a pointer to a double where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 53 of file numbers.c.

References log_pedantic, st_char_get(), st_empty(), st_length_get(), and st_length_int().

bool_t float_conv (stringer_t * s, float_t * number)

Convert a managed string to a float.

Parameters:

s the managed string to be converted.

number a pointer to a float where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 21 of file numbers.c.

References log_pedantic, st_char_get(), st_empty(), st_length_get(), and st_length_int().

bool_t int16_conv_bl (void * *block*, size_t *length*, int16_t * *number*)

Convert a numerical string into a signed 16-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to a signed 16-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 564 of file numbers.c.

References data, and log_pedantic.

Referenced by deserialize_int16(), int16_conv_ns(), and int16_conv_st().

bool_t int16_conv_ns (char * *string*, int16_t * *number*)

Convert a null-terminated string into a signed 16-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the null-terminated string to be converted.

number a pointer to a signed 16-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 620 of file numbers.c.

References int16_conv_bl(), and ns_length_get().

bool_t int16_conv_st (stringer_t * *string*, int16_t * *number*)

Convert a managed string into a signed 16-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the managed string to be converted.

number a pointer to a signed 16-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 631 of file numbers.c.

References `int16_conv_bl()`, `st_data_get()`, and `st_length_get()`.

Referenced by `xml_get_xpath_int16()`.

size_t int16_digits (int16_t *number*)

digits.c

digits.c

Returns:

the number of digits needed to represent the specified integer, including a negative sign.

Definition at line 118 of file digits.c.

References `length`.

Referenced by `serialize_int16()`.

bool_t int32_conv_bl (void * *block*, size_t *length*, int32_t * *number*)

Convert a numerical string into a signed 32-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to a signed 32-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 485 of file numbers.c.

References `data`, and `log_pedantic`.

Referenced by `deserialize_int32()`, `http_parse_header()`, `imap_fetch_parse_partial()`, `int32_conv_ns()`, and `int32_conv_st()`.

bool_t int32_conv_ns (char * *string*, int32_t * *number*)

Convert a null-terminated string into a signed 32-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the null-terminated string to be converted.

number a pointer to a signed 32-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 541 of file numbers.c.

References `int32_conv_bl()`, and `ns_length_get()`.

Referenced by `process_kill()`.

`bool_t int32_conv_st (stringer_t * string, int32_t * number)`

Convert a managed string into a signed 32-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the managed string to be converted.

number a pointer to a signed 32-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 552 of file numbers.c.

References `int32_conv_bl()`, `st_data_get()`, and `st_length_get()`.

Referenced by `cache_set_value()`, `config_value_set()`, `relay_set_value()`, `servers_set_value()`, and `xml_get_xpath_int32()`.

`size_t int32_digits (int32_t number)`

Count the number of digits needed to represent a base-10 32-bit signed integer.

Returns:

the number of digits needed to represent the specified integer, including a negative sign.

Definition at line 99 of file digits.c.

References `length`.

Referenced by `serialize_int32()`.

`bool_t int64_conv_bl (void * block, size_t length, int64_t * number)`

Convert a numerical string into a signed 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to a signed 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 406 of file numbers.c.

References data, and log_pedantic.

Referenced by deserialize_int64(), int64_conv_ns(), int64_conv_st(), and ssize_conv_bl().

bool_t int64_conv_ns (char * *string*, int64_t * *number*)

Convert a null-terminated string into a signed 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the null-terminated string to be converted.

number a pointer to a signed 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 462 of file numbers.c.

References int64_conv_bl(), and ns_length_get().

bool_t int64_conv_st (stringer_t * *string*, int64_t * *number*)

Convert a managed string into a signed 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the managed string to be converted.

number a pointer to a signed 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 473 of file numbers.c.

References int64_conv_bl(), st_data_get(), and st_length_get().

Referenced by cache_set_value(), config_value_set(), relay_set_value(), servers_set_value(), and xml_get_xpath_int64().

size_t int64_digits (int64_t *number*)

Count the number of digits needed to represent a base-10 64-bit signed integer.

Returns:

the number of digits needed to represent the specified integer, including a negative sign.

Definition at line 80 of file digits.c.

References length.

Referenced by `serialize_int64()`.

`bool_t int8_conv_bl (void * block, size_t length, int8_t * number)`

Convert a numerical string into a signed 8-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to a signed 8-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 643 of file `numbers.c`.

References `data`, and `log_pedantic`.

Referenced by `int8_conv_ns()`, and `int8_conv_st()`.

`bool_t int8_conv_ns (char * string, int8_t * number)`

Convert a null-terminated string into a signed 8-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the null-terminated string to be converted.

number a pointer to a signed 8-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 699 of file `numbers.c`.

References `int8_conv_bl()`, and `ns_length_get()`.

`bool_t int8_conv_st (stringer_t * string, int8_t * number)`

Convert a managed string into a signed 8-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the managed string to be converted.

number a pointer to a signed 8-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 710 of file `numbers.c`.

References `int8_conv_bl()`, `st_data_get()`, and `st_length_get()`.

Referenced by `cache_set_value()`, `config_value_set()`, `relay_set_value()`, `servers_set_value()`, and `xml_get_xpath_int8()`.

`size_t int8_digits (int8_t number)`

Count the number of digits needed to represent a base-10 8-bit signed integer.

Returns:

the number of digits needed to represent the specified integer, including a negative sign.

Definition at line 137 of file `digits.c`.

References `length`.

`bool_t size_conv_bl (void * block, size_t length, size_t * number)`

Convert a numerical string into a `size_t` value.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to an `size_t` value where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 87 of file `numbers.c`.

References `uint64_conv_bl()`.

Referenced by `http_body()`.

`bool_t ssize_conv_bl (void * block, size_t length, ssize_t * number)`

Convert a numerical string into an `ssize_t` value.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to an `ssize_t` value where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 99 of file `numbers.c`.

References `int64_conv_bl()`.

bool_t uint16_conv_bl (void * *block*, size_t *length*, uint16_t * *number*)

Convert a numerical string to a 16-bit unsigned integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the block of memory to be converted.

length the length, in bytes, of the numerical string.

number a pointer to an unsigned 16-bit integer that will hold the result of the conversion.

Returns:

true on success, or false on failure.

Definition at line 264 of file numbers.c.

References data, and log_pedantic.

Referenced by deserialize_uint16(), uint16_conv_ns(), and uint16_conv_st().

bool_t uint16_conv_ns (char * *string*, uint16_t * *number*)

Convert a numerical string to a 16-bit unsigned integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to a null-terminated string containing the data to be converted.

number a pointer to an unsigned 16-bit integer that will hold the result of the conversion.

Returns:

true on success, or false on failure.

Definition at line 312 of file numbers.c.

References ns_length_get(), and uint16_conv_bl().

bool_t uint16_conv_st (stringer_t * *string*, uint16_t * *number*)

Convert a numerical string to a 16-bit unsigned integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to a managed string containing the data to be converted.

number a pointer to an unsigned 16-bit integer that will hold the result of the conversion.

Returns:

true on success, or false on failure.

Definition at line 323 of file numbers.c.

References st_data_get(), st_length_get(), and uint16_conv_bl().

Referenced by `cache_set_value()`, `config_value_set()`, `relay_set_value()`, `servers_set_value()`, and `xml_get_xpath_uint16()`.

size_t uint16_digits (uint16_t *number*)

Count the number of digits needed to represent a base-10 16-bit unsigned integer.

Returns:

the number of digits needed to represent the specified integer.

Definition at line 50 of file `digits.c`.

References `length`.

Referenced by `serialize_uint16()`.

bool_t uint32_conv_bl (void * *block*, size_t *length*, uint32_t * *number*)

Convert a numerical string into an unsigned 32-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to an unsigned 32-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 193 of file `numbers.c`.

References `data`, and `log_pedantic`.

Referenced by `cache_config()`, `deserialize_uint32()`, `relay_config()`, `servers_config()`, `uint32_conv_ns()`, and `uint32_conv_st()`.

bool_t uint32_conv_ns (char * *string*, uint32_t * *number*)

Convert a null-terminated string into an unsigned 32-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the null-terminated string to be converted.

number a pointer to an unsigned 32-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 241 of file `numbers.c`.

References `ns_length_get()`, and `uint32_conv_bl()`.

bool_t uint32_conv_st (stringer_t * *string*, uint32_t * *number*)

Convert a managed string into an unsigned 32-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the managed string to be converted.

number a pointer to an unsigned 32-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 252 of file numbers.c.

References st_data_get(), st_length_get(), and uint32_conv_bl().

Referenced by cache_set_value(), config_value_set(), imap_fetch_body_part(), imap_search_messages_date_compare(), relay_set_value(), servers_set_value(), and xml_get_xpath_uint32().

size_t uint32_digits (uint32_t *number*)

Count the number of digits needed to represent a base-10 32-bit unsigned integer.

Returns:

the number of digits needed to represent the specified integer.

Definition at line 34 of file digits.c.

References length.

Referenced by serialize_uint32().

bool_t uint64_conv_bl (void * *block*, size_t *length*, uint64_t * *number*)

Convert a numerical string into an unsigned 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the numerical string to be converted.

length the length, in bytes, of the numerical data.

number a pointer to an unsigned 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 111 of file numbers.c.

References data, and log_pedantic.

Referenced by deserialize_uint64(), imap_parse_literal(), pop_num_parse(), pop_top_parse(), size_conv_bl(), smtp_check_receive_quota(), uint64_conv_ns(), uint64_conv_pl(), and uint64_conv_st().

bool_t uint64_conv_ns (char * *string*, uint64_t * *number*)

Convert a null-terminated string into an unsigned 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the null-terminated string to be converted.

number a pointer to an unsigned 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 159 of file numbers.c.

References ns_length_get(), and uint64_conv_bl().

Referenced by lib_load_xml(), and portal_endpoint().

bool_t uint64_conv_pl (placer_t *string*, uint64_t * *number*)

Convert a placer into an unsigned 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the placer to be converted.

number a pointer to an unsigned 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 181 of file numbers.c.

References pl_data_get(), pl_length_get(), and uint64_conv_bl().

Referenced by smtp_parse_mail_from_path(), and stamp_counter_check().

bool_t uint64_conv_st (stringer_t * *string*, uint64_t * *number*)

Convert a managed string into an unsigned 64-bit integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to the managed string to be converted.

number a pointer to an unsigned 64-bit number where the result of the conversion will be stored.

Returns:

true on success or false on failure.

Definition at line 170 of file numbers.c.

References st_data_get(), st_length_get(), and uint64_conv_bl().

Referenced by `cache_set_value()`, `config_value_set()`, `imap_narrow_messages()`, `imap_search_messages_range()`, `imap_search_messages_size()`, `portal_get_upload_attachment()`, `relay_set_value()`, `servers_set_value()`, `teacher_process()`, and `xml_get_xpath_uint64()`.

size_t uint64_digits (uint64_t *number*)

Count the number of digits needed to represent a base-10 64-bit unsigned integer.

Returns:

the number of digits needed to represent specified integer.

Definition at line 19 of file `digits.c`.

References `length`.

Referenced by `mail_build_signature()`, `serialize_uint64()`, `sess_key()`, and `smtp_store_spamsig()`.

bool_t uint8_conv_bl (void * *block*, size_t *length*, uint8_t * *number*)

Convert a numerical string to an unsigned 8-bit unsigned integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

block a pointer to the block of memory to be converted.

length the length, in bytes, of the numerical string.

number a pointer to an unsigned 8-bit integer that will hold the result of the conversion.

Returns:

true on success, or false on failure.

Definition at line 335 of file `numbers.c`.

References `data`, and `log_pedantic`.

Referenced by `ip_str_subnet()`, `uint8_conv_ns()`, and `uint8_conv_st()`.

bool_t uint8_conv_ns (char * *string*, uint8_t * *number*)

Convert a numerical string to an 8-bit unsigned integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to a null-terminated string containing the data to be converted.

number a pointer to an unsigned 8-bit integer that will hold the result of the conversion.

Returns:

true on success, or false on failure.

Definition at line 383 of file `numbers.c`.

References `ns_length_get()`, and `uint8_conv_bl()`.

bool_t uint8_conv_st (stringer_t * *string*, uint8_t * *number*)

Convert a numerical string to an 8-bit unsigned integer.

Note:

This function checks for both numeric underflows and overflows.

Parameters:

string a pointer to a managed string containing the data to be converted.

number a pointer to an unsigned 8-bit integer that will hold the result of the conversion.

Returns:

true on success, or false on failure.

Definition at line 394 of file numbers.c.

References st_data_get(), st_length_get(), and uint8_conv_bl().

Referenced by cache_set_value(), config_value_set(), relay_set_value(), servers_set_value(), and xml_get_xpath_uint8().

size_t uint8_digits (uint8_t *number*)

Count the number of digits needed to represent a base-10 8-bit unsigned integer.

Returns:

the number of digits needed to represent the specified integer.

Definition at line 65 of file digits.c.

References length.

magma/core/parsers/parsers.h File Reference

The function declarations and types needed by the generic parsing functions.

```
#include "numbers/numbers.h"
#include "formats/formats.h"
#include "special/special.h"
```

Data Structures

- struct **tok_state_t**

Functions

- uint64_t **time_datestamp** (void)
- *time.c* stringer_t * **time_print_gmt** (stringer_t *s, chr_t *format, time_t moment)
- *Get a specified time as a formatted string converted to GMT.* stringer_t * **time_print_local** (stringer_t *s, chr_t *format, time_t moment)
- *Get a specified time as a formatted string.* uint64_t **time_till_midnight** (void)
- *Get the number of seconds until midnight.* uchr_t **lower_chr** (uchr_t c)
- *Return the lowercase representation of a character.* stringer_t * **lower_st** (stringer_t *s)
- *Transform a managed string (in-place) into lowercase.* uchr_t **upper_chr** (uchr_t c)
- *Return the uppercase representation of a character.* stringer_t * **upper_st** (stringer_t *s)
- *Transform a managed string (in-place) into uppercase.* uint64_t **tok_get_count_st** (stringer_t *string, char token)
- *Count the number of times a token is found in a managed string.* uint64_t **tok_get_count_bl** (void *block, size_t length, char token)
- *Count the number of times a token is found in a specified block of memory.* uint64_t **str_tok_get_count_bl** (void *block, size_t length, chr_t *token, size_t token)
- *Count the number of times a string token is found in a specified block of memory.* int **tok_get_pl** (placer_t string, char token, uint64_t fragment, placer_t *value)
- *Retrieve a specified token from a placer.* int **tok_get_st** (stringer_t *string, char token, uint64_t fragment, placer_t *value)
- *Retrieve a specified token from a managed string.* int **tok_get_bl** (void *block, size_t length, char token, uint64_t fragment, placer_t *value)
- *Retrieve a specified token from a block of data.* int **tok_get_ns** (char *string, size_t length, char token, uint64_t fragment, placer_t *value)
- *Retrieve a specified token from a null-terminated string.* int **str_tok_get_bl** (char *block, size_t length, chr_t *token, size_t token, uint64_t fragment, placer_t *value)
- *Retrieve a specified string-split token from a null-terminated string.* int **tok_pop** (tok_state_t *state, placer_t *value)
- void **tok_pop_init_st** (tok_state_t *state, stringer_t *string, char token)
- void **tok_pop_init_bl** (tok_state_t *state, void *block, size_t length, char token)
- bool_t **pl_skip_characters** (placer_t *place, char *skipchars, size_t nchars)
- *Skip past any of the specified characters found at the beginning of the placer, and update the placer accordingly.* bool_t **pl_skip_to_characters** (placer_t *place, char *skiptochars, size_t nchars)
- *Skip to the first instance of any of the specified characters in the placer, and update the placer accordingly.* bool_t **pl_shrink_before_characters** (placer_t *place, char *shrinkchars, size_t nchars)
- *Truncate a placer to start before any of the specified characters, and update the placer accordingly.* bool_t **pl_get_embraced** (placer_t str, placer_t *out, unsigned char opening, unsigned char closing, bool_t required)
- *Get a placer pointing to the text contained within a specified pair of opening and closing braces.* bool_t **pl_update_start** (placer_t *place, size_t nchars, bool_t more)
- *Ensure that a placer contains at least a certain amount of data, and update it accordingly.* placer_t **line_pl_ns** (char *string, uint64_t number)

- *Get a placer pointing to the specified line (' delimited) of a null-terminated string.* **placer_t line_pl_pl** (**placer_t** string, uint64_t **number**)
Get a placer pointing to the specified line (' delimited) of another placer. **placer_t line_pl_st** (**stringer_t** *string, uint64_t **number**)
Get a placer pointing to the specified line (' delimited) of a managed string. **placer_t line_pl_bl** (char *block, size_t **length**, uint64_t **number**)
Get a placer pointing to the specified line (' delimited) of content in a data buffer. **placer_t pl_trim** (**placer_t** place)
 - **trim.c placer_t pl_trim_end** (**placer_t** place)
 - **placer_t pl_trim_start** (**placer_t** place)
 - *Trim the leading whitespace from a placer.* void **st_trim** (**stringer_t** *string)
-

Detailed Description

The function declarations and types needed by the generic parsing functions.

Definition in file **parsers.h**.

Function Documentation

placer_t line_pl_bl (char * *block*, size_t *length*, uint64_t *number*)

Get a placer pointing to the specified line (' delimited) of content in a data buffer.

Parameters:

block a pointer to the block of data to be scanned.
length the length, in bytes, of the specified data buffer.
number the zero-based index of the line to be retrieved from the buffer.

Returns:

a null placer on failure, or a placer pointing to the specified line on success.
 Definition at line 22 of file line.c.
 References log_pedantic, mm_empty(), pl_init(), and pl_null().
 Referenced by line_pl_ns(), line_pl_pl(), and line_pl_st().

placer_t line_pl_ns (char * *string*, uint64_t *number*)

Get a placer pointing to the specified line (' delimited) of a null-terminated string.

See also:

line_pl_bl()

Parameters:

string a pointer to a null-terminated string to be scanned.
number the zero-based index of the line to be retrieved from the string.

Returns:

a null placer on failure, or a placer pointing to the specified line on success.
Definition at line 65 of file line.c.
References line_pl_bl(), and ns_length_get().

placer_t line_pl_pl (placer_t *string*, uint64_t *number*)

Get a placer pointing to the specified line ('
' delimited) of another placer.

Parameters:

string a pointer to the placer to be scanned.
number the zero-based index of the line to be retrieved from the placer.

Returns:

a null placer on failure, or a placer pointing to the specified line on success.
Definition at line 87 of file line.c.
References line_pl_bl(), pl_char_get(), and pl_length_get().

placer_t line_pl_st (stringer_t * *string*, uint64_t *number*)

Get a placer pointing to the specified line ('
' delimited) of a managed string.

Parameters:

string a pointer to the managed string to be scanned.
number the zero-based index of the line to be retrieved from the managed string.

Returns:

a null placer on failure, or a placer pointing to the specified line on success.
Definition at line 76 of file line.c.
References line_pl_bl(), st_char_get(), and st_length_get().
Referenced by client_read_line(), con_read_line(), and imap_parse_literal().

uchr_t lower_chr (uchr_t *c*)

Return the lowercase representation of a character.

Parameters:

c the character to be transformed.

Returns:

the input character, in lowercase.

Definition at line 30 of file case.c.

Referenced by `credential_address()`, `hex_decode_chr()`, `mm_cmp_ci_eq()`, `pop_user_parse()`, `smtp_parse_helo_domain()`, `smtp_parse_mail_from_path()`, `smtp_parse_rcpt_to()`, `st_cmp_ci_ends()`, `st_cmp_ci_eq()`, `st_cmp_ci_starts()`, and `st_search_ci()`.

stringer_t* lower_st (stringer_t * s)

Transform a managed string (in-place) into lowercase.

Parameters:

s the managed string to be modified.

Returns:

NULL on error, or a pointer to the input managed string, in lowercase.

Definition at line 63 of file case.c.

References `log_pedantic`, and `st_empty_out()`.

Referenced by `contact_business_valid_email()`, `http_parse_header()`, and `smtp_rcpt_to()`.

bool_t pl_get_embraced (placer_t str, placer_t * out, unsigned char opening, unsigned char closing, bool_t required)

Get a placer pointing to the text contained within a specified pair of opening and closing braces.

Parameters:

str a placer pointing to the string to be parsed.

out a pointer to a placer that will store the string between the braces on success.

opening the character to be the opening brace of the sequence.

closing the character to be the closing brace of the sequence.

required if true, fail if no data was found between the pair of braces.

Returns:

true if the brace sequence was complete, or false if either component could not be located.

Definition at line 455 of file token.c.

References `pl_char_get()`, and `pl_empty()`.

bool_t pl_shrink_before_characters (placer_t * place, char * shrinkchars, size_t nchars)

Truncate a placer to start before any of the specified characters, and update the placer accordingly.

Parameters:

place a pointer to a placer that will be updated to be truncated before any of the specified characters.

shrinkchars a pointer to a buffer containing bytes that will be skipped when they are found at the end of the placer.

nchars the number of characters to be tested in the collection in *shrinkchars*.

Returns:

true if the shrink operation completed before the end of the placer was reached, or false otherwise.

Definition at line 418 of file token.c.

References `pl_char_get()`, `pl_empty()`, and `pl_length_get()`.

Referenced by `portal_upload()`.

`bool_t pl_skip_characters (placer_t * place, char * skipchars, size_t nchars)`

Skip past any of the specified characters found at the beginning of the placer, and update the placer accordingly.

Parameters:

place a pointer to a placer that will be updated to skip past any of the specified characters.

skipchars a pointer to a buffer containing bytes that will be skipped at the beginning of the placer.

nchars the number of characters to be tested in the collection in *skipchars*.

Returns:

true if the skip operation completed before the end of the placer was reached, or false otherwise.

Definition at line 349 of file `token.c`.

References `pl_char_get()`, and `pl_empty()`.

Referenced by `portal_upload()`.

`bool_t pl_skip_to_characters (placer_t * place, char * skiptochars, size_t nchars)`

Skip to the first instance of any of the specified characters in the placer, and update the placer accordingly.

Parameters:

place a pointer to a placer that will be updated to skip to any of the specified characters.

skiptochars a pointer to a buffer containing bytes that will be skipped to when they are first found in the placer.

nchars the number of characters to be tested in the collection in *skiptochars*.

Returns:

true if the skip operation completed before the end of the placer was reached, or false otherwise.

Definition at line 385 of file `token.c`.

References `pl_char_get()`, and `pl_empty()`.

Referenced by `portal_upload()`.

`placer_t pl_trim (placer_t place)`

`trim.c`

Definition at line 67 of file `trim.c`.

References `CHARS_TO_TRIM`, `pl_char_get()`, `pl_init()`, `pl_length_get()`, and `pl_null()`.

Referenced by `nvp_parse()`, and `smtp_parse_mail_from_path()`.

`placer_t pl_trim_end (placer_t place)`

Definition at line 144 of file `trim.c`.

References `CHARS_TO_TRIM`, `pl_char_get()`, `pl_init()`, `pl_length_get()`, and `pl_null()`.

Referenced by `smtp_parse_helo_domain()`, `smtp_parse_mail_from_path()`, and `smtp_parse_rcpt_to()`.

placer_t pl_trim_start (placer_t *place*)

Trim the leading whitespace from a placer.

Parameters:

place a placer containing the string to have its leading whitespace trimmed.

Returns:

a placer pointing to the trimmed value inside the originally specified input string.

Definition at line 115 of file trim.c.

References CHARS_TO_TRIM, pl_char_get(), pl_init(), pl_length_get(), and pl_null().

Referenced by get_header_opt().

bool_t pl_update_start (placer_t * *place*, size_t *nchars*, bool_t *more*)

Ensure that a placer contains at least a certain amount of data, and update it accordingly.

Parameters:

place a pointer to the placer containing the original data, which will be updated on success.

nchars the minimum number of characters that the placer needs to contain to pass the test.

more if true, more characters following the minimum buffer size are necessary.

Returns:

true if the placer meets the minimum size check, or false if it does not.

Definition at line 498 of file token.c.

void st_trim (stringer_t * *string*)

Definition at line 18 of file trim.c.

References CHARS_TO_TRIM, mm_move(), mm_wipe(), st_avail_get(), st_char_get(), st_length_get(), and st_length_set().

int str_tok_get_bl (char * *block*, size_t *length*, chr_t * *token*, size_t *toklen*, uint64_t *fragment*, placer_t * *value*)

Retrieve a specified string-split token from a null-terminated string.

Parameters:

block a pointer to the block of memory to be tokenized.

length the maximum number of characters to be scanned from the input data.

token the token string that will be used to split the data.

toklen the length, in bytes, of the token string.

fragment the zero-indexed token number to be extracted from the data.

value a pointer to a placer that will receive the value of the extracted token on success, or **pl_null()** on failure.

Returns:

-1 on failure, 0 on success, or 1 if the token was extracted successfully, but was the last one in the string.

Definition at line 303 of file token.c.

References mm_empty(), pl_char_get(), pl_init(), pl_length_get(), pl_null(), placer_t, and st_search_cs().

Referenced by portal_upload().

uint64_t str_tok_get_count_bl (void * *block*, size_t *length*, chr_t * *token*, size_t *toklen*)

Count the number of times a string token is found in a specified block of memory.

Parameters:

block a pointer to a block of memory to be scanned.

length the length, in bytes, of the block of memory to be scanned.

token a pointer to the string token being used to split the input data.

toklen the length, in bytes, of the specified token.

Returns:

the number of times the string token was found in the block of memory, or 0 on failure.

Definition at line 270 of file token.c.

References count, mm_empty(), pl_init(), pl_length_get(), placer_t, and st_search_cs().

Referenced by portal_upload().

uint64_t time_datestamp (void)

time.c

time.c

Returns:

0 on failure or a 64-bit unsigned integer containing the formatted current date on success.

Definition at line 41 of file time.c.

Referenced by log_rotate(), log_start(), pattern_update(), and process_maint().

stringer_t* time_print_gmt (stringer_t * *s*, chr_t * *format*, time_t *moment*)

Get a specified time as a formatted string converted to GMT.

Parameters:

s a managed string where the formatted time is to be stored.

format a null-terminated string containing the format of the time string.

moment the specified time to be formatted, as a UNIX time value.

Returns:

NULL on failure, or a pointer to the managed string containing the result on success.

Definition at line 97 of file time.c.

References log_pedantic, mm_wipe(), st_avail_get(), st_char_get(), and st_length_set().

Referenced by http_response_cookie(), http_response_header(), and http_response_options().

stringer_t* time_print_local (stringer_t * *s*, chr_t * *format*, time_t *moment*)

Get a specified time as a formatted string.

Parameters:

s a managed string where the formatted time is to be stored.
format a null-terminated string containing the format of the time string.
moment the specified time to be formatted, as a UNIX time value.

Returns:

NULL on failure, or a pointer to the managed string containing the result on success.
Definition at line 66 of file time.c.
References log_pedantic, mm_wipe(), st_avail_get(), st_char_get(), and st_length_set().
Referenced by imap_id().

uint64_t time_till_midnight (void)

Get the number of seconds until midnight.

Returns:

an unsigned 64-bit integer containing the number of seconds until midnight, or 86400 on failure.
Definition at line 19 of file time.c.
Referenced by process_maint().

int tok_get_bl (void * *block*, size_t *length*, char *token*, uint64_t *fragment*, placer_t * *value*)

Retrieve a specified token from a block of data.

See also:

tok_get_ns()

Parameters:

block a pointer to the memory block to be tokenized.
length the maximum number of characters to be scanned at the input data block.
token the token character that will be used to split the data block.
fragment the zero-indexed token number to be extracted from the data block.
value a pointer to a placer that will receive the value of the extracted token on success, or **pl_null()** on failure.

Returns:

-1 on failure, 0 on success, or 1 if the token was extracted successfully, but was the last one found in the data block.
Definition at line 141 of file token.c.
References tok_get_ns().
Referenced by nvp_parse(), smtp_parse_mail_from_path(), tok_get_pl(), and tok_get_st().

uint64_t tok_get_count_bl (void * *block*, size_t *length*, char *token*)

Count the number of times a token is found in a specified block of memory.

Parameters:

block a pointer to a block of memory to be scanned.
length the length, in bytes, of the block of memory to be scanned.
token a character specifying the token to be searched.

Returns:

the number of times the token was found in the block of memory, or 0 on failure.

Definition at line 23 of file token.c.

References `count`, `log_options`, `M_LOG_PEDANTIC`, `M_LOG_STACK_TRACE`, and `mm_empty()`.

Referenced by `smtp_parse_mail_from_path()`, and `tok_get_count_st()`.

uint64_t tok_get_count_st (stringer_t * *string*, char *token*)

Count the number of times a token is found in a managed string.

See also:

`tok_get_count_bl()`

Parameters:

string a pointer to the managed string to be scanned.
token a character specifying the token to be searched.

Returns:

the number of times the token was found in the managed string, or 0 on failure.

Definition at line 58 of file token.c.

References `st_data_get()`, `st_length_get()`, and `tok_get_count_bl()`.

Referenced by `get_header_opt()`, `get_header_value_noopt()`, `http_parse_method()`, `http_parse_pairs()`, `imap_fetch_body_part()`, `imap_narrow_messages()`, `imap_search_messages_date()`, `imap_search_messages_date_compare()`, `imap_search_messages_range()`, `ip_str_subnet()`, `mail_domain_get()`, `mail_mime_type_parameters()`, `portal_get_upload_attachment()`, and `stamp_counter_check()`.

int tok_get_ns (char * *string*, size_t *length*, char *token*, uint64_t *fragment*, placer_t * *value*)

Retrieve a specified token from a null-terminated string.

Parameters:

string a pointer to the null-terminated string to be tokenized.
length the maximum number of characters to be scanned from the input string.
token the token character that will be used to split the string.
fragment the zero-indexed token number to be extracted from the string.
value a pointer to a placer that will receive the value of the extracted token on success, or **`pl_null()`** on failure.

Returns:

-1 on failure, 0 on success, or 1 if the token was extracted successfully, but was the last one in the string.

Definition at line 72 of file token.c.

References `log_options`, `M_LOG_PEDANTIC`, `M_LOG_STACK_TRACE`, `mm_empty()`, `pl_init()`, and `pl_null()`.

Referenced by ip_str_subnet(), lib_load_openssl(), and tok_get_bl().

int tok_get_pl (placer_t *string*, char *token*, uint64_t *fragment*, placer_t * *value*)

Retrieve a specified token from a placer.

See also:

token_get_ns()

Parameters:

string a pointer to the placer to be tokenized.

token the token character that will be used to split the placer.

fragment the zero-indexed token number to be extracted from the placer.

value a pointer to a placer that will receive the value of the extracted token on success, or **pl_null()** on failure.

Returns:

-1 on failure, 0 on success, or 1 if the token was extracted successfully, but was the last one found in the placer.

Definition at line 169 of file token.c.

References pl_data_get(), pl_length_get(), and tok_get_bl().

Referenced by http_data_value_parse(), http_parse_context(), http_parse_method(), and smtp_parse_mail_from_path().

int tok_get_st (stringer_t * *string*, char *token*, uint64_t *fragment*, placer_t * *value*)

Retrieve a specified token from a managed string.

See also:

tok_get_ns()

Parameters:

string a pointer to the managed string to be tokenized.

token the token character that will be used to split the managed string.

fragment the zero-indexed token number to be extracted from the managed string.

value a pointer to a placer that will receive the value of the extracted token on success, or **pl_null()** on failure.

Returns:

-1 on failure, 0 on success, or 1 if the token was extracted successfully, but was the last one found in the managed string.

Definition at line 155 of file token.c.

References st_data_get(), st_length_get(), and tok_get_bl().

Referenced by get_header_opt(), get_header_value_noopt(), http_parse_context(), http_parse_pairs(), imap_fetch_body_part(), imap_folder_create(), imap_folder_remove(), imap_folder_rename(), imap_narrow_messages(), imap_parse_address_part(), imap_search_messages_date(), imap_search_messages_date_compare(), imap_search_messages_range(), mail_domain_get(), mail_mime_type_parameters(), portal_get_upload_attachment(), smtp_auth_plain(), and stamp_counter_check().

int tok_pop (tok_state_t * state, placer_t * value)

on error the function returns -1 and sets value to EMPTY_PLACER on success the function returns 0 and stores the token in the placer pointed at by value if the end is hit, then the function returns 1, and places any remaining data in value

Definition at line 201 of file token.c.

References pl_init(), pl_null(), tok_state_t::position, tok_state_t::remaining, and tok_state_t::token.

Referenced by nvp_parse().

void tok_pop_init_bl (tok_state_t * state, void * block, size_t length, char token)

Definition at line 174 of file token.c.

References tok_state_t::block, tok_state_t::length, tok_state_t::position, tok_state_t::remaining, and tok_state_t::token.

void tok_pop_init_st (tok_state_t * state, stringer_t * string, char token)

Definition at line 185 of file token.c.

References tok_state_t::block, tok_state_t::length, tok_state_t::position, tok_state_t::remaining, st_char_get(), st_length_get(), and tok_state_t::token.

Referenced by nvp_parse().

uchr_t upper_chr (uchr_t c)

Return the uppercase representation of a character.

Parameters:

c the character to be transformed.

Returns:

the input character, in uppercase.

Definition at line 20 of file case.c.

stringer_t* upper_st (stringer_t * s)

Transform a managed string (in-place) into uppercase.

Parameters:

s the managed string to be modified.

Returns:

NULL on error, or a pointer to the input managed string, in uppercase.

Definition at line 40 of file case.c.

References log_pedantic, and st_empty_out().

Referenced by http_response_allow_cross(), imap_fetch_body_tag(), imap_fetch_bodystructure(), and mail_mime_type_parameters().

magma/providers/parsers/parsers.h File Reference

The entry point for modules involved with accessing functionality provided by alien code.

Functions

- **bool_t lib_load_jansson** (void)
- *json.c* **chr_t * lib_version_jansson** (void)
- *Return the version string of libjansson.* **bool_t lib_load_xml** (void)
- *xml.c* **bool_t xml_start** (void)
- *Initialize the xml parser library.* **chr_t * xml_get_xpath_ns** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)
- *Get the content of an evaluated xpath expression as a null-terminated string.* **const chr_t * lib_version_xml** (void)
- *Return the version string of libxml.* **int16_t xml_get_xpath_int16** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)
- *Get the value of a node element or attribute selected by an xpath expression as a signed 16-bit integer.* **int32_t xml_get_xpath_int32** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)
- *Get the value of a node element or attribute selected by an xpath expression as a signed 32-bit integer.* **int64_t xml_get_xpath_int64** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)
- *Get the value of a node element or attribute selected by an xpath expression as a signed 64-bit integer.* **int8_t xml_get_xpath_int8** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)
- *Get the value of a node element or attribute selected by an xpath expression as a signed 8-bit integer.* **int_t xml_xpath_set_namespace** (xmlXPathContextPtr ctx, xmlChar *prefix, xmlChar *ns_uri)
- *Set the namespace for an xpath context.* **size_t xml_get_xpath_node_count** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)
- *This function is not currently being called by any other part of the code.* **stringer_t * xml_dump_doc** (xmlDocPtr doc)
- *Get an xml document object as a string.* **stringer_t * xml_get_xpath_st** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)
- *Get the value of a node element or attribute selected by an xpath expression as a managed string.* **stringer_t * xml_node_get_content_st** (xmlNodePtr node)
- *Get the content of an xml node as a managed string.* **uint16_t xml_get_xpath_uint16** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)
- *Get the value of a node element or attribute selected by an xpath expression as an unsigned 16-bit integer.* **uint32_t xml_get_xpath_uint32** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)
- *Get the value of a node element or attribute selected by an xpath expression as an unsigned 32-bit integer.* **uint64_t xml_get_xpath_uint64** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)
- *Get the value of a node element or attribute selected by an xpath expression as an unsigned 64-bit integer.* **uint8_t xml_get_xpath_uint8** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)
- *Get the value of a node element or attribute selected by an xpath expression as an unsigned 8-bit integer.* **bool_t xml_set_xpath_uint64** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query, uint64_t val)
- *Set the value of a node element selected by an xpath expression, as a 64-bit unsigned integer.* **bool_t xml_set_xpath_ns** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query, **uchr_t** *val)
- *Set the value of a node element selected by an xpath expression, as a null-terminated string.* **bool_t xml_set_xpath_property** (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query, **uchr_t** *name, **uchr_t** *val)
- *Set the value of a specified property of a node element selected by an xpath expression.* **void xml_error** (void *ctx, **const chr_t** *format,...)
- *A function handler for displaying xml parser error messages to the user.* **void xml_free_doc** (xmlDocPtr doc)
- *Free an xml document.* **void xml_free_parser_ctx** (xmlParserCtxtPtr ctx)
- *Destroy an xml parser context.* **void xml_free_xpath_ctx** (xmlXPathContextPtr ctx)
- *Free an xml xpath context.* **void xml_free_xpath_obj** (xmlXPathObjectPtr obj)

- *Free an xml xpath object.* void **xml_node_free** (xmlNodePtr node)
- *Free an xml node.* void **xml_node_set_content** (xmlNodePtr node, **uchr_t** *content)
- *Set the content of an xml node.* void **xml_stop** (void)
- *Cleanup the state and allocated memory of the xml parser in preparation to be shutdown.* xmlAttrPtr **xml_node_set_property** (xmlNodePtr node, **uchr_t** *name, **uchr_t** *value)
- *Set an attribute of an xml node.* xmlChar * **xml_encode** (xmlDocPtr doc, **stringer_t** *string)
- *Encode an xml string, performing proper replacement of defined entities and non-ASCII values.* xmlDocPtr **xml_create_doc** (xmlParserCtxtPtr ctx, const **chr_t** *buffer, **int_t** size, const **chr_t** *url, const **chr_t** *encoding, **int_t** options)
- *Create a new xml document object from specified user data.* xmlNodePtr **xml_node_add_sibling** (xmlNodePtr current, xmlNodePtr element)
- *Add an xml node to the list of siblings of another xml node.* xmlNodePtr **xml_node_new** (**uchr_t** *name)
- *Create an xml node.* xmlParserCtxtPtr **xml_create_parser_ctx** (void)
- *Create and initialize a new xml parser context.* xmlXPathContextPtr **xml_create_xpath_ctx** (xmlDocPtr doc)
- *Create an xpath context for an xml document.* xmlXPathObjectPtr **xml_xpath_eval** (const **uchr_t** *xpath, xmlXPathContextPtr ctx)

Evaluate an xml xpath expression.

Detailed Description

The entry point for modules involved with accessing functionality provided by alien code.

Definition in file **parsers.h**.

Function Documentation

bool_t lib_load_jansson (void)

json.c

json.c

Returns:

true on success or false on failure.

Definition at line 27 of file json.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

bool_t lib_load_xml (void)

xml.c

xml.c

Returns:

true on success or false on failure.

Definition at line 31 of file xml.c.

References lib_symbols(), M_BIND, uint64_conv_ns(), and xml_version_string.

Referenced by lib_load().

chr_t* lib_version_jansson (void)

Return the version string of libjansson.

Returns:

a pointer to a character string containing the libjansson version information.

Definition at line 19 of file json.c.

References jansson_version_d.

Referenced by lib_load().

const chr_t* lib_version_xml (void)

Return the version string of libxml.

Returns:

a pointer to a character string containing the libxml version information.

Definition at line 22 of file xml.c.

References xml_version_string.

Referenced by lib_load().

xmlDocPtr xml_create_doc (xmlParserCtxtPtr ctx, const chr_t * buffer, int_t size, const chr_t * url, const chr_t * encoding, int_t options)

Create a new xml document object from specified user data.

See also:

xmlCtxtReadMemory()

Parameters:

ctx a pointer to the xml context to be used for the parsing operation.

buffer a null-terminated string containing the xml data to be parsed.

size the length, in bytes, of the xml data buffer to be parsed.

url a null-terminated string containing the base url to be used for the document.

encoding a null-terminated string specifying the document encoding type, or NULL for default.

options a value containing a mask of xml parser options of type xmlParserOption.

Returns:

NULL on failure, or a pointer to the xml document tree of the parsed xml text on success.

Definition at line 1054 of file xml.c.

References log_pedantic, and xmlCtxtReadMemory_d.

Referenced by http_page_get().

xmlParserCtxtPtr xml_create_parser_ctx (void)

Create and initialize a new xml parser context.

Returns:

NULL on failure, or a newly initialized xml parser context on success.

Definition at line 963 of file xml.c.

References log_pedantic, xml_error(), and xmlNewParserCtxt_d.

Referenced by http_page_get().

xmlXPathContextPtr xml_create_xpath_ctx (xmlDocPtr doc)

Create an xpath context for an xml document.

Parameters:

doc a pointer to the input xml document object.

Returns:

NULL on failure, or a pointer to the new xpath context on success.

Definition at line 919 of file xml.c.

References log_pedantic, and xmlXPathNewContext_d.

Referenced by http_page_get().

stringer_t* xml_dump_doc (xmlDocPtr doc)

Get an xml document object as a string.

Parameters:

doc a pointer to the xml document object to be serialized.

Returns:

NULL on failure or a pointer to a managed string containing the specified xml document's serialized data on success.

Definition at line 1020 of file xml.c.

References mm_free(), st_import(), and xmlDocDumpFormatMemory_d.

Referenced by contact_print_form(), contact_print_message(), portal_print_login(), statistics_process(), teacher_print_form(), and teacher_print_message().

xmlChar* xml_encode (xmlDocPtr doc, stringer_t * string)

Encode an xml string, performing proper replacement of defined entities and non-ASCII values.

Parameters:

doc a pointer to the xml document containing the DTD to be used for the encoding process.

string a managed string containing the xml data to be encoded.

NULL on failure, or a newly allocated null-terminated string containing the encoded xml data on success.

Definition at line 74 of file xml.c.

References st_data_get(), and xmlEncodeEntitiesReentrant_d.

Referenced by contact_print_form().

void xml_error (void * *ctx*, const chr_t * *format*, ...)

A function handler for displaying xml parser error messages to the user.

Parameters:

ctx a placeholder; ignored.

format a null-terminated string containing a format string for the error message to be displayed.

... a variable argument list containing the parameters to the format string.

Returns:

This function returns no value.

Definition at line 859 of file xml.c.

References log_mutex, mutex_lock(), and mutex_unlock().

Referenced by xml_create_parser_ctx().

void xml_free_doc (xmlDocPtr *doc*)

Free an xml document.

Parameters:

doc a pointer to the xml document to be freed.

Returns:

This function returns no value.

Definition at line 1003 of file xml.c.

References log_pedantic, and xmlFreeDoc_d.

Referenced by http_page_free().

void xml_free_parser_ctx (xmlParserCtxtPtr *ctx*)

Destroy an xml parser context.

Parameters:

ctx a pointer to the xml parser context to be freed.

Returns:

This function returns no value.

Definition at line 986 of file xml.c.

References log_pedantic, and xmlFreeParserCtxt_d.

Referenced by http_page_free().

void xml_free_xpath_ctx (xmlXPathContextPtr *ctx*)

Free an xml xpath context.

Parameters:

ctx a pointer to the xml xpath context to be freed.

Returns:

This function returns no value.

Definition at line 885 of file xml.c.

References `log_pedantic`, and `xmlXPathFreeContext_d`.

Referenced by `http_page_free()`.

void xml_free_xpath_obj (xmlXPathObjectPtr *obj*)

Free an xml xpath object.

Parameters:

obj a pointer to the xml xpath object to be freed.

Returns:

This function returns no value.

Definition at line 902 of file xml.c.

References `log_pedantic`, and `xmlXPathFreeObject_d`.

Referenced by `xml_set_xpath_ns()`, `xml_set_xpath_property()`, and `xml_set_xpath_uint64()`.

int16_t xml_get_xpath_int16 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as a signed 16-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as a signed 16-bit integer.

Definition at line 508 of file xml.c.

References `int16_conv_st()`, `log_pedantic`, `ns_length_get()`, `PLACER`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

int32_t xml_get_xpath_int32 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as a signed 32-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as a signed 32-bit integer.

Definition at line 453 of file xml.c.

References `int32_conv_st()`, `log_pedantic`, `ns_length_get()`, `PLACER`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

int64_t xml_get_xpath_int64 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as a signed 64-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as a signed 64-bit integer.

Definition at line 398 of file `xml.c`.

References `int64_conv_st()`, `log_pedantic`, `ns_length_get()`, `PLACER`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

int8_t xml_get_xpath_int8 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as a signed 8-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as a signed 8-bit integer.

Definition at line 563 of file `xml.c`.

References `int8_conv_st()`, `log_pedantic`, `ns_length_get()`, `PLACER`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

size_t xml_get_xpath_node_count (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

This function is not currently being called by any other part of the code.

Definition at line 822 of file `xml.c`.

References `log_pedantic`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

chr_t* xml_get_xpath_ns (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the content of an evaluated xpath expression as a null-terminated string.

Parameters:

ctx a pointer to the xpath context to be used for the evaluation.

xpath_query a null-terminated string containing the xpath expression to be evaluated.

Returns:

NULL on failure, or a null-terminated string containing the value of the xpath expression's specified node on success.

Definition at line 771 of file xml.c.

References `log_pedantic`, `ns_dupe()`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

stringer_t* xml_get_xpath_st (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as a managed string.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as a managed string.

Definition at line 618 of file xml.c.

References `log_pedantic`, `ns_length_get()`, `st_import()`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

uint16_t xml_get_xpath_uint16 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as an unsigned 16-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as an unsigned 16-bit integer.

Definition at line 288 of file xml.c.

References `log_pedantic`, `ns_length_get()`, `PLACER`, `uint16_conv_st()`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

uint32_t xml_get_xpath_uint32 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as an unsigned 32-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as an unsigned 32-bit integer.

Definition at line 234 of file xml.c.

References `log_pedantic`, `ns_length_get()`, `PLACER`, `uint32_conv_st()`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

uint64_t xml_get_xpath_uint64 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as an unsigned 64-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as an unsigned 64-bit integer.

Definition at line 179 of file xml.c.

References `log_pedantic`, `ns_length_get()`, `PLACER`, `uint64_conv_st()`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

uint8_t xml_get_xpath_uint8 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as an unsigned 8-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as an unsigned 8-bit integer.

Definition at line 343 of file xml.c.

References `log_pedantic`, `ns_length_get()`, `PLACER`, `uint8_conv_st()`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

xmlNodePtr xml_node_add_sibling (xmlNodePtr *current*, xmlNodePtr *element*)

Add an xml node to the list of siblings of another xml node.

Note:

If the sibling node was already inserted into a document it will first be unlinked.

Parameters:

current a pointer to the xml node to which the sibling node will be added.

element a pointer to the sibling node to be added to the target node.

Returns:

NULL on failure, or a pointer to the sibling node on success.

Definition at line 99 of file xml.c.

References `xmlAddSibling_d`.

Referenced by `contact_business_add_error()`, and `teacher_add_error()`.

void xml_node_free (xmlNodePtr *node*)

Free an xml node.

Parameters:

node a pointer to the xml node to be freed.

Returns:

This function returns no value.

Definition at line 62 of file xml.c.

References xmlFreeNode_d.

Referenced by contact_business_add_error(), and teacher_add_error().

stringer_t* xml_node_get_content_st (xmlNodePtr *node*)

Get the content of an xml node as a managed string.

Parameters:

node a pointer to the xml node to be queried.

Returns:

NULL on failure, or a pointer to a managed string

Definition at line 119 of file xml.c.

References log_pedantic, st_import(), xmlBufferContent_d, xmlBufferCreate_d, xmlBufferFree_d, xmlBufferLength_d, and xmlNodeBufGetContent_d.

xmlNodePtr xml_node_new (uchr_t * *name*)

Create an xml node.

Parameters:

name the name of the new xml node.

Returns:

NULL on failure, or a pointer to the newly allocated and initialized xml node on success.

Definition at line 109 of file xml.c.

References xmlNewNode_d.

Referenced by contact_business_add_error(), and teacher_add_error().

void xml_node_set_content (xmlNodePtr *node*, uchr_t * *content*)

Set the content of an xml node.

Parameters:

node a pointer to the xml node to be set.

content a null-terminated string containing the new content of the xml node.

Returns:

This function returns no value.

Definition at line 155 of file xml.c.

References xmlNodeSetContent_d.

Referenced by contact_business_add_error(), portal_print_login(), teacher_add_error(), xml_set_xpath_ns(), and xml_set_xpath_uint64().

xmlAttrPtr xml_node_set_property (xmlNodePtr *node*, uchr_t * *name*, uchr_t * *value*)

Set an attribute of an xml node.

Parameters:

node a pointer to the xml node to be set.

name a null-terminated string containing the name of the node attribute to be set.

value a null-terminated string containing the new value of the specified xml node's attribute.

Returns:

NULL on failure, or a pointer to node's attribute on success.

Definition at line 168 of file xml.c.

References xmlSetProp_d.

Referenced by contact_business_add_error(), teacher_add_error(), and xml_set_xpath_property().

bool_t xml_set_xpath_ns (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*, uchr_t * *val*)

Set the value of a node element selected by an xpath expression, as a null-terminated string.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

val the new value of the selected node, as a null-terminated string.

Returns:

true if the value was set successfully, or false on failure.

Definition at line 707 of file xml.c.

References log_pedantic, xml_free_xpath_obj(), xml_node_set_content(), and xml_xpath_eval().

Referenced by contact_print_form(), contact_print_message(), statistics_process(), teacher_print_form(), and teacher_print_message().

**bool_t xml_set_xpath_property (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*,
uchr_t * *name*, uchr_t * *val*)**

Set the value of a specified property of a node element selected by an xpath expression.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

name a null-terminated string containing the name of the node's property to be set.

val the new value of the selected node's property, as a null-terminated string.

Returns:

true if the selected node's value was set successfully, or false on failure.

Definition at line 740 of file xml.c.

References log_pedantic, xml_free_xpath_obj(), xml_node_set_property(), and xml_xpath_eval().

Referenced by contact_print_form(), contact_print_message(), and teacher_print_form().

**bool_t xml_set_xpath_uint64 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*,
uint64_t *val*)**

Set the value of a node element selected by an xpath expression, as a 64-bit unsigned integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

val the new value of the selected node, as a 64-bit unsigned integer.

Returns:

true if the value was set successfully, or false on failure.

Definition at line 673 of file xml.c.

References log_pedantic, xml_free_xpath_obj(), xml_node_set_content(), and xml_xpath_eval().

Referenced by statistics_process().

bool_t xml_start (void)

Initialize the xml parser library.

Returns:

This function returns no value.

Definition at line 1082 of file xml.c.

References xmlInitParser_d.

Referenced by process_start().

void xml_stop (void)

Cleanup the state and allocated memory of the xml parser in preparation to be shutdown.

Returns:

This function returns no value.

Definition at line 1069 of file xml.c.

References xmlCleanupGlobals_d, xmlCleanupParser_d, and xmlMemoryDump_d.

Referenced by process_stop().

xmlXPathObjectPtr xml_xpath_eval (const uchr_t * *xpath*, xmlXPathContextPtr *ctx*)

Evaluate an xml xpath expression.

Parameters:

a null-terminated string containing the xpath expression to be evaluated.

a pointer to the xpath context in which to perform the evaluation. NULL on failure, or a pointer to the xml path object of the evaluation on success.

Definition at line 943 of file xml.c.

References log_pedantic, and xmlXPathEvalExpression_d.

Referenced by contact_business_add_error(), portal_print_login(), teacher_add_error(), xml_set_xpath_ns(), xml_set_xpath_property(), and xml_set_xpath_uint64().

int_t xml_xpath_set_namespace (xmlXPathContextPtr *ctxt*, xmlChar * *prefix*, xmlChar * *ns_uri*)

Set the namespace for an xpath context.

See also:

xmlXPathRegisterNs()

Parameters:

ctxt a pointer to the specified xpath context.

prefix a pointer to the prefix of the namespace as a string.

ns_uri a pointer to the uri of the namespace as a string.

Returns:

-1 on failure or 0 on success.

Definition at line 87 of file xml.c.

References xmlXPathRegisterNs_d.

Referenced by http_page_get().

magma/core/parsers/special/bracket.c File Reference

Functions for extracting a value inside a pair of brackets.

```
#include "magma.h"
```

Functions

- **placer_t bracket_extract_pl** (void *block, size_t length)

Get a pointer to a block of data between a pair of brackets.

Detailed Description

Functions for extracting a value inside a pair of brackets.

Definition in file **bracket.c**.

Function Documentation

placer_t bracket_extract_pl (void * *block*, size_t *length*)

Get a pointer to a block of data between a pair of brackets.

Note:

The data is of format "[_data_]"

Parameters:

block a buffer containing the bracketed data.

length the length, in bytes, of the data buffer to be parsed.

Returns:

a null placer on failure, or a placer pointing to the data between the brackets on success.

Definition at line 22 of file bracket.c.

References data, log_pedantic, pl_init(), and pl_null().

Referenced by cache_config(), relay_config(), and servers_config().

magma/core/parsers/special/special.h File Reference

Declarations for functions involved in extracting data from specialized formats.

Functions

- **placer_t bracket_extract_pl** (void *block, size_t length)

Get a pointer to a block of data between a pair of brackets.

Detailed Description

Declarations for functions involved in extracting data from specialized formats.

Definition in file **special.h**.

Function Documentation

placer_t bracket_extract_pl (void * *block*, size_t *length*)

Get a pointer to a block of data between a pair of brackets.

Note:

The data is of format "[_data_]"

Parameters:

block a buffer containing the bracketed data.

length the length, in bytes, of the data buffer to be parsed.

Returns:

a null placer on failure, or a placer pointing to the data between the brackets on success.

Definition at line 22 of file bracket.c.

References data, log_pedantic, pl_init(), and pl_null().

Referenced by cache_config(), relay_config(), and servers_config().

magma/core/parsers/time.c File Reference

Functions used to parse time.

```
#include "magma.h"
```

Functions

- `uint64_t time_till_midnight` (void)
- *Get the number of seconds until midnight.* `uint64_t time_datestamp` (void)
- *Get the current date as an integer of the form YYYYMMDD.* `stringer_t * time_print_local` (`stringer_t *s`, `chr_t *format`, `time_t moment`)
- *Get a specified time as a formatted string.* `stringer_t * time_print_gmt` (`stringer_t *s`, `chr_t *format`, `time_t moment`)

Get a specified time as a formatted string converted to GMT.

Detailed Description

Functions used to parse time.

Definition in file `time.c`.

Function Documentation

`uint64_t time_datestamp` (void)

Get the current date as an integer of the form YYYYMMDD.

time.c

Returns:

0 on failure or a 64-bit unsigned integer containing the formatted current date on success.

Definition at line 41 of file `time.c`.

Referenced by `log_rotate()`, `log_start()`, `pattern_update()`, and `process_maint()`.

`stringer_t* time_print_gmt` (`stringer_t * s`, `chr_t * format`, `time_t moment`)

Get a specified time as a formatted string converted to GMT.

Parameters:

s a managed string where the formatted time is to be stored.

format a null-terminated string containing the format of the time string.

moment the specified time to be formatted, as a UNIX time value.

Returns:

NULL on failure, or a pointer to the managed string containing the result on success.

Definition at line 97 of file `time.c`.

References `log_pedantic`, `mm_wipe()`, `st_avail_get()`, `st_char_get()`, and `st_length_set()`.

Referenced by `http_response_cookie()`, `http_response_header()`, and `http_response_options()`.

stringer_t* time_print_local (stringer_t * s, chr_t * *format*, time_t *moment*)

Get a specified time as a formatted string.

Parameters:

s a managed string where the formatted time is to be stored.

format a null-terminated string containing the format of the time string.

moment the specified time to be formatted, as a UNIX time value.

Returns:

NULL on failure, or a pointer to the managed string containing the result on success.

Definition at line 66 of file time.c.

References log_pedantic, mm_wipe(), st_avail_get(), st_char_get(), and st_length_set().

Referenced by imap_id().

uint64_t time_till_midnight (void)

Get the number of seconds until midnight.

Returns:

an unsigned 64-bit integer containing the number of seconds until midnight, or 86400 on failure.

Definition at line 19 of file time.c.

Referenced by process_maint().

magma/core/parsers/token.c File Reference

Functions for tokenizing strings.

```
#include "magma.h"
```

Functions

- `uint64_t tok_get_count_bl` (void *block, size_t length, char token)
- *Count the number of times a token is found in a specified block of memory.* `uint64_t tok_get_count_st` (`stringer_t` *string, char token)
- *Count the number of times a token is found in a managed string.* `int tok_get_ns` (char *string, size_t length, char token, `uint64_t` fragment, `placer_t` *value)
- *Retrieve a specified token from a null-terminated string.* `int tok_get_bl` (void *block, size_t length, char token, `uint64_t` fragment, `placer_t` *value)
- *Retrieve a specified token from a block of data.* `int tok_get_st` (`stringer_t` *string, char token, `uint64_t` fragment, `placer_t` *value)
- *Retrieve a specified token from a managed string.* `int tok_get_pl` (`placer_t` string, char token, `uint64_t` fragment, `placer_t` *value)
- *Retrieve a specified token from a placer.* `void tok_pop_init_bl` (`tok_state_t` *state, void *block, size_t length, char token)
- `void tok_pop_init_st` (`tok_state_t` *state, `stringer_t` *string, char token)
- `int tok_pop` (`tok_state_t` *state, `placer_t` *value)
- `uint64_t str_tok_get_count_bl` (void *block, size_t length, `chr_t` *token, size_t toklen)
- *Count the number of times a string token is found in a specified block of memory.* `int str_tok_get_bl` (char *block, size_t length, `chr_t` *token, size_t toklen, `uint64_t` fragment, `placer_t` *value)
- *Retrieve a specified string-split token from a null-terminated string.* `bool_t pl_skip_characters` (`placer_t` *place, char *skipchars, size_t nchars)
- *Skip past any of the specified characters found at the beginning of the placer, and update the placer accordingly.* `bool_t pl_skip_to_characters` (`placer_t` *place, char *skiptochars, size_t nchars)
- *Skip to the first instance of any of the specified characters in the placer, and update the placer accordingly.* `bool_t pl_shrink_before_characters` (`placer_t` *place, char *shrinkchars, size_t nchars)
- *Truncate a placer to start before any of the specified characters, and update the placer accordingly.* `bool_t pl_get_embraced` (`placer_t` str, `placer_t` *out, unsigned char opening, unsigned char closing, `bool_t` required)
- *Get a placer pointing to the text contained within a specified pair of opening and closing braces.* `bool_t pl_update_start` (`placer_t` *place, size_t nchars, `bool_t` more)

Ensure that a placer contains at least a certain amount of data, and update it accordingly.

Detailed Description

Functions for tokenizing strings.

Definition in file `token.c`.

Function Documentation

`bool_t pl_get_embraced` (`placer_t` str, `placer_t` * out, unsigned char opening, unsigned char closing, `bool_t` required)

Get a placer pointing to the text contained within a specified pair of opening and closing braces.

Parameters:

str a placer pointing to the string to be parsed.
out a pointer to a placer that will store the string between the braces on success.
opening the character to be the opening brace of the sequence.
closing the character to be the closing brace of the sequence.
required if true, fail if no data was found between the pair of braces.

Returns:

true if the brace sequence was complete, or false if either component could not be located.
Definition at line 455 of file token.c.
References `pl_char_get()`, and `pl_empty()`.

`bool_t pl_shrink_before_characters (placer_t * place, char * shrinkchars, size_t nchars)`

Truncate a placer to start before any of the specified characters, and update the placer accordingly.

Parameters:

place a pointer to a placer that will be updated to be truncated before any of the specified characters.
shrinkchars a pointer to a buffer containing bytes that will be skipped when they are found at the end of the placer.
nchars the number of characters to be tested in the collection in *shrinkchars*.

Returns:

true if the shrink operation completed before the end of the placer was reached, or false otherwise.
Definition at line 418 of file token.c.
References `pl_char_get()`, `pl_empty()`, and `pl_length_get()`.
Referenced by `portal_upload()`.

`bool_t pl_skip_characters (placer_t * place, char * skipchars, size_t nchars)`

Skip past any of the specified characters found at the beginning of the placer, and update the placer accordingly.

Parameters:

place a pointer to a placer that will be updated to skip past any of the specified characters.
skipchars a pointer to a buffer containing bytes that will be skipped at the beginning of the placer.
nchars the number of characters to be tested in the collection in *skipchars*.

Returns:

true if the skip operation completed before the end of the placer was reached, or false otherwise.
Definition at line 349 of file token.c.
References `pl_char_get()`, and `pl_empty()`.
Referenced by `portal_upload()`.

`bool_t pl_skip_to_characters (placer_t * place, char * skiptochars, size_t nchars)`

Skip to the first instance of any of the specified characters in the placer, and update the placer accordingly.

Parameters:

place a pointer to a placer that will be updated to skip to any of the specified characters.
skiptochars a pointer to a buffer containing bytes that will be skipped to when they are first found in the placer.
nchars the number of characters to be tested in the collection in *skiptochars*.

Returns:

true if the skip operation completed before the end of the placer was reached, or false otherwise.
 Definition at line 385 of file token.c.
 References `pl_char_get()`, and `pl_empty()`.
 Referenced by `portal_upload()`.

bool_t pl_update_start (placer_t * *place*, size_t *nchars*, bool_t *more*)

Ensure that a placer contains at least a certain amount of data, and update it accordingly.

Parameters:

place a pointer to the placer containing the original data, which will be updated on success.
nchars the minimum number of characters that the placer needs to contain to pass the test.
more if true, more characters following the minimum buffer size are necessary.

Returns:

true if the placer meets the minimum size check, or false if it does not.
 Definition at line 498 of file token.c.

int str_tok_get_bl (char * *block*, size_t *length*, chr_t * *token*, size_t *toklen*, uint64_t *fragment*, placer_t * *value*)

Retrieve a specified string-split token from a null-terminated string.

Parameters:

block a pointer to the block of memory to be tokenized.
length the maximum number of characters to be scanned from the input data.
token the token string that will be used to split the data.
toklen the length, in bytes, of the token string.
fragment the zero-indexed token number to be extracted from the data.
value a pointer to a placer that will receive the value of the extracted token on success, or `pl_null()` on failure.

Returns:

-1 on failure, 0 on success, or 1 if the token was extracted successfully, but was the last one in the string.
 Definition at line 303 of file token.c.
 References `mm_empty()`, `pl_char_get()`, `pl_init()`, `pl_length_get()`, `pl_null()`, `placer_t`, and `st_search_cs()`.
 Referenced by `portal_upload()`.

uint64_t str_tok_get_count_bl (void * *block*, size_t *length*, chr_t * *token*, size_t *toklen*)

Count the number of times a string token is found in a specified block of memory.

Parameters:

block a pointer to a block of memory to be scanned.
length the length, in bytes, of the block of memory to be scanned.
token a pointer to the string token being used to split the input data.
tokenlen the length, in bytes, of the specified token.

Returns:

the number of times the string token was found in the block of memory, or 0 on failure.
 Definition at line 270 of file token.c.
 References count, mm_empty(), pl_init(), pl_length_get(), placer_t, and st_search_cs().
 Referenced by portal_upload().

int tok_get_bl (void * *block*, size_t *length*, char *token*, uint64_t *fragment*, placer_t * *value*)

Retrieve a specified token from a block of data.

See also:

tok_get_ns()

Parameters:

block a pointer to the memory block to be tokenized.
length the maximum number of characters to be scanned at the input data block.
token the token character that will be used to split the data block.
fragment the zero-indexed token number to be extracted from the data block.
value a pointer to a placer that will receive the value of the extracted token on success, or **pl_null()** on failure.

Returns:

-1 on failure, 0 on success, or 1 if the token was extracted successfully, but was the last one found in the data block.
 Definition at line 141 of file token.c.
 References tok_get_ns().
 Referenced by nvp_parse(), smtp_parse_mail_from_path(), tok_get_pl(), and tok_get_st().

uint64_t tok_get_count_bl (void * *block*, size_t *length*, char *token*)

Count the number of times a token is found in a specified block of memory.

Parameters:

block a pointer to a block of memory to be scanned.
length the length, in bytes, of the block of memory to be scanned.
token a character specifying the token to be searched.

Returns:

the number of times the token was found in the block of memory, or 0 on failure.
 Definition at line 23 of file token.c.
 References count, log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, and mm_empty().
 Referenced by smtp_parse_mail_from_path(), and tok_get_count_st().

uint64_t tok_get_count_st (stringer_t * *string*, char *token*)

Count the number of times a token is found in a managed string.

See also:

tok_get_count_bl()

Parameters:

string a pointer to the managed string to be scanned.
token a character specifying the token to be searched.

Returns:

the number of times the token was found in the managed string, or 0 on failure.

Definition at line 58 of file token.c.

References st_data_get(), st_length_get(), and tok_get_count_bl().

Referenced by get_header_opt(), get_header_value_noopt(), http_parse_method(), http_parse_pairs(), imap_fetch_body_part(), imap_narrow_messages(), imap_search_messages_date(), imap_search_messages_date_compare(), imap_search_messages_range(), ip_str_subnet(), mail_domain_get(), mail_mime_type_parameters(), portal_get_upload_attachment(), and stamp_counter_check().

int tok_get_ns (char * *string*, size_t *length*, char *token*, uint64_t *fragment*, placer_t * *value*)

Retrieve a specified token from a null-terminated string.

Parameters:

string a pointer to the null-terminated string to be tokenized.
length the maximum number of characters to be scanned from the input string.
token the token character that will be used to split the string.
fragment the zero-indexed token number to be extracted from the string.
value a pointer to a placer that will receive the value of the extracted token on success, or **pl_null()** on failure.

Returns:

-1 on failure, 0 on success, or 1 if the token was extracted successfully, but was the last one in the string.

Definition at line 72 of file token.c.

References log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, mm_empty(), pl_init(), and pl_null().

Referenced by ip_str_subnet(), lib_load_openssl(), and tok_get_bl().

int tok_get_pl (placer_t *string*, char *token*, uint64_t *fragment*, placer_t * *value*)

Retrieve a specified token from a placer.

See also:

token_get_ns()

Parameters:

string a pointer to the placer to be tokenized.

token the token character that will be used to split the placer.

fragment the zero-indexed token number to be extracted from the placer.

value a pointer to a placer that will receive the value of the extracted token on success, or **pl_null()** on failure.

Returns:

-1 on failure, 0 on success, or 1 if the token was extracted successfully, but was the last one found in the placer.

Definition at line 169 of file token.c.

References `pl_data_get()`, `pl_length_get()`, and `tok_get_bl()`.

Referenced by `http_data_value_parse()`, `http_parse_context()`, `http_parse_method()`, and `smtp_parse_mail_from_path()`.

int tok_get_st (stringer_t * *string*, char *token*, uint64_t *fragment*, placer_t * *value*)

Retrieve a specified token from a managed string.

See also:

tok_get_ns()

Parameters:

string a pointer to the managed string to be tokenized.

token the token character that will be used to split the managed string.

fragment the zero-indexed token number to be extracted from the managed string.

value a pointer to a placer that will receive the value of the extracted token on success, or **pl_null()** on failure.

Returns:

-1 on failure, 0 on success, or 1 if the token was extracted successfully, but was the last one found in the managed string.

Definition at line 155 of file token.c.

References `st_data_get()`, `st_length_get()`, and `tok_get_bl()`.

Referenced by `get_header_opt()`, `get_header_value_noopt()`, `http_parse_context()`, `http_parse_pairs()`, `imap_fetch_body_part()`, `imap_folder_create()`, `imap_folder_remove()`, `imap_folder_rename()`, `imap_narrow_messages()`, `imap_parse_address_part()`, `imap_search_messages_date()`, `imap_search_messages_date_compare()`, `imap_search_messages_range()`, `mail_domain_get()`, `mail_mime_type_parameters()`, `portal_get_upload_attachment()`, `smtp_auth_plain()`, and `stamp_counter_check()`.

int tok_pop (tok_state_t * *state*, placer_t * *value*)

on error the function returns -1 and sets value to EMPTY_PLACER on success the function returns 0 and stores the token in the placer pointed at by value if the end is hit, then the function returns 1, and places any remaining data in value

Definition at line 201 of file token.c.

References `pl_init()`, `pl_null()`, `tok_state_t::position`, `tok_state_t::remaining`, and `tok_state_t::token`.

Referenced by `nvp_parse()`.

void tok_pop_init_bl (tok_state_t * *state*, void * *block*, size_t *length*, char *token*)

Definition at line 174 of file token.c.

References tok_state_t::block, tok_state_t::length, tok_state_t::position, tok_state_t::remaining, and tok_state_t::token.

void tok_pop_init_st (tok_state_t * *state*, stringer_t * *string*, char *token*)

Definition at line 185 of file token.c.

References tok_state_t::block, tok_state_t::length, tok_state_t::position, tok_state_t::remaining, st_char_get(), st_length_get(), and tok_state_t::token.

Referenced by nvp_parse().

magma/core/parsers/trim.c File Reference

Functions used to trim whitespace from strings.

```
#include "magma.h"
```

Defines

- #define **CHARS_TO_TRIM** ' ', '\n', '\r', '\t', '\v'

Functions

- void **st_trim** (**stringer_t** *string)
- **placer_t** **pl_trim** (**placer_t** place)
- *trim.c* **placer_t** **pl_trim_start** (**placer_t** place)
- Trim the leading whitespace from a placer. **placer_t** **pl_trim_end** (**placer_t** place)

Detailed Description

Functions used to trim whitespace from strings.

Definition in file **trim.c**.

Define Documentation

#define CHARS_TO_TRIM ' ', '\n', '\r', '\t', '\v'

Definition at line 15 of file trim.c.

Referenced by **pl_trim()**, **pl_trim_end()**, **pl_trim_start()**, and **st_trim()**.

Function Documentation

placer_t **pl_trim** (**placer_t** *place*)

trim.c

Definition at line 67 of file trim.c.

References **CHARS_TO_TRIM**, **pl_char_get()**, **pl_init()**, **pl_length_get()**, and **pl_null()**.

Referenced by **nvp_parse()**, and **smtp_parse_mail_from_path()**.

placer_t **pl_trim_end** (**placer_t** *place*)

Definition at line 144 of file trim.c.

References **CHARS_TO_TRIM**, **pl_char_get()**, **pl_init()**, **pl_length_get()**, and **pl_null()**.

Referenced by **smtp_parse_helo_domain()**, **smtp_parse_mail_from_path()**, and **smtp_parse_rcpt_to()**.

placer_t pl_trim_start (placer_t *place*)

Trim the leading whitespace from a placer.

Parameters:

place a placer containing the string to have its leading whitespace trimmed.

Returns:

a placer pointing to the trimmed value inside the originally specified input string.

Definition at line 115 of file trim.c.

References CHARS_TO_TRIM, pl_char_get(), pl_init(), pl_length_get(), and pl_null().

Referenced by get_header_opt().

void st_trim (stringer_t * *string*)

Definition at line 18 of file trim.c.

References CHARS_TO_TRIM, mm_move(), mm_wipe(), st_avail_get(), st_char_get(), st_length_get(), and st_length_set().

magma/core/strings/allocation.c File Reference

Functions used to allocate stringers.

```
#include "magma.h"
```

Functions

- void **st_free** (**stringer_t** *s)
- *Free a managed string.* void **st_cleanup** (**stringer_t** *s)
- *A checked managed string free front-end function.* **stringer_t** * **st_merge_opts** (uint32_t opts, **chr_t** *format,...)
- *Concatenate a variable number of user-supplied multi-type strings.* **stringer_t** * **st_import** (const void *s, size_t len)
- *Create a new (contiguous managed) managed string on the heap to hold a copy of the specified data buffer.* **stringer_t** * **st_copy_in** (**stringer_t** *s, void *buf, size_t len)
- *Copy data into a managed string.* **stringer_t** * **st_dupe_opts** (uint32_t opts, **stringer_t** *s)
- *Create a duplicate copy of a managed string with a specified set of allocation options.* **stringer_t** * **st_dupe** (**stringer_t** *s)
- *Duplicate a managed string.* **stringer_t** * **st_append_opts** (size_t align, **stringer_t** *s, **stringer_t** *append)
- *Append one managed string to another, with aligned memory boundaries.* **stringer_t** * **st_alloc_opts** (uint32_t opts, size_t len)
- *Allocate a managed string with a specified options mask.* **stringer_t** * **st_realloc** (**stringer_t** *s, size_t len)
- *Reallocate a managed string to a specified size.* **stringer_t** * **st_output** (**stringer_t** *output, size_t len)
- *Return a suitable managed string to store data of a specified length.* **stringer_t** * **st_nullify** (**chr_t** *input, size_t len)

Create a null-terminated string out of a block of memory and return it as a newly allocated nuller.

Detailed Description

Functions used to allocate stringers.

\$Author\$ \$Date\$ \$Revision:\$

Definition in file **allocation.c**.

Function Documentation

stringer_t* st_alloc_opts (uint32_t opts, size_t len)

Allocate a managed string with a specified options mask.

See also:

st_valid_options()

Note:

All requested allocation masks should conform to the validation imposed by **st_valid_opts()**. The supported types are: placer, nuller, block, managed, and mapped. The following logic is applied to requested string allocation options: 1. Any allocation options specified for strings to be allocated on the stack are IGNORED. 2. All jointed strings allocate memory for the header and data separately and then link them EXCEPT: Jointed placers only allocate space for a header. Jointed mapped strings allocate the data with an aligned mmap() operation. 3. Contiguous strings allocate space for the header and data together in a

single contiguous block. 4. After allocation, the length field is set for block strings. 5. After allocation, the available field is set for managed strings. 6. Placers can only be jointed; mapped strings can only be contiguous.

Parameters:

opts the allocation options mask to be used by the newly allocated string.
len the length, in bytes, of the managed string to be allocated.

Returns:

NULL on failure or a pointer to the newly allocated managed string on success.

Definition at line 414 of file allocation.c.

References `align()`, `block_t`, `BLOCK_T`, `CONTIGUOUS`, `HEAP`, `JOINTED`, `log_pedantic`, `magma`, `MAGMA_SPOOL_DATA`, `managed_t`, `MANAGED_T`, `mapped_t`, `MAPPED_T`, `MEMORYBUF`, `mm_alloc()`, `mm_free()`, `mm_sec_alloc()`, `mm_sec_free()`, `mm_set()`, `nuller_t`, `NULLER_T`, `magma_t::page_length`, `placer_t`, `PLACER_T`, `SECURE`, `spool_mktemp()`, `st_info_opts()`, `st_valid_opts()`, and `STACK`.

Referenced by `base64_decode_opts()`, `base64_encode_opts()`, `credential_address()`, `credential_alloc_auth()`, `credential_alloc_mail()`, `dkim_create()`, `ecies_key_private_hex()`, `http_body()`, `mail_add_forward_headers()`, `mail_message_cleanup()`, `mail_mime_split()`, `meta_data_user_build_storage_keys()`, `qp_encode()`, `register_data_insert_user()`, `smtp_check_greylis()`, `smtp_data_read()`, `st_append_opts()`, `st_dupe_opts()`, `st_import()`, `st_merge_opts()`, `st_nullify()`, `st_vaprint_opts()`, and `url_encode()`.

stringer_t* st_append_opts (size_t align, stringer_t * s, stringer_t * append)

Append one managed string to another, with aligned memory boundaries.

Parameters:

align an alignment value to be used for managed string (re)allocation.
s the managed string to be extended, which will be allocated for the caller if passed as NULL.
append the managed string to be appended to *s*.

Returns:

NULL on failure, or a pointer to the appended result on success.

Definition at line 363 of file allocation.c.

References `HEAP`, `JOINTED`, `log_pedantic`, `MANAGED_T`, `mm_copy()`, `st_alloc_opts()`, `st_avail_get()`, `st_data_get()`, `st_length_get()`, `st_length_set()`, `st_realloc()`, and `st_valid_append()`.

Referenced by `http_body()`, `imap_id()`, `imap_search()`, `imap_status()`, and `mail_add_forward_headers()`.

void st_cleanup (stringer_t * s)

A checked managed string free front-end function.

See also:

`st_free()`

Parameters:

s the managed string to be freed.

Returns:

This function returns no value.

Definition at line 98 of file allocation.c.

References `st_free()`.

Referenced by `client_close()`, `con_destroy()`, `credential_alloc_auth()`, `credential_free()`, `dkim_create()`, `http_data_free()`, `http_free_content()`, `http_print_301()`, `http_response_header()`, `http_response_options()`, `http_session_reset()`, `imap_command_parser()`, `imap_fetch_body()`, `imap_fetch_body_header()`, `imap_fetch_body_mime()`, `imap_fetch_body_tag()`, `imap_fetch_bodystructure()`, `imap_fetch_envelope()`, `imap_fetch_response_free()`, `imap_folder_create()`, `imap_folder_rename()`, `imap_login()`, `imap_session_destroy()`, `magma_folder_find_full_name()`, `mail_add_forward_headers()`, `mail_add_outbound_headers()`, `mail_add_required_headers()`, `mail_build_signature()`, `mail_cache_destroy()`, `mail_cache_reset()`, `mail_cache_set()`, `mail_destroy()`, `mail_destroy_header()`, `mail_destroy_message()`, `mail_get_boundary()`, `mail_insert_chunk_base64()`, `mail_mime_boundary()`, `mail_mime_free()`, `mail_mime_get_smtp_envelope()`, `message_free()`, `meta_data_fetch_user()`, `meta_data_user_build_storage_keys()`, `meta_get()`, `meta_user_build()`, `meta_user_destroy()`, `pop_session_destroy()`, `pop_user()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_messages_list()`, `portal_folder_contacts_add()`, `portal_folder_mail_add()`, `portal_folder_mail_remove()`, `portal_message_attachments()`, `portal_message_body()`, `portal_message_header()`, `portal_smtp_merge_headers()`, `register_business_step2()`, `register_data_insert_user()`, `register_session_cache()`, `register_session_free()`, `sess_destroy()`, `sess_get()`, `sess_release_attachment()`, `sess_token()`, `smtp_auth_login()`, `smtp_auth_plain()`, `smtp_bounce()`, `smtp_check_filters()`, `smtp_check_greylist()`, `smtp_ehlo()`, `smtp_free_inbound()`, `smtp_free_outbound()`, `smtp_free_recipients()`, `smtp_helo()`, `smtp_list_free_filter()`, `smtp_reply()`, `smtp_session_destroy()`, `smtp_session_reset()`, `st_info_opts()`, `st_merge_opts()`, `teacher_data_delete()`, `teacher_data_free()`, and `teacher_process()`.

`stringer_t* st_copy_in (stringer_t * s, void * buf, size_t len)`

Copy data into a managed string.

Parameters:

s the managed string to store the copied contents of the data.
buf a pointer to the buffer containing the data to be copied.
len the length of the data to be copied.

Returns:

NULL on failure, or a pointer to the managed string on success.

Definition at line 235 of file `allocation.c`.

References `log_pedantic`, `mm_copy()`, `st_avail_get()`, `st_data_get()`, and `st_length_set()`.

Referenced by `credential_alloc_auth()`, `ecies_key_private_hex()`, and `meta_data_user_build_storage_keys()`.

`stringer_t* st_dupe (stringer_t * s)`

Duplicate a managed string.

See also:

`st_dupe_opts()`

Note:

The allocation options of the duplicated string will be the same as that of the source string.

Parameters:

s the managed string to be duplicated.

Returns:

NULL on failure, or a copy of the input managed string on success.

Definition at line 349 of file `allocation.c`.

References `st_dupe_opts()`.

Referenced by `ar_dupe()`, `contact_folder_rename()`, `get_temp_file_handle()`, `http_load_file()`, `http_print_301()`, `imap_folder_create()`, `imap_folder_rename()`, `mail_add_forward_headers()`, `mail_add_outbound_headers()`, `mail_cache_get()`, `meta_data_user_build()`, `meta_user_build()`, `register_business_step1()`, `register_business_step2()`, `register_print_message()`, `register_print_step1()`, `register_print_step2()`, `register_print_step3()`, `register_session_get()`, `smtp_add_recipient()`, `smtp_data_outbound()`, `smtp_fetch_inbound()`, `smtp_forward_message()`, and `teacher_process()`.

stringer_t* st_dupe_opts (uint32_t opts, stringer_t * s)

Create a duplicate copy of a managed string with a specified set of allocation options.

See also:

`st_valid_opts()`

Note:

Both the source and destination managed strings must have option values that pass `st_valid_opts()`. The following procedure occurs when creating the duplicate managed string: If the destination is a placer, 0 bytes are allocated for the duplicate's underlying data. If the destination is a constant, nuller, or block, its underlying data buffer will be of equal size to the source's data length. If the destination is managed or mapped, and so is the source, its underlying data buffer will be of equal size to the source's available size; but if the source is neither, the underlying data buffer of the destination managed or mapped string will equal the size of the source's data length. A deep copy will be made of the source data to the destination if the destination is NOT a placer.

Parameters:

opts the allocation options mask to be used for the newly created managed string.
s the target managed string to be duplicated.

Returns:

NULL on failure or a pointer to a copy of the duplicated managed string on success.

Definition at line 275 of file `allocation.c`.

References `BLOCK_T`, `CONSTANT_T`, `log_pedantic`, `MANAGED_T`, `MAPPED_T`, `MEMORYBUF`, `mm_copy()`, `NULLER_T`, `PLACER_T`, `st_alloc_opts()`, `st_avail_get()`, `st_data_get()`, `st_data_set()`, `st_info_opts()`, `st_length_get()`, `st_length_set()`, `st_valid_opts()`, and `st_valid_tracked()`.

Referenced by `args_parse()`, `cache_set_value()`, `config_value_set()`, `get_temp_file_handle()`, `http_data_value_parse()`, `http_load_file()`, `http_parse_header()`, `http_parse_method()`, `http_response_connection()`, `imap_fetch_response_add()`, `imap_folder_create()`, `imap_folder_rename()`, `imap_login()`, `magma_folder_name()`, `mail_add_forward_headers()`, `mail_cache_set()`, `mt_dupe()`, `portal_message_body()`, `relay_set_value()`, `servers_set_value()`, `sess_create()`, `smtp_fetch_inbound()`, `smtp_relay_message()`, and `st_dupe()`.

void st_free (stringer_t * s)

Free a managed string.

See also:

`st_valid_free()`, `st_valid_opts()`

Note:

Any managed string to be freed should conform with the validity checks enforced by `st_valid_free()/st_valid_opts()` *NO* managed string should be passed to `st_free()` if it was allocated on

the stack, or contains foreign data. The following logic is carried out depending on the allocation options of the string to be freed: Jointed placer: Free header, if secure or on heap Free underlying data if it is not foreign Jointed nuller: Free header and underlying data Jointed block: Free header and underlying data Jointed managed: Free header and underlying data Jointed mapped: Free the header and (munmap) underlying data Contiguous nuller: Free header (this includes the underlying data because they are already merged) Contiguous block: Free header (this includes the underlying data because they are already merged) Contiguous managed: Free header (this includes the underlying data because they are already merged)

Parameters:

s the managed string to be freed.

Returns:

This function returns no value.

HIGH: Finish this logic out. Stack allocations are skipped, unless they are jointed. In that case the data is freed unless it carries a foreigner flag. Note its possible for a jointed string to have a NULL data reference. We should be picking up on that too. Contiguous heap buffers are just destroyed.

Do we need to differentiate between stack structures, and heap data blocks needing to be freed, or not?

Definition at line 34 of file allocation.c.

References block_t, BLOCK_T, CONTIGUOUS, data, debug_hook(), FOREIGNDATA, HEAP, JOINTED, length, log_pedantic, managed_t, MANAGED_T, mapped_t, MAPPED_T, MEMORYBUF, mm_free(), mm_sec_free(), nuller_t, NULLER_T, placer_t, PLACER_T, SECURE, st_info_opts(), and st_valid_free().

Referenced by ar_free(), base64_decode_opts(), base64_encode_opts(), cache_free(), cache_set_value(), client_print(), compress_bzip(), compress_lzo(), compress_zlib(), config_free(), config_load_cmdline_settings(), config_load_file_settings(), config_value_set(), contact_business(), contact_folder_rename(), contact_free(), contact_print_form(), contact_print_message(), credential_alloc_auth(), decompress_block_lzo(), decompress_bzip(), decompress_lzo(), decompress_zlib(), digest_hash(), dspam_check(), dspam_train(), ecies_key_private_hex(), get_temp_file_handle(), http_content_load_directory(), http_content_load_fonts(), http_content_refresh(), http_content_start(), http_data_header_parse_line(), http_data_value_parse(), http_load_file(), http_print_301(), imap_build_array(), imap_command_parser(), imap_fetch_body(), imap_fetch_body_tag(), imap_fetch_bodystructure(), imap_fetch_response_add(), imap_folder_create(), imap_folder_name_escaped(), imap_folder_rename(), imap_id(), imap_list(), imap_lsub(), imap_narrow_folders(), imap_parse_address(), imap_parse_address_part(), imap_parse_literal(), imap_range_build(), imap_search(), imap_search_messages_body(), imap_search_messages_date(), imap_search_messages_header(), imap_search_messages_text(), imap_status(), ip_presentation(), ip_standard(), ip_subnet(), mail_add_forward_headers(), mail_add_outbound_headers(), mail_add_required_headers(), mail_build_signature(), mail_create_message(), mail_discover_encoding(), mail_discover_insertion_point(), mail_discover_type(), mail_extract_tag(), mail_header_fetch_all(), mail_insert_chunk_base64(), mail_insert_chunk_text(), mail_load_header(), mail_load_message(), mail_message_cleanup(), mail_mime_content_encoding(), mail_mime_content_id(), mail_mime_encode_part(), mail_mime_encoding(), mail_mime_generate_boundary(), mail_mime_get_smtp_envelope(), mail_mime_split(), mail_mime_type(), mail_mime_type_group(), mail_mime_type_parameters(), mail_mime_type_sub(), mail_mod_subject(), mail_modify_part(), meta_data_fetch_all_tags(), meta_data_user_build(), meta_data_user_build_storage_keys(), meta_data_user_save_storage_keys(), meta_folders_by_name(), meta_folders_name(), meta_get(), mt_free(), nvp_alloc(), nvp_parse(), pop_pass(), pop_user(), portal_endpoint_attachments_progress(), portal_endpoint_folders_add(), portal_endpoint_folders_rename(), portal_endpoint_messages_list(), portal_endpoint_messages_send(), portal_endpoint_search(), portal_parse_json_str_array(), portal_print_login(), portal_smtp_create_data(), protocol_secure(), rand_choices(), register_data_fetch_blocklist(), register_data_insert_user(), register_print_captcha(), register_print_message(), register_print_step1(), register_print_step2(), register_print_step3(), register_session_cache(), register_session_get(), relay_free(), relay_set_value(), sanity_check(), scramble_decrypt(), scramble_encrypt(), serial_get(), serial_increment(), serial_reset(), servers_free(), servers_set_value(), sess_get(),

smtp_accept_message(), smtp_auth_login(), smtp_auth_plain(), smtp_bounce(), smtp_check_receive_quota(), smtp_data(), smtp_data_outbound(), smtp_data_read(), smtp_forward_message(), smtp_mail_from(), smtp_rcpt_to(), smtp_reply(), smtp_rollout(), smtp_update_receive_stats(), spool_cleanup(), spool_mktemp(), spool_start(), spool_stop(), ssl_start(), st_cleanup(), st_replace(), st_vaprint_opts(), stamp_counter_check(), statistics_process(), symmetric_decrypt(), symmetric_encrypt(), teacher_add_cookie(), teacher_data_get(), teacher_data_save(), teacher_print_form(), teacher_print_message(), virus_stop(), warehouse_fetch_patterns(), and zbase32_decode().

stringer_t* st_import (const void * s, size_t len)

Create a new (contiguous managed) managed string on the heap to hold a copy of the specified data buffer.

Parameters:

s the address of the buffer to be duplicated.
len the length, in bytes, of the copied buffer.

Returns:

NULL on failure, or a pointer to the newly allocated managed string on success.

Definition at line 214 of file allocation.c.

References CONTIGUOUS, HEAP, MANAGED_T, mm_copy(), st_alloc_opts(), st_data_get(), and st_length_set().

Referenced by cache_get(), con_reverse_lookup(), dspam_check(), ecies_key_public_hex(), get_temp_file_handle(), http_data_header_parse_line(), http_load_file(), imap_fetch_body(), imap_fetch_body_tag(), imap_fetch_bodystructure(), imap_fetch_message(), imap_fetch_return_header(), imap_parse_astring(), imap_parse_nstring(), imap_search_messages_date(), mail_add_outbound_headers(), mail_add_required_headers(), mail_header_fetch_all(), mail_load_header(), mail_load_message(), mail_mime_content_encoding(), mail_mime_content_id(), mail_mime_type_group(), mail_mime_type_parameters_key(), mail_mime_type_parameters_value(), mail_mime_type_sub(), meta_data_user_build_storage_keys(), meta_data_user_save_storage_keys(), meta_folders_name(), nvp_parse(), pop_pass_parse(), pop_user_parse(), portal_endpoint_attachments_add(), portal_endpoint_folders_add(), portal_endpoint_folders_rename(), portal_endpoint_messages_list(), portal_parse_json_str_array(), portal_upload(), register_captcha_generate(), res_field_string(), servers_network_start(), smtp_parse_auth(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(), smtp_parse_rcpt_to(), tank_load(), teacher_add_cookie(), xml_dump_doc(), xml_get_xpath_st(), and xml_node_get_content_st().

stringer_t* st_merge_opts (uint32_t opts, chr_t * format, ...)

Concatenate a variable number of user-supplied multi-type strings.

Parameters:

opts the options value of the resulting managed string that will be allocated for the caller.
format a format string consisting of the characters 's' for specifying that the next variable argument will be a managed string or 'n' if it is a null-terminated string.
... a variable argument list containing the strings to be concatenated to one another.

Returns:

a newly allocated string containing the concatenations of all specified managed and null-terminated strings.

Definition at line 115 of file allocation.c.

References `length`, `log_pedantic`, `MEMORYBUF`, `mm_copy()`, `ns_empty()`, `ns_length_get()`, `st_alloc_opts()`, `st_char_get()`, `st_cleanup()`, `st_data_get()`, `st_empty()`, `st_info_opts()`, `st_length_get()`, `st_length_set()`, `st_valid_destination()`, and `st_valid_tracked()`.

Referenced by `credential_alloc_auth()`, `mail_add_forward_headers()`, `mail_add_inbound_headers()`, `mail_add_outbound_headers()`, `mail_add_required_headers()`, and `spool_path()`.

`stringer_t* st_nullify (chr_t * input, size_t len)`

Create a null-terminated string out of a block of memory and return it as a newly allocated nuller.

Parameters:

input a pointer to the data block to be copied.

len the size, in bytes, of the data block to be copied before being terminated with a null byte.

Returns:

NULL on failure, or a pointer to the new null-terminated string on success.

Definition at line 719 of file `allocation.c`.

References `CONTIGUOUS`, `HEAP`, `MANAGED_T`, `mm_copy()`, `st_alloc_opts()`, `st_data_get()`, and `st_length_set()`.

Referenced by `portal_message_body()`.

`stringer_t* st_output (stringer_t * output, size_t len)`

Return a suitable managed string to store data of a specified length.

Note:

If a NULL output string is specified, one will be allocated for the caller.

Parameters:

output the input managed string (can be NULL).

len the size, in bytes, of the requested data buffer.

Returns:

NULL on failure or if input was not large enough or a pointer to a validated output managed string.

Definition at line 690 of file `allocation.c`.

References `log_pedantic`, `st_alloc`, `st_avail_get()`, and `st_valid_destination()`.

Referenced by `symmetric_key()`, and `symmetric_vector()`.

`stringer_t* st_realloc (stringer_t * s, size_t len)`

Reallocate a managed string to a specified size.

Note:

The caller can request a new size that is either smaller (truncated) or larger than the original string. Only these types of strings can be reallocated: nuller, block, managed, and mapped. The following logic is applied to string reallocation requests: For jointed strings, a new data buffer is allocated of the requested length, the original contents copied in, the data field of the new string is set, and the original data buffer freed. The stringer header returned to the caller resides at the same address as the original. For contiguous

strings, a new data buffer is allocated for both the requested length and the new stringer header, and the data of both parts are copied from the original string to the resulting one. For block strings, the length field of the resulting string is set to the requested length. For managed strings, the length field of the resulting string is set to the original length, but the available field is adjusted accordingly. For mapped strings, an aligned `mmap()` call is made, and both the length and available fields are set to the new length.

Parameters:

s the managed string to have its size reallocated.
len the new size, in bytes, of the resulting managed string.

Returns:

NULL on failure or a pointer to the newly reallocated managed string on success.

Definition at line 552 of file `allocation.c`.

References `block_t`, `BLOCK_T`, `CONTIGUOUS`, `data`, `JOINTED`, `log_pedantic`, `magma`, `managed_t`, `MANAGED_T`, `mapped_t`, `MAPPED_T`, `magma_t::memory`, `MEMORYBUF`, `mm_alloc()`, `mm_copy()`, `mm_free()`, `mm_sec_alloc()`, `mm_sec_free()`, `nuller_t`, `NULLER_T`, `magma_t::page_length`, `magma_t::secure`, `SECURE`, `st_info_opts()`, `st_length_get()`, `st_valid_opts()`, and `system_limit_cur()`.

Referenced by `serialize_int16()`, `serialize_int32()`, `serialize_int64()`, `serialize_ns()`, `serialize_st()`, `serialize_uint16()`, `serialize_uint32()`, `serialize_uint64()`, `smtp_data_read()`, and `st_append_opts()`.

magma/core/strings/data.c File Reference

Functions used to inspect the data of managed strings.

```
#include "magma.h"
```

Functions

- **bool_t st_empty** (**stringer_t** *s)
 - *Determine whether the specified managed string is empty or not.* **bool_t st_empty_out** (**stringer_t** *s, **uchr_t** **ptr, **size_t** *len)
 - *Determine whether the specified managed string is empty or not, and store its underlying data and data length.* **void st_data_set** (**stringer_t** *s, **void** *data)
 - *Set the underlying data of a jointed managed string.* **void** * **st_data_get** (**stringer_t** *s)
 - *Retrieve the data associated with a managed string.* **chr_t** * **st_char_get** (**stringer_t** *s)
 - *Retrieve a character pointer to a managed string's data buffer.* **uchr_t** * **st_uchar_get** (**stringer_t** *s)
 - *Retrieve an unsigned character pointer to a managed string's data buffer.* **void** **st_wipe** (**stringer_t** *s)
- Wipe all of a managed string's allocated memory and if applicable, reset its length.*
-

Detailed Description

Functions used to inspect the data of managed strings.

\$Author\$ \$Author\$ \$Revision\$

Definition in file **data.c**.

Function Documentation

chr_t * **st_char_get** (**stringer_t** * s)

Retrieve a character pointer to a managed string's data buffer.

data.c

See also:

st_data_get()

Parameters:

s the input managed string.

Returns:

NULL on failure or for an improperly constructed string; otherwise, a pointer to the string's data.

Definition at line 158 of file data.c.

References **st_data_get**().

Referenced by **adjust_message_encryption**(), **cache_add**(), **cache_append**(), **cache_config**(), **cache_delete**(), **cache_get**(), **cache_get_u64**(), **cache_increment**(), **cache_output_settings**(), **cache_set**(), **cache_set_u64**(), **cache_set_value**(), **cache_silent_add**(), **cipher_name**(), **client_read**(), **client_read_line**(), **con_read**(), **con_read_line**(), **con_write_st**(), **config_load_cmdline_settings**(), **config_load_database_settings**(), **config_load_file_settings**(), **config_output_value**(), **config_output_value_generic**(), **config_value_set**(), **contact_abuse_checks**(), **contact_abuse_increment_history**(), **contact_business_valid_email**(), **contact_detail_delete**(), **contact_detail_upsert**(), **contact_insert**(), **contact_print_form**(), **contact_update**(), **credential_address**(), **credential_username**(), **decrypt_user_messages**(), **deserialize_int16**(), **deserialize_int32**(), **deserialize_int64**(), **deserialize_ns**(), **deserialize_st**(), **deserialize_uint16**(), **deserialize_uint32**(),

deserialize_uint64(), digest_name(), double_conv(), dspam_check(), dspam_train(), encrypt_user_messages(),
 float_conv(), folder_exists(), get_header_value_noopt(), get_temp_file_handle(), hex_encode_st_debug(),
 http_body(), http_content_load_directory(), http_data_header_parse(), http_data_value_decode(),
 http_data_value_parse(), http_load_file(), http_page_get(), http_parse_context(), http_parse_header(),
 http_parse_method(), http_parse_pairs(), http_print_301(), http_response_allow_cross(),
 http_response_cookie(), http_response_header(), http_response_options(), imap_append(),
 imap_append_message(), imap_build_array(), imap_capability(), imap_check(), imap_close(),
 imap_command_log_safe(), imap_copy(), imap_create(), imap_delete(), imap_examine(), imap_expunge(),
 imap_fetch(), imap_fetch_body(), imap_fetch_body_header(), imap_fetch_body_portion(),
 imap_fetch_bodystructure(), imap_fetch_envelope(), imap_fetch_parse_partial(), imap_fetch_return_header(),
 imap_folder_compare(), imap_id(), imap_idle(), imap_init(), imap_invalid(), imap_list(), imap_login(),
 imap_logout(), imap_lsub(), imap_narrow_messages(), imap_noop(), imap_parse_address_breaker(),
 imap_parse_address_part(), imap_parse_address_put(), imap_parse_literal(), imap_parse_qstring(),
 imap_process(), imap_rename(), imap_search(), imap_search_messages_date(),
 imap_search_messages_header(), imap_select(), imap_starttls(), imap_status(), imap_store(), imap_subscribe(),
 imap_unsubscribe(), ip_presentation(), ip_reversed(), ip_standard(), lib_load(), line_pl_st(), lock_get(),
 lock_release(), magma_folder_insert(), magma_folder_rename(), mail_add_forward_headers(),
 mail_add_outbound_headers(), mail_add_required_headers(), mail_build_signature(), mail_create_directory(),
 mail_db_insert_duplicate_message(), mail_db_insert_message(), mail_discover_encoding(),
 mail_discover_insertion_point(), mail_discover_type(), mail_extract_address(), mail_extract_tag(),
 mail_get_boundary(), mail_get_chunk(), mail_header_end(), mail_header_fetch_all(),
 mail_header_fetch_cleaned(), mail_header_pop(), mail_headers(), mail_insert_chunk_base64(),
 mail_insert_chunk_text(), mail_load_header(), mail_load_message(), mail_load_message_top(),
 mail_message(), mail_message_cleanup(), mail_message_path(), mail_mime_boundary(), mail_mime_child(),
 mail_mime_content_encoding(), mail_mime_content_id(), mail_mime_count(), mail_mime_encoding(),
 mail_mime_generate_boundary(), mail_mime_header(), mail_mime_part(), mail_mime_type(),
 mail_mime_type_group(), mail_mime_type_parameters_key(), mail_mime_type_parameters_value(),
 mail_mime_type_sub(), mail_mime_update(), mail_mod_subject(), mail_modify_part(), mail_setup_basic(),
 mail_signature_add(), meta_data_check_mailbox(), meta_data_delete_tag(),
 meta_data_fetch_mailbox_aliases(), meta_data_fetch_messages(), meta_data_insert_folder(),
 meta_data_insert_tag(), meta_data_update_folder_name(), meta_data_user_build(),
 meta_data_user_save_storage_keys(), mt_get_char(), pl_char_get(), pop_pass(), pop_retr(), pop_top(),
 portal_config_collection(), portal_config_entry(), portal_contact_details(), portal_endpoint(),
 portal_endpoint_alert_acknowledge(), portal_endpoint_alert_list(), portal_endpoint_aliases(),
 portal_endpoint_attachments_add(), portal_endpoint_attachments_remove(), portal_endpoint_auth(),
 portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(),
 portal_endpoint_contacts_list(), portal_endpoint_contacts_load(), portal_endpoint_contacts_move(),
 portal_endpoint_contacts_remove(), portal_endpoint_folders_add(), portal_endpoint_folders_list(),
 portal_endpoint_folders_remove(), portal_endpoint_folders_rename(), portal_endpoint_folders_tags(),
 portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(),
 portal_endpoint_messages_load(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(),
 portal_endpoint_messages_send(), portal_endpoint_messages_tag(), portal_endpoint_messages_tags(),
 portal_message_attachments(), portal_message_body(), portal_message_header(), portal_message_tags_array(),
 portal_meta(), portal_settings_changepass(), portal_settings_identity(), portal_upload(),
 portal_validate_request(), process_kill(), register_abuse_check_blocklist(), register_abuse_checks(),
 register_abuse_increment_history(), register_business_step1(), register_business_validate_password(),
 register_business_validate_username(), register_captcha_generate(), register_data_check_username(),
 register_data_insert_user(), register_print_captcha(), register_session_cache(), register_session_get(),
 relay_config(), relay_output_settings(), relay_set_value(), serial_get(), serial_increment(), serial_reset(),
 serialize_int16(), serialize_int32(), serialize_int64(), serialize_ns(), serialize_st(), serialize_uint16(),
 serialize_uint32(), serialize_uint64(), servers_config(), servers_output_settings(), servers_set_value(),
 sess_get(), signal_shutdown(), smtp_add_bypass_entry(), smtp_bounce(), smtp_check_authorized_from(),
 smtp_check_filters(), smtp_check_greylist(), smtp_check_rbl(), smtp_check_receive_quota(),
 smtp_client_connect(), smtp_client_send_data(), smtp_client_send_helo(), smtp_client_send_mailfrom(),
 smtp_client_send_nullfrom(), smtp_client_send_rcptto(), smtp_data_finish(), smtp_data_outbound(),
 smtp_data_read(), smtp_ehlo(), smtp_fetch_authorization(), smtp_fetch_inbound(), smtp_helo(), smtp_init(),

smtp_insert_spamsig(), smtp_rcpt_to(), smtp_reply(), smtp_rollout(), smtp_update_receive_stats(), spf_check(), spool_check(), spool_cleanup(), spool_mktemp(), spool_start(), spool_stop(), sql_query_conn(), ssl_start(), st_info_opts(), st_merge_opts(), st_replace(), st_trim(), teacher_add_cookie(), teacher_print_message(), teacher_process(), time_print_gmt(), time_print_local(), tok_pop_init_st(), user_config_delete(), user_config_upsert(), virus_engine_create(), and virus_start().

void* st_data_get (stringer_t * s)

Retrieve the data associated with a managed string.

Parameters:

s the input managed string.

Returns:

NULL on failure or for an improperly constructed string; otherwise, a pointer to the string's data.

Definition at line 110 of file data.c.

References block_t, BLOCK_T, constant_t, CONSTANT_T, debug_hook(), log_pedantic, managed_t, MANAGED_T, mapped_t, MAPPED_T, MEMORYBUF, nuller_t, NULLER_T, placer_t, PLACER_T, st_info_opts(), and st_valid_opts().

Referenced by adjust_message_encryption(), alert_alloc(), alias_alloc(), base64_decode(), base64_decode_mod(), base64_encode(), base64_encode_mod(), client_read(), client_read_line(), compress_bzip(), compress_import(), compress_lzo(), compress_zlib(), con_read(), con_read_line(), contact_alloc(), contact_detail_alloc(), contact_name(), credential_alloc_auth(), credential_alloc_mail(), decompress_block_lzo(), decompress_bzip(), decompress_lzo(), decompress_zlib(), digest_hash(), dkim_check(), dkim_create(), domain_alloc(), file_load(), file_read(), hex_decode_st(), hex_encode_st(), http_body(), http_parse_origin(), imap_command_parser(), imap_fetch_bodystructure(), int16_conv_st(), int32_conv_st(), int64_conv_st(), int8_conv_st(), magma_folder_alloc(), mail_add_forward_headers(), mail_add_outbound_headers(), mail_discover_insertion_point(), mail_extract_address(), mail_get_boundary(), mail_insert_chunk_base64(), mail_load_message(), mail_mime_header(), mail_mime_part(), mail_mod_subject(), mail_modify_part(), message_alloc(), meta_data_user_build_storage_keys(), meta_folder_stats_tag_alloc(), pl_data_get(), pop_num_parse(), pop_pass_parse(), pop_top_parse(), pop_user_parse(), qp_decode(), qp_encode(), rand_choices(), rand_write(), register_data_insert_user(), sanity_check(), scramble_encrypt(), scramble_import(), sess_get(), smtp_check_greylist(), smtp_data_finish(), smtp_data_read(), sql_query_conn(), st_append_opts(), st_char_get(), st_copy_in(), st_dupe_opts(), st_empty(), st_empty_out(), st_import(), st_merge_opts(), st_nullify(), st_replace(), st_uchar_get(), st_vaprint_opts(), st_vsprint(), st_wipe(), symmetric_decrypt(), symmetric_encrypt(), symmetric_vector(), tank_store(), tok_get_count_st(), tok_get_st(), uint16_conv_st(), uint32_conv_st(), uint64_conv_st(), uint8_conv_st(), url_decode(), url_encode(), user_config_entry_alloc(), virus_check(), xml_encode(), zbase32_decode(), and zbase32_encode().

void st_data_set (stringer_t * s, void * data)

Set the underlying data of a jointed managed string.

Parameters:

s the managed string to be adjusted.

data the data buffer to be attached to the input managed string.

Note:

The underlying data of *s* will be released, unless it contains foreign data.

Returns:

This function does not return a value.

Definition at line 53 of file data.c.

References block_t, BLOCK_T, FOREIGNDATA, JOINED, log_pedantic, managed_t, MANAGED_T, MAPPED_T, MEMORYBUF, mm_free(), mm_sec_free(), mm_sec_secured(), nuller_t, NULLER_T, placer_t, PLACER_T, SECURE, st_info_opts(), and st_valid_joined().

Referenced by contact_name(), credential_alloc_mail(), smtp_data_finish(), smtp_data_read(), and st_dupe_opts().

bool_t st_empty (stringer_t * s)

Determine whether the specified managed string is empty or not.

Parameters:

s the input managed string.

Returns:

true if string is NULL or uninitialized or empty; false otherwise.

Definition at line 20 of file data.c.

References st_data_get(), and st_length_get().

Referenced by cache_free(), cache_output_settings(), cache_set_value(), cache_validate(), cipher_name(), client_read_line(), compress_import(), con_read_line(), config_load_cmdline_settings(), config_load_database_settings(), config_load_file_settings(), config_output_value(), config_output_value_generic(), config_value_set(), contact_create(), contact_edit(), contact_find_detail(), contact_find_name(), contact_folder_create(), contact_folder_rename(), contact_update(), credential_alloc_auth(), digest_hash(), digest_name(), double_conv(), float_conv(), lock_get(), lock_release(), magma_folder_find_full_name(), magma_folder_find_name(), magma_folder_name(), mail_add_outbound_headers(), mail_add_required_headers(), mail_header_fetch_all(), mail_header_fetch_cleaned(), mail_header_pop(), mail_insert_chunk_text(), mail_mime_child(), mail_mime_count(), mail_mime_part(), mail_mime_split(), message_folder_create(), meta_data_check_mailbox(), meta_data_user_build(), meta_data_user_build_storage_keys(), meta_get(), meta_remove(), meta_user_build(), meta_user_prune(), mt_is_empty(), pl_empty(), pop_pass(), relay_free(), relay_output_settings(), relay_set_value(), relay_validate(), scramble_import(), servers_free(), servers_network_start(), servers_output_settings(), servers_set_value(), servers_validate(), smtp_auth_login(), smtp_auth_plain(), st_info_opts(), st_merge_opts(), and user_config_edit().

bool_t st_empty_out (stringer_t * s, uchr_t ** ptr, size_t * len)

Determine whether the specified managed string is empty or not, and store its underlying data and data length.

Parameters:

s the input managed string.

ptr a pointer to a null-terminated string that will receive the address of the managed string's underlying data.

len a pointer to store the size of the managed string's underlying data buffer.

Returns:

true if string is NULL or uninitialized or empty; false otherwise.

Definition at line 36 of file data.c.

References st_data_get(), and st_length_get().

Referenced by `base64_decode()`, `base64_decode_mod()`, `base64_decode_opts()`, `base64_encode()`, `base64_encode_mod()`, `base64_encode_opts()`, `client_write()`, `contact_name()`, `contact_validate_detail()`, `contact_validate_name()`, `credential_address()`, `ecies_decrypt()`, `ecies_encrypt()`, `hex_count_st()`, `hex_decode_st()`, `hex_encode_st()`, `hex_valid_st()`, `http_parse_origin()`, `imap_command_log_safe()`, `imap_count_folder_levels()`, `imap_fetch_body_portion()`, `imap_valid_folder_name()`, `imap_valid_sequence()`, `lower_st()`, `mail_count_received()`, `mail_extract_address()`, `mail_mime_encode_part()`, `qp_decode()`, `qp_encode()`, `smtp_parse_auth()`, `st_cmp_ci_ends()`, `st_cmp_ci_eq()`, `st_cmp_ci_starts()`, `st_cmp_cs_ends()`, `st_cmp_cs_eq()`, `st_cmp_cs_starts()`, `st_replace()`, `st_search_chr()`, `st_search_ci()`, `st_search_cs()`, `st_swap()`, `symmetric_decrypt()`, `symmetric_encrypt()`, `upper_st()`, `url_decode()`, `url_encode()`, `url_valid_st()`, `zbase32_decode()`, and `zbase32_encode()`.

`uchr_t* st_uchar_get (stringer_t * s)`

Retrieve an unsigned character pointer to a managed string's data buffer.

See also:

`st_data_get()`

Parameters:

s the input managed string.

Returns:

NULL on failure or for an improperly constructed string; otherwise, a pointer to the string's data.

Definition at line 169 of file `data.c`.

References `st_data_get()`.

Referenced by `http_parse_origin()`.

`void st_wipe (stringer_t * s)`

Wipe all of a managed string's allocated memory and if applicable, reset its length.

Parameters:

s the managed string to be wiped.

Returns:

This function returns no value.

Definition at line 179 of file `data.c`.

References `log_pedantic`, `MEMORYBUF`, `mm_wipe()`, `st_avail_get()`, `st_data_get()`, `st_info_opts()`, `st_length_set()`, `st_valid_opts()`, and `st_valid_tracked()`.

Referenced by `ip_standard()`, and `pop_pass()`.

magma/providers/storage/data.c File Reference

The meta functions needed by the engine module.

```
#include "magma.h"
```

Functions

- `uint64_t tank_insert_object` (`int64_t transaction`, `uint64_t hnum`, `uint64_t tnum`, `uint64_t unum`, `uint64_t size`, `uint64_t flags`)
- *Insert a tank object into the mysql database.* `bool_t tank_delete_object` (`int64_t transaction`, `uint64_t hnum`, `uint64_t tnum`, `uint64_t unum`, `uint64_t onum`)

Delete a tank object from the mysql database.

Detailed Description

The meta functions needed by the engine module.

Definition in file `data.c`.

Function Documentation

`bool_t tank_delete_object` (**`int64_t transaction`**, **`uint64_t hnum`**, **`uint64_t tnum`**, **`uint64_t unum`**, **`uint64_t onum`**)

Delete a tank object from the mysql database.

Parameters:

transaction a mysql connection identifier for which a transaction has been started.

hnum the host number.

tnum the storage tank number.

unum the usernum of the user that owns the object.

onum the stored object id.

Returns:

false on failure, or true on success.

Definition at line 72 of file `data.c`.

References `log_pedantic`, `mm_wipe()`, and `stmt_exec_affected_conn()`.

Referenced by `tank_delete()`.

`uint64_t tank_insert_object` (**`int64_t transaction`**, **`uint64_t hnum`**, **`uint64_t tnum`**, **`uint64_t unum`**, **`uint64_t size`**, **`uint64_t flags`**)

Insert a tank object into the mysql database.

Parameters:

transaction a mysql connection identifier for which a transaction has been started.

hnum the host number.

tnum the storage tank number.

unum the usernum of the user that owns the object.

size the length, in bytes, of the object to be stored.

flags 0 on failure, or the objectnum field for the newly inserted object.

Definition at line 24 of file data.c.

References `mm_wipe()`, and `stmt_insert_conn()`.

Referenced by `tank_store()`.

magma/servers/http/data.c File Reference

Assorted functions for handling HTTP data.

```
#include "magma.h"
```

Functions

- void **http_data_free** (**http_data_t** ***data**)
- *Free an http data object.* **http_data_t** * **http_data_get** (**connection_t** ***con**, **HTTP_DATA** **source**, **chr_t** ***name**)
- *Get a name/value pair associated with an http connection, by name.* void **http_data_value_decode** (**stringer_t** ***string**)
- *Decode an escaped URI component into its original data.* **int_t** **http_data_value_parse** (**connection_t** ***con**, **HTTP_DATA** **source**, **placer_t** **pair**)
- *Parse a string containing a name/value pair, and place it in the specified connection's pairs holder.* **http_data_t** * **http_data_header_parse_line** (**chr_t** ***buf**, **size_t** **len**)
- *Parse a data buffer into a http header name/value pair.* **http_data_t** * **http_data_header_parse** (**connection_t** ***con**)

Parse the current line of input from an http client connection into a http header name/value pair.

Detailed Description

Assorted functions for handling HTTP data.

Definition in file **data.c**.

Function Documentation

void http_data_free (**http_data_t** * **data**)

Free an http data object.

data.c

Parameters:

data the http data object to be freed.

Returns:

This function returns no value.

Definition at line 20 of file data.c.

References `mm_free()`, `http_data_t::name`, `st_cleanup()`, and `http_data_t::value`.

Referenced by `http_data_value_parse()`, `http_parse_header()`, `http_parse_pairs()`, and `portal_upload()`.

http_data_t * **http_data_get** (**connection_t** * **con**, **HTTP_DATA** **source**, **chr_t** * **name**)

Get a name/value pair associated with an http connection, by name.

Note:

The name/value pairs searched can be supplied by a client through http request headers, or through GET or POST data.

Parameters:

con the connection object to be queried.

source the source of the client supplied value pair: HTTP_DATA_GET, HTTP_DATA_POST or HTTP_DATA_ANY.

name the name associated with the name/value pair to be retrieved.

Returns:

NULL on failure, or a pointer to the http name/value data pair requested on success.

TODO: We could just use the index find function.

Definition at line 39 of file data.c.

References `data`, `connection_t::http`, `HTTP_DATA_ANY`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `http_data_t::name`, `NULLER`, `http_data_t::source`, and `st_cmp_ci_eq()`.

Referenced by `contact_business()`, `contact_print_form()`, `contact_process()`, `http_body()`, `http_response_allow_cross()`, `multipart_get_boundary()`, `register_business_step1()`, `register_business_step2()`, `register_process()`, and `teacher_process()`.

`http_data_t* http_data_header_parse (connection_t * con)`

Parse the current line of input from an http client connection into a http header name/value pair.

Parameters:

con the connection object of the http client to be read.

Returns:

NULL on failure, or a pointer to an http header name/value pair on success.

Definition at line 231 of file data.c.

References `http_data_header_parse_line()`, `connection_t::line`, `connection_t::network`, `st_char_get()`, and `st_length_get()`.

Referenced by `http_parse_header()`.

`http_data_t* http_data_header_parse_line (chr_t * buf, size_t len)`

Parse a data buffer into a http header name/value pair.

Parameters:

con the connection object of the http client to be read.

Returns:

NULL on failure, or a pointer to an http header name/value pair on success.

Definition at line 168 of file data.c.

References `HTTP_DATA_HEADER`, `mm_alloc()`, `http_data_t::name`, `http_data_t::source`, `st_free()`, `st_import()`, and `http_data_t::value`.

Referenced by `http_data_header_parse()`, and `portal_upload()`.

void http_data_value_decode (stringer_t * *string*)

Decode an escaped URI component into its original data.

Note:

Since the un-escaped string will always be at least as small as the encoded value, the input managed string is transformed in place.

Parameters:

string a managed string containing the escaped URI data to be decoded.

Returns:

This function returns no value.

Definition at line 73 of file data.c.

References hex_decode_chr(), length, st_char_get(), st_length_get(), and st_length_set().

Referenced by http_data_value_parse().

int_t http_data_value_parse (connection_t * *con*, HTTP_DATA *source*, placer_t *pair*)

Parse a string containing a name/value pair, and place it in the specified connection's pairs holder.

Note:

Each name/value pair is assumed to be delimited with a '=' character, and each half is URI-decoded before storage.

Parameters:

con a pointer to the connection object of the http client submitting the user data to be parsed.

source an HTTP_DATA value specifying the source of the name/value pair (can be HTTP_DATA_HEADER, HTTP_DATA_GET, or HTTP_DATA_POST).

pair a placer pointing to a string containing the name/value pair to be parsed.

Returns:

0 on general or parsing failure, or 1 if the specified input buffer was successfully parsed and stored.

Definition at line 116 of file data.c.

References CONTIGUOUS, data, HEAP, connection_t::http, magma_t::http, http_data_free(), http_data_value_decode(), inx_insert(), magma_t::log, log_pedantic, M_TYPE_STRINGER, magma, MANAGED_T, mm_alloc(), http_data_t::name, pl_empty(), placer_t, http_data_t::source, multi_t::st, st_char_get(), st_dupe_opts(), st_free(), st_length_int(), tok_get_pl(), multi_t::val, and http_data_t::value.

Referenced by http_parse_pairs().

magma/core/strings/info.c File Reference

A collection of functions used to extract information from stringer options.

#include "magma.h"

Functions

- const **chr_t** * **st_info_type** (uint32_t opts)
- *Get a readable description of a managed string's data type.* const **chr_t** * **st_info_layout** (uint32_t opts)
- *Get a readable description of a managed string's data layout.* const **chr_t** * **st_info_allocator** (uint32_t opts)
- *Get a readable description of a managed string's allocator type.* **chr_t** * **st_info_opts** (uint32_t opts, **chr_t** *s, size_t len)

Get a readable description of all of a managed string's option flags. Variables

- **chr_t** * **st_option_flags** []
- **chr_t** * **st_option_types** []
- **chr_t** * **st_option_layouts** []
- **chr_t** * **st_option_allocators** []

Detailed Description

A collection of functions used to extract information from stringer options.

\$Author\$ \$Author\$ \$Revision\$

Definition in file **info.c**.

Function Documentation

const chr_t* st_info_allocator (uint32_t opts)

Get a readable description of a managed string's allocator type.

Parameters:

opts a value containing the managed string's option mask.

Returns:

a pointer to a null-terminated string containing a description of the managed string's allocator type.

Definition at line 101 of file info.c.

References HEAP, SECURE, st_option_allocators, and STACK.

Referenced by st_info_opts().

const chr_t* st_info_layout (uint32_t opts)

Get a readable description of a managed string's data layout.

Parameters:

opts a value containing the managed string's option mask.

Returns:

a pointer to a null-terminated string containing a description of the managed string's data layout.

Definition at line 80 of file info.c.

References CONTIGUOUS, JOINTED, and st_option_layouts.

Referenced by st_info_opts().

chr_t* st_info_opts (uint32_t *opts*, chr_t * *s*, size_t *len*)

Get a readable description of all of a managed string's option flags.

Parameters:

opts a value containing the managed string's option mask.

s a pointer to a null-terminated string to receive the options description.

len the length, in bytes, of the description output buffer.

Returns:

NULL on failure, or a pointer to the output buffer containing the managed string's options description on success.

LOW: Turn this into a loop.

Definition at line 127 of file info.c.

References FOREIGNDATA, NULLER, st_append, st_char_get(), st_cleanup(), st_empty(), st_info_allocator(), st_info_layout(), st_info_type(), st_length_int(), and st_option_flags.

Referenced by st_alloc_opts(), st_avail_get(), st_avail_set(), st_data_get(), st_data_set(), st_dupe_opts(), st_free(), st_length_get(), st_length_set(), st_merge_opts(), st_opt_get(), st_opt_set(), st_realloc(), st_vaprint_opts(), st_vsprint(), and st_wipe().

const chr_t* st_info_type (uint32_t *opts*)

Get a readable description of a managed string's data type.

Parameters:

opts a value containing the managed string's option mask.

Returns:

a pointer to a null-terminated string containing a description of the managed string's data type.

Definition at line 47 of file info.c.

References BLOCK_T, CONSTANT_T, MANAGED_T, MAPPED_T, NULLER_T, PLACER_T, and st_option_types.

Referenced by st_info_opts().

Variable Documentation

chr_t* st_option_allocators[]

```
Initial value: {  
    "UNKNOWN",  
    "STACK",  
    "HEAD",  
    "SECURE"  
}
```

Definition at line 35 of file info.c.

Referenced by st_info_allocator().

chr_t* st_option_flags[]

```
Initial value: {  
    "FOREIGNDATA"  
}
```

Definition at line 15 of file info.c.

Referenced by st_info_opts().

chr_t* st_option_layouts[]

```
Initial value: {  
    "UNKNOWN",  
    "CONTIGUOUS",  
    "JOINTED"  
}
```

Definition at line 29 of file info.c.

Referenced by st_info_layout().

chr_t* st_option_types[]

```
Initial value: {  
    "UNKNOWN",  
    "CONSTANT",  
    "PLACER",  
    "NULLER",  
    "BLOCK",  
    "MANAGED",  
    "MAPPED"  
}
```

Definition at line 19 of file info.c.

Referenced by st_info_type().

magma/core/strings/length.c File Reference

Fncions used to find stringer lengths.

```
#include "magma.h"
```

Functions

- `size_t st_length_get (stringer_t *s)`
 - *Return the length of the data in a managed string. `int_t st_length_int (stringer_t *s)`*
 - `size_t st_length_set (stringer_t *s, size_t len)`
 - *Set the data length for a managed string that supports data length tracking. `size_t st_avail_get (stringer_t *s)`*
 - *Return the total data buffer size allocated for a managed string. `size_t st_avail_set (stringer_t *s, size_t avail)`*
-
- Set the total data buffer size allocated for a managed string.*

Detailed Description

Fncions used to find stringer lengths.

Definition in file **length.c**.

Function Documentation

size_t st_avail_get (stringer_t * s)

Return the total data buffer size allocated for a managed string.

Parameters:

s the input managed string.

Returns:

0 on failure, or the total buffer size in bytes on success.

Definition at line 151 of file length.c.

References `log_pedantic`, `managed_t`, `MANAGED_T`, `mapped_t`, `MAPPED_T`, `MEMORYBUF`, `st_info_opts()`, `st_length_get()`, and `st_valid_opts()`.

Referenced by `base64_decode()`, `base64_decode_mod()`, `base64_encode()`, `base64_encode_mod()`, `client_read()`, `client_read_line()`, `con_read()`, `con_read_line()`, `digest_hash()`, `file_load()`, `file_read()`, `hex_decode_st()`, `hex_encode_st()`, `host_platform()`, `host_version()`, `imap_parse_address_put()`, `ip_presentation()`, `ip_reversed()`, `ip_standard()`, `ip_subnet()`, `mail_add_required_headers()`, `rand_write()`, `sanity_check()`, `serialize_int16()`, `serialize_int32()`, `serialize_int64()`, `serialize_ns()`, `serialize_st()`, `serialize_uint16()`, `serialize_uint32()`, `serialize_uint64()`, `st_append_opts()`, `st_copy_in()`, `st_dupe_opts()`, `st_output()`, `st_trim()`, `st_vsprint()`, `st_wipe()`, `time_print_gmt()`, and `time_print_local()`.

size_t st_avail_set (stringer_t * s, size_t avail)

Set the total data buffer size allocated for a managed string.

Parameters:

s the input managed string.
avail the new buffer size for the managed string.

Returns:

0 on failure, or the managed string's new buffer size in bytes on success.

Definition at line 188 of file length.c.

References log_pedantic, managed_t, MANAGED_T, mapped_t, MAPPED_T, MEMORYBUF, st_info_opts(), and st_valid_avail().

size_t st_length_get (stringer_t * s)

Return the length of the data in a managed string.

Parameters:

s the input managed string.

Returns:

0 on failure, or the length of the managed string's data in bytes.

Definition at line 20 of file length.c.

References block_t, BLOCK_T, constant_t, CONSTANT_T, data, log_pedantic, managed_t, MANAGED_T, mapped_t, MAPPED_T, MEMORYBUF, nuller_t, NULLER_T, placer_t, PLACER_T, st_info_opts(), and st_valid_opts().

Referenced by adjust_message_encryption(), alert_alloc(), alias_alloc(), base64_decode(), base64_decode_mod(), base64_encode(), base64_encode_mod(), cache_add(), cache_append(), cache_config(), cache_delete(), cache_get(), cache_get_u64(), cache_increment(), cache_set(), cache_set_u64(), cache_set_value(), cache_silent_add(), client_read(), client_read_line(), compress_bzip(), compress_import(), compress_lzo(), compress_zlib(), con_read(), con_read_line(), con_write_st(), config_value_set(), contact_alloc(), contact_business_valid_email(), contact_detail_alloc(), contact_detail_delete(), contact_detail_upsert(), contact_folder_create(), contact_folder_rename(), contact_insert(), contact_print_form(), contact_print_message(), contact_update(), credential_address(), credential_alloc_auth(), credential_alloc_mail(), credential_username(), decompress_block_lzo(), deserialize_int16(), deserialize_int32(), deserialize_int64(), deserialize_ns(), deserialize_st(), deserialize_uint16(), deserialize_uint32(), deserialize_uint64(), digest_hash(), dkim_check(), dkim_create(), domain_alloc(), double_conv(), dspam_train(), float_conv(), get_header_value_noopt(), hex_decode_st(), hex_encode_st(), hex_encode_st_debug(), http_body(), http_data_header_parse(), http_data_value_decode(), http_load_file(), http_page_get(), http_parse_context(), http_parse_header(), http_parse_pairs(), http_response(), imap_append_message(), imap_build_array(), imap_command_parser(), imap_copy(), imap_fetch_body(), imap_fetch_body_header(), imap_fetch_bodystructure(), imap_fetch_envelope(), imap_fetch_message(), imap_fetch_parse_partial(), imap_folder_compare(), imap_id(), imap_init(), imap_narrow_folders(), imap_parse_address_breaker(), imap_parse_address_part(), imap_parse_address_put(), imap_process(), imap_search(), imap_search_messages_date(), imap_search_messages_header(), imap_starttls(), imap_status(), imap_valid_folder_name(), int16_conv_st(), int32_conv_st(), int64_conv_st(), int8_conv_st(), ip_subnet(), line_pl_st(), magma_folder_alloc(), magma_folder_insert(), magma_folder_rename(), mail_add_forward_headers(), mail_add_outbound_headers(), mail_add_required_headers(), mail_build_signature(), mail_db_insert_duplicate_message(), mail_db_insert_message(), mail_discover_encoding(), mail_discover_insertion_point(), mail_discover_type(), mail_get_boundary(), mail_get_chunk(), mail_header_end(), mail_header_fetch_all(), mail_header_fetch_cleaned(), mail_header_pop(), mail_headers(), mail_insert_chunk_base64(), mail_insert_chunk_text(), mail_load_header(), mail_load_message_top(), mail_message(), mail_message_cleanup(), mail_mime_boundary(), mail_mime_child(), mail_mime_content_encoding(), mail_mime_content_id(), mail_mime_count(), mail_mime_encoding(), mail_mime_header(), mail_mime_part(), mail_mime_type(), mail_mime_type_group(), mail_mime_type_parameters_key(), mail_mime_type_parameters_value(),

mail_mime_type_sub(), mail_mime_update(), mail_mod_subject(), mail_modify_part(), mail_signature_add(), mail_store_message(), message_alloc(), meta_data_check_mailbox(), meta_data_delete_tag(), meta_data_insert_folder(), meta_data_insert_tag(), meta_data_update_folder_name(), meta_data_user_build(), meta_data_user_save_storage_keys(), meta_folder_stats_tag_alloc(), mt_get_length(), nvp_parse(), pl_length_get(), pop_num_parse(), pop_pass_parse(), pop_retr(), pop_top(), pop_top_parse(), pop_user_parse(), portal_endpoint_alert_acknowledge(), portal_endpoint_attachments_add(), portal_endpoint_attachments_progress(), portal_endpoint_attachments_remove(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(), portal_endpoint_contacts_list(), portal_endpoint_contacts_load(), portal_endpoint_contacts_move(), portal_endpoint_contacts_remove(), portal_endpoint_folders_add(), portal_endpoint_folders_list(), portal_endpoint_folders_remove(), portal_endpoint_folders_rename(), portal_endpoint_folders_tags(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(), portal_endpoint_messages_load(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(), portal_endpoint_messages_send(), portal_endpoint_messages_tag(), portal_endpoint_search(), portal_message_attachments(), portal_print_login(), portal_settings_changepass(), portal_upload(), portal_validate_request(), qp_decode(), rand_write(), register_abuse_check_blocklist(), register_business_validate_password(), register_business_validate_username(), register_captcha_generate(), register_data_check_username(), register_data_insert_user(), register_print_captcha(), register_print_message(), register_print_step1(), register_print_step2(), register_print_step3(), relay_config(), relay_set_value(), scramble_decrypt(), scramble_encrypt(), scramble_import(), serialize_int16(), serialize_int32(), serialize_int64(), serialize_ns(), serialize_st(), serialize_uint16(), serialize_uint32(), serialize_uint64(), servers_config(), servers_set_value(), smtp_accept_message(), smtp_auth_plain(), smtp_check_authorized_from(), smtp_check_filters(), smtp_check_greylist(), smtp_check_receive_quota(), smtp_client_connect(), smtp_client_send_data(), smtp_client_send_helo(), smtp_client_send_mailfrom(), smtp_client_send_rcptto(), smtp_data_outbound(), smtp_fetch_authorization(), smtp_fetch_inbound(), smtp_insert_spamsig(), smtp_update_receive_stats(), sql_query_conn(), st_append_opts(), st_avail_get(), st_dupe_opts(), st_empty(), st_empty_out(), st_length_int(), st_merge_opts(), st_realloc(), st_trim(), statistics_process(), symmetric_key(), tank_store(), teacher_print_form(), teacher_print_message(), teacher_process(), tok_get_count_st(), tok_get_st(), tok_pop_init_st(), uint16_conv_st(), uint32_conv_st(), uint64_conv_st(), uint8_conv_st(), url_decode(), user_config_delete(), user_config_entry_alloc(), user_config_upsert(), and virus_check().

int_t st_length_int (stringer_t * s)

Return the length of a managed string as an integer.

Parameters:

s the managed string as type stringer_t *.

Returns:

the string length as an integer, capped at INT_MAX.

Definition at line 75 of file length.c.

References log_pedantic, st_length_get(), and st_valid_opts().

Referenced by cache_add(), cache_append(), cache_config(), cache_delete(), cache_get(), cache_get_u64(), cache_increment(), cache_output_settings(), cache_set(), cache_set_u64(), cipher_name(), config_load_cmdline_settings(), config_load_database_settings(), config_load_file_settings(), config_output_value(), config_output_value_generic(), contact_abuse_checks(), contact_abuse_increment_history(), contact_detail_delete(), contact_detail_upsert(), digest_name(), double_conv(), float_conv(), http_body(), http_data_value_parse(), http_load_file(), http_parse_header(), http_parse_method(), http_print_301(), http_response_allow_cross(), http_response_cookie(), http_response_header(), http_response_options(), imap_append(), imap_append_message(), imap_capability(), imap_check(), imap_close(), imap_command_log_safe(), imap_copy(), imap_create(), imap_delete(), imap_examine(), imap_expunge(), imap_fetch(), imap_id(), imap_idle(), imap_init(), imap_invalid(), imap_list(), imap_login(), imap_logout(), imap_lsub(), imap_narrow_messages(), imap_noop(), imap_process(), imap_rename(), imap_search(), imap_select(), imap_status(), imap_store(), imap_subscribe(), imap_unsubscribe(), lock_get(), lock_release(), mail_create_directory(), mail_message_path(),

mail_signature_add(), mail_store_message(), meta_data_fetch_mailbox_aliases(), meta_data_user_build(), pl_length_int(), pop_pass(), portal_config_entry(), process_kill(), register_business_step1(), register_session_cache(), register_session_get(), relay_config(), relay_output_settings(), serial_get(), serial_increment(), serial_reset(), servers_config(), servers_output_settings(), servers_set_value(), smtp_bounce(), smtp_check_filters(), smtp_check_greylist(), smtp_check_rbl(), smtp_client_connect(), smtp_client_send_data(), smtp_client_send_mailfrom(), smtp_client_send_nullfrom(), smtp_client_send_rcptto(), smtp_ehlo(), smtp_fetch_authorization(), smtp_fetch_inbound(), smtp_helo(), smtp_init(), smtp_rcpt_to(), smtp_reply(), spf_check(), spool_check(), spool_cleanup(), spool_mktemp(), spool_start(), spool_stop(), sql_query_conn(), st_info_opts(), user_config_delete(), user_config_upsert(), and virus_start().

size_t st_length_set (stringer_t * s, size_t len)

Set the data length for a managed string that supports data length tracking.

Parameters:

s the input managed string.

len the new value of the managed string's data length.

Returns:

0 on error, or the new data length on success.

Definition at line 104 of file length.c.

References log_pedantic, managed_t, MANAGED_T, mapped_t, MAPPED_T, MEMORYBUF, placer_t, PLACER_T, st_info_opts(), and st_valid_tracked().

Referenced by base64_decode(), base64_decode_mod(), base64_encode(), base64_encode_mod(), client_read(), client_read_line(), con_read(), con_read_line(), contact_name(), credential_address(), credential_alloc_mail(), credential_username(), decompress_block_lzo(), decompress_bzip(), decompress_lzo(), decompress_zlib(), deserialize_st(), digest_hash(), file_load(), file_read(), hex_decode_st(), hex_encode_st(), hex_encode_st_debug(), http_data_value_decode(), http_load_file(), http_parse_header(), imap_build_array(), imap_parse_address_part(), imap_parse_address_put(), imap_parse_literal(), imap_parse_qstring(), imap_search(), imap_starttls(), imap_valid_folder_name(), ip_presentation(), ip_reversed(), ip_standard(), mail_add_required_headers(), mail_extract_address(), mail_extract_tag(), mail_header_fetch_cleaned(), mail_load_header(), mail_load_message(), mail_load_message_top(), mail_message_cleanup(), mail_mime_generate_boundary(), pop_starttls(), qp_decode(), rand_choices(), rand_write(), serialize_int16(), serialize_int32(), serialize_int64(), serialize_ns(), serialize_st(), serialize_uint16(), serialize_uint32(), serialize_uint64(), smtp_data_finish(), smtp_data_read(), smtp_starttls(), st_append_opts(), st_copy_in(), st_dupe_opts(), st_import(), st_merge_opts(), st_nullify(), st_replace(), st_trim(), st_vaprint_opts(), st_vsprint(), st_wipe(), symmetric_decrypt(), symmetric_encrypt(), symmetric_key(), symmetric_vector(), time_print_gmt(), time_print_local(), url_decode(), url_encode(), zbase32_decode(), and zbase32_encode().

magma/core/strings/multi.c File Reference

Functions for handling the multi-type structure.

```
#include "magma.h"
```

Functions

- **multi_t mt_get_null** (void)
- *Return an empty multi-type object.* **bool_t mt_is_empty** (**multi_t** multi)
- *Determine whether a multi-type object is empty.* **bool_t mt_is_number** (**multi_t** multi)
- *Determine whether a multi-type object is a number.* **uint64_t mt_get_number** (**multi_t** multi)
- *Get the value of a numerical multi-type object.* **char * mt_get_char** (**multi_t** multi)
- *Get a character pointer to the value of a multi-type object.* **size_t mt_get_length** (**multi_t** multi)
- *Get the length of the data associated with a multi-type object.* **M_TYPE mt_get_type** (**multi_t** multi)
- *Get the data type associated with a multi-type object.* **multi_t mt_set_type** (**multi_t** multi, **M_TYPE** target)
- *Set the data type of a multi-type data object.* **void mt_free** (**multi_t** multi)
- *Free the data underlying a multi-type data object (for managed and null-terminated strings).* **multi_t mt_dupe** (**multi_t** multi)
- *Duplicate a multi-type object.* **bool_t ident_mt_mt** (**multi_t** one, **multi_t** two)
- *Check to see if the values of two multi-type objects are identical.* **int32_t cmp_mt_mt** (**multi_t** one, **multi_t** two)

Compare the values of two multi-type objects of the same data type.

Detailed Description

Functions for handling the multi-type structure.

Definition in file **multi.c**.

Function Documentation

int32_t cmp_mt_mt (**multi_t** one, **multi_t** two)

Compare the values of two multi-type objects of the same data type.

multi.c

Note:

The types of the two input objects must match, or be a comparison between a managed string and null-terminated string.

Parameters:

one the first multi-type object to be compared.

two the second multi-type object to be compared.

Returns:

Definition at line 686 of file multi.c.

References **multi_t::binary**, **multi_t::dbl**, **multi_t::fl**, **multi_t::i16**, **multi_t::i32**, **multi_t::i64**, **multi_t::i8**, **log_options**, **log_pedantic**, **M_LOG_PEDANTIC**, **M_LOG_STACK_TRACE**, **M_TYPE_BOOLEAN**, **M_TYPE_DOUBLE**, **M_TYPE_FLOAT**, **M_TYPE_INT16**, **M_TYPE_INT32**, **M_TYPE_INT64**,

M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, mt_get_type(), multi_t::ns, NULLER, multi_t::st, st_cmp_cs_eq(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by tree_cmp().

bool_t ident_mt_mt (multi_t *one*, multi_t *two*)

Check to see if the values of two multi-type objects are identical.

Note:

The types of the two input objects must match, or be a comparison between a managed string and null-terminated string.

Parameters:

one the first multi-type object to be compared.

two the second multi-type object to be compared.

Returns:

true if the two objects have identical values; false if they don't, or on failure.

Definition at line 568 of file multi.c.

References multi_t::binary, multi_t::dbl, multi_t::fl, multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, log_options, log_pedantic, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, mt_get_type(), multi_t::ns, ns_length_get(), PLACER, multi_t::st, st_cmp_cs_eq(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by hashed_bucket_find_key(), hashed_delete(), linked_delete(), and linked_find().

multi_t mt_dupe (multi_t *multi*)

Duplicate a multi-type object.

Parameters:

multi the multi-type object to be examined.

Returns:

a copy of the input object (a deep copy for managed strings and null-terminated strings).

Definition at line 485 of file multi.c.

References multi_t::binary, CONTIGUOUS, multi_t::dbl, multi_t::fl, HEAP, multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, MANAGED_T, mt_get_null(), multi_t::ns, ns_dupe(), multi_t::st, st_dupe_opts(), type(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by hashed_bucket_alloc(), linked_record_alloc(), and tree_insert().

void mt_free (multi_t *multi*)

Free the data underlying a multi-type data object (for managed and null-terminated strings).

Parameters:

multi the multi-type object to be freed.

Returns:

This function returns no value.

Definition at line 463 of file multi.c.

References M_TYPE_NULLER, M_TYPE_STRINGER, multi_t::ns, ns_free(), multi_t::st, st_free(), multi_t::type, and multi_t::val.

Referenced by hashed_delete(), hashed_free(), hashed_truncate(), linked_record_free(), tree_delete(), tree_insert(), and tree_truncate().

char* mt_get_char (multi_t *multi*)

Get a character pointer to the value of a multi-type object.

Parameters:

multi the multi-type object to be examined.

a a pointer to the value of the input object, or NULL on failure.

Definition at line 176 of file multi.c.

References multi_t::binary, multi_t::bl, multi_t::dbl, multi_t::fl, multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, log_options, M_LOG_INFO, M_LOG_STACK_TRACE, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, multi_t::ns, multi_t::st, st_char_get(), type(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by hashed_bucket().

size_t mt_get_length (multi_t *multi*)

Get the length of the data associated with a multi-type object.

Parameters:

multi the multi-type object to be examined.

Returns:

the length in bytes of the data associated with the input object, or 0 on failure.

Definition at line 250 of file multi.c.

References log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, multi_t::ns, ns_length_get(), multi_t::st, st_length_get(), type(), multi_t::type, and multi_t::val.

Referenced by hashed_bucket().

multi_t mt_get_null (void)

Return an empty multi-type object.

Returns:

an empty multi-type object.

Definition at line 22 of file multi.c.

References EMPTY, and multi_t::type.

Referenced by hashed_cursor_key_active(), hashed_cursor_key_next(), inx_cursor_key_active(), inx_cursor_key_next(), linked_cursor_key_active(), linked_cursor_key_next(), linked_record_get_key(), mt_dupe(), tree_cursor_key_next(), and tree_cursor_next().

uint64_t mt_get_number (multi_t *multi*)

Get the value of a numerical multi-type object.

Parameters:

multi the multi-type object to be examined.

Returns:

the numerical value of the input object, or 0 on failure.

Definition at line 121 of file multi.c.

References multi_t::dbl, EMPTY, multi_t::fl, multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, type(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by hashed_bucket().

M_TYPE mt_get_type (multi_t *multi*)

Get the data type associated with a multi-type object.

Parameters:

multi the multi-type object to be examined.

Returns:

the data type of the input object, or EMPTY on failure.

Definition at line 319 of file multi.c.

References EMPTY, log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_PLACER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, type(), and multi_t::type.

Referenced by cmp_mt_mt(), and ident_mt_mt().

bool_t mt_is_empty (multi_t *multi*)

Determine whether a multi-type object is empty.

Note:

All numeric (including boolean and floating point) types will always return false.

Parameters:

multi the multi-type object to be examined.

Returns:

true if the object is empty (by type or value), or false otherwise.

Definition at line 36 of file multi.c.

References multi_t::bl, EMPTY, log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, multi_t::ns, ns_empty(), multi_t::st, st_empty(), type(), multi_t::type, and multi_t::val.

Referenced by config_load_cmdline_settings(), config_load_file_settings(), and tree_insert().

bool_t mt_is_number (multi_t *multi*)

Determine whether a multi-type object is a number.

Parameters:

multi the multi-type object to be examined. return true if the input object is any integer, boolean, or floating point; false otherwise.

Definition at line 85 of file multi.c.

References EMPTY, log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, type(), and multi_t::type.

Referenced by hashed_bucket().

multi_t mt_set_type (multi_t *multi*, M_TYPE *target*)

Set the data type of a multi-type data object.

Parameters:

multi the multi-type object to be adjusted.

target the value of the new data type for the input object.

Returns:

a copy of the modified multi-type data object.

Definition at line 392 of file multi.c.

References EMPTY, log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_PLACER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, type(), and multi_t::type.

Referenced by nvp_parse().

magma/core/strings/nuller.c File Reference

Functions for handling null terminated strings.

```
#include "magma.h"
```

Functions

- `size_t ns_length_get (const chr_t *s)`
 - *Return the length of a null-terminated string.* `bool_t ns_empty (chr_t *s)`
 - *Return whether a null-terminated string is empty or not.* `bool_t ns_empty_out (chr_t *s, chr_t **ptr, size_t *len)`
 - *Return whether a null-terminated string is empty or not, while storing its address and length.* `int ns_length_int (chr_t *s)`
 - *Return the length of a null-terminated string as an int, capped at INT_MAX.* `chr_t * ns_alloc (size_t len)`
 - *Allocate (and wipe) a buffer for a null-terminated string.* `void ns_free (chr_t *s)`
 - *Free a null-terminated string.* `void ns_cleanup (chr_t *s)`
 - *A checked null-terminated string free front-end function.* `chr_t * ns_dupe (chr_t *s)`
 - *Duplicate a null-terminated string.* `chr_t * ns_import (void *block, size_t len)`
 - *Get a block of memory as a null-terminated string.* `chr_t * ns_append (chr_t *s, chr_t *append)`
 - *Append one string to another and return the result.* `void ns_wipe (chr_t *s, size_t len)`
- Zero out a null-terminated string.*
-

Detailed Description

Functions for handling null terminated strings.

Definition in file **nuller.c**.

Function Documentation

chr_t* ns_alloc (size_t len)

Allocate (and wipe) a buffer for a null-terminated string.

nuller.c

Parameters:

len the length of the buffer to be allocated.

Returns:

NULL on failure or if len was 0; a pointer to the newly allocated memory otherwise.

Definition at line 71 of file nuller.c.

References `log_pedantic`, `mm_alloc()`, and `ns_wipe()`.

Referenced by `deserialize_ns()`, `mail_message_path()`, and `ns_append()`.

chr_t* ns_append (chr_t * s, chr_t * append)

Append one string to another and return the result.

Parameters:

s a pointer to a leading null-terminated string to to which the other string will be appended.
append a pointer to a null-terminated string to be appended to the leading string.

Returns:

NULL if a memory allocation failure occurred, or a pointer to the resulting string on success.
Definition at line 173 of file nuller.c.
References log_pedantic, mm_copy(), ns_alloc(), ns_dupe(), ns_free(), and ns_length_get().
Referenced by portal_endpoint_error(), and portal_endpoint_response().

void ns_cleanup (chr_t * s)

A checked null-terminated string free front-end function.

See also:

ns_free()

Parameters:

s the null-terminated string to be freed.

Returns:

This function returns no value.
Definition at line 115 of file nuller.c.
References ns_free().

chr_t* ns_dupe (chr_t * s)

Duplicate a null-terminated string.

Parameters:

s the null-terminated string to be duplicated.

Returns:

NULL on failure, or a pointer to a copy of the input string.
Definition at line 129 of file nuller.c.
References length, log_info, log_pedantic, mm_alloc(), mm_copy(), and ns_length_get().
Referenced by cache_set_value(), config_value_set(), mt_dupe(), ns_append(), relay_set_value(), servers_set_value(), and xml_get_xpath_ns().

bool_t ns_empty (chr_t * s)

Return whether a null-terminated string is empty or not.

Parameters:

s the input as a null-terminated string.

Returns:

true if string is NULL or zero length; false otherwise.
Definition at line 32 of file nuller.c.

References ns_length_get().

Referenced by cache_free(), cache_output_settings(), cache_set_value(), cache_validate(), config_output_value(), config_output_value_generic(), config_value_set(), mt_is_empty(), relay_free(), relay_output_settings(), relay_set_value(), relay_validate(), servers_free(), servers_output_settings(), servers_set_value(), servers_validate(), sql_start(), and st_merge_opts().

bool_t ns_empty_out (chr_t * s, chr_t ** ptr, size_t * len)

Return whether a null-terminated string is empty or not, while storing its address and length.

Parameters:

s the input as a null-terminated string.

ptr a pointer address to receive a copy of the string's location.

len a pointer to a variable to receive the length of the string, in bytes.

Returns:

true if string is NULL or zero length; false otherwise.

Definition at line 44 of file nuller.c.

References ns_length_get().

void ns_free (chr_t * s)

Free a null-terminated string.

Parameters:

s the string to be freed.

Returns:

This function returns no value.

Definition at line 98 of file nuller.c.

References log_pedantic, and mm_free().

Referenced by adjust_message_encryption(), cache_free(), cache_set_value(), config_free(), config_value_set(), ip_str_subnet(), mail_copy_message(), mail_load_header(), mail_load_message(), mail_message_path(), mail_remove_message(), mail_store_message(), mail_store_message_data(), mt_free(), ns_append(), ns_cleanup(), portal_endpoint_error(), portal_endpoint_response(), relay_free(), relay_set_value(), servers_free(), and servers_set_value().

chr_t* ns_import (void * block, size_t len)

Get a block of memory as a null-terminated string.

Parameters:

block a pointer to the buffer containing the data to be copied.

len the length, in bytes, of the data buffer to be copied.

Returns:

NULL on failure, or a pointer to the newly allocated null-terminated string on success.

Definition at line 154 of file nuller.c.

References `log_pedantic`, `mm_alloc()`, and `mm_copy()`.

Referenced by `adjust_message_encryption()`, `cache_set_value()`, `config_value_set()`, `ip_str_subnet()`, `relay_set_value()`, and `servers_set_value()`.

size_t ns_length_get (const chr_t * s)

Return the length of a null-terminated string.

Parameters:

s the input as a null-terminated string.

Returns:

the length in bytes of the string.

Definition at line 21 of file `nnuller.c`.

Referenced by `con_reverse_lookup()`, `con_write_ns()`, `config_fetch_host_number()`, `config_fetch_settings()`, `ecies_key_private_hex()`, `ecies_key_public_hex()`, `get_temp_file_handle()`, `host_platform()`, `host_version()`, `http_load_file()`, `ident_mt_mt()`, `imap_build_array()`, `imap_fetch_bodystructure()`, `imap_fetch_message()`, `imap_search_messages_date()`, `int16_conv_ns()`, `int32_conv_ns()`, `int64_conv_ns()`, `int8_conv_ns()`, `ip_presentation()`, `ip_str_addr()`, `ip_str_subnet()`, `lib_load_openssl()`, `line_pl_ns()`, `log_internal()`, `log_rotate()`, `log_start()`, `mail_mime_get_media_type()`, `mail_path_finder()`, `meta_folders_name()`, `mt_get_length()`, `ns_append()`, `ns_dupe()`, `ns_empty()`, `ns_empty_out()`, `ns_length_int()`, `portal_endpoint()`, `portal_endpoint_attachments_add()`, `portal_endpoint_error()`, `portal_endpoint_folders_add()`, `portal_endpoint_folders_rename()`, `portal_endpoint_response()`, `portal_parse_json_str_array()`, `rand_choices()`, `register_data_insert_user()`, `register_print_step1()`, `register_print_step2()`, `servers_network_start()`, `spool_path()`, `st_merge_opts()`, `stmt_rebuild()`, `stmt_start()`, `tank_start()`, `teacher_add_cookie()`, `uint16_conv_ns()`, `uint32_conv_ns()`, `uint64_conv_ns()`, `uint8_conv_ns()`, `virus_check()`, `xml_get_xpath_int16()`, `xml_get_xpath_int32()`, `xml_get_xpath_int64()`, `xml_get_xpath_int8()`, `xml_get_xpath_st()`, `xml_get_xpath_uint16()`, `xml_get_xpath_uint32()`, `xml_get_xpath_uint64()`, and `xml_get_xpath_uint8()`.

int ns_length_int (chr_t * s)

Return the length of a null-terminated string as an int, capped at `INT_MAX`.

Parameters:

s the null-terminated string to be evaluated.

Returns:

the length of the string.

Definition at line 54 of file `nnuller.c`.

References `log_pedantic`, and `ns_length_get()`.

Referenced by `lib_load_bzip()`.

void ns_wipe (chr_t * s, size_t len)

Zero out a null-terminated string.

Parameters:

s the string to be wiped.

len the number of bytes to be wiped at the start of the string.

Returns:

This function returns no parameters.

Definition at line 209 of file nuller.c.

References `log_pedantic`, and `mm_set()`.

Referenced by `ns_alloc()`.

magma/core/strings/opts.c File Reference

Functions for handling managed string options.

#include "magma.h"

Functions

- **int_t st_opt_set** (stringer_t *s, uint32_t opt, bool_t enabled)
- *Enable or disable a set of option(s) for a managed string, with validity testing.* **bool_t st_opt_get** (stringer_t *s, uint32_t opt)

Check to see if the managed string has the specified option enabled.

Detailed Description

Functions for handling managed string options.

Definition in file **opts.c**.

Function Documentation

bool_t st_opt_get (stringer_t * s, uint32_t opt)

Check to see if the managed string has the specified option enabled.

opts.c

Parameters:

s the managed string to be checked. opt a bitmask of the managed string options to be tested.

Returns:

-1 on error, 0 if opt is not set, and 1 if opt is set.

Definition at line 53 of file opts.c.

References log_pedantic, MEMORYBUF, st_info_opts(), and st_valid_opts().

Referenced by contact_free(), and contact_name().

int_t st_opt_set (stringer_t * s, uint32_t opt, bool_t enabled)

Enable or disable a set of option(s) for a managed string, with validity testing.

Parameters:

s the input managed string.

opt the bitmask of option(s) to be enabled or disabled for the managed string.

enabled if true, set the option(s); if false, disable them.

Returns:

-1 on error, 0 if successfully disabled, or 1 if successfully enabled.

Definition at line 22 of file opts.c.

References log_pedantic, MEMORYBUF, st_info_opts(), and st_valid_opts().

Referenced by contact_name().

magma/core/strings/print.c File Reference

Functions for printing formatted data to managed strings.

```
#include "magma.h"
```

Functions

- `size_t st_vsprintf (stringer_t *s, chr_t *format, va_list args)`
- *Print to a managed string and return the number of bytes written.* `size_t st_sprint (stringer_t *s, chr_t *format,...)`
- *Print to a managed string and return the number of bytes written.* `stringer_t * st_quick (stringer_t *s, chr_t *format,...)`
- *Print to a managed string and return a pointer to the result.* `stringer_t * st_vaprint_opts (uint32_t opts, chr_t *format, va_list args)`
- *Return a managed string containing sprintf()-style formatted data.* `stringer_t * st_aprint (chr_t *format,...)`
- *Return a managed string containing sprintf()-style formatted data.* `stringer_t * st_aprint_opts (uint32_t opts, chr_t *format,...)`

Return a managed string containing sprintf()-style formatted data, with custom allocation options.

Detailed Description

Functions for printing formatted data to managed strings.

Definition in file **print.c**.

Function Documentation

stringer_t* st_aprint (chr_t * *format*, ...)

Return a managed string containing sprintf()-style formatted data.

See also:

`st_vaprint_opts()`

Parameters:

format a format string for the output string data.

... a variable argument list of parameters to be formatted as output.

Returns:

NULL on failure or a managed string containing the final formatted data on success.

Definition at line 151 of file print.c.

References CONTIGUOUS, HEAP, MANAGED_T, and st_vaprint_opts().

Referenced by portal_endpoint_attachments_progress(), portal_endpoint_search(), portal_message_attachments(), register_captcha_random_font(), serial_get(), serial_increment(), serial_reset(), and spool_mktemp().

stringer_t* st_aprint_opts (uint32_t *opts*, chr_t * *format*, ...)

Return a managed string containing `sprintf()`-style formatted data, with custom allocation options.

Parameters:

opts the option value of the newly allocated managed string that will contain the result.
format a format string for the output string data.
... a variable argument list of parameters to be formatted as output.

Returns:

NULL on failure or a managed string of the specified allocation options containing the final formatted data on success.

Definition at line 170 of file `print.c`.

References `st_vaprint_opts()`.

Referenced by `http_response_cookie()`, `imap_fetch_message()`, and `imap_search()`.

`stringer_t* st_quick (stringer_t * s, chr_t * format, ...)`

Print to a managed string and return a pointer to the result.

See also:

`st_print()`

Parameters:

s a pointer to the managed string that will receive the output of the print operation.
format a format string specifying the arguments of the print operation.
... a variable argument list containing the parameters to the print format string.

Returns:

a pointer to the managed string that received the printed output.

Definition at line 87 of file `print.c`.

References `st_vsprint()`.

Referenced by `http_response_allow_cross()`, and `http_response_cookie()`.

`size_t st_sprint (stringer_t * s, chr_t * format, ...)`

Print to a managed string and return the number of bytes written.

Parameters:

s a pointer to the managed string that will receive the output of the print operation.
format a format string specifying the arguments of the print operation.
... a variable argument list containing the parameters to the print format string.

Returns:

-1 on failure, or the number of characters printed to the string, excluding the terminating null byte.

Definition at line 67 of file `print.c`.

References `st_vsprint()`.

Referenced by `contact_abuse_checks()`, `contact_abuse_increment_history()`, `host_platform()`, `host_version()`, `imap_id()`, `imap_search()`, `ip_reversed()`, `ip_standard()`, `ip_subnet()`, `lock_get()`, `lock_release()`, `mail_build_signature()`, `smtp_bounce()`, `smtp_check_greylist()`, `smtp_reply()`, `user_lock()`, and `user_unlock()`.

stringer_t* st_vaprint_opts (uint32_t *opts*, chr_t * *format*, va_list *args*)

Return a managed string containing sprintf()-style formatted data.

Parameters:

opts an options value to be passed to the allocation of the resulting managed string.

format a format string for the output string data.

args a variable argument list of parameters to be formatted as output.

Returns:

NULL on failure or a managed string containing the final formatted data on success.

Definition at line 106 of file print.c.

References `length`, `log_pedantic`, `MEMORYBUF`, `st_alloc_opts()`, `st_data_get()`, `st_free()`, `st_info_opts()`, `st_length_set()`, `st_valid_destination()`, and `st_valid_tracked()`.

Referenced by `st_aprint()`, and `st_aprint_opts()`.

size_t st_vsprint (stringer_t * *s*, chr_t * *format*, va_list *args*)

Print to a managed string and return the number of bytes written.

See also:

`vsnprintf()`

Note:

If the destination string pointer is NULL the function will simply return how much room would have been necessary.

Parameters:

s a pointer to the managed string that will receive the output of the print operation.

format a format string specifying the arguments of the print operation.

args a va_list containing the parameters to the print format string.

Returns:

-1 on failure, or the number of characters printed to the string, excluding the terminating null byte.

Definition at line 24 of file print.c.

References `length`, `log_pedantic`, `MEMORYBUF`, `st_avail_get()`, `st_data_get()`, `st_info_opts()`, `st_length_set()`, `st_valid_destination()`, and `st_valid_tracked()`.

Referenced by `st_quick()`, and `st_sprint()`.

magma/core/strings/replace.c File Reference

Functions used for string replacement.

```
#include "magma.h"
```

Functions

- **int_t st_replace** (**stringer_t** **target, **stringer_t** *pattern, **stringer_t** *replacement)
- **replace.c stringer_t * st_swap** (**stringer_t** *target, **uchr_t** pattern, **uchr_t** replacement)

Replace all instances of one character in a managed string with another.

Detailed Description

Functions used for string replacement.

Definition in file **replace.c**.

Function Documentation

int_t st_replace (**stringer_t** ** *target*, **stringer_t** * *pattern*, **stringer_t** * *replacement*)

replace.c

Definition at line 24 of file replace.c.

References log_pedantic, PLACER, st_alloc, st_char_get(), st_cmp_cs_starts(), st_data_get(), st_empty_out(), st_free(), st_length_set(), and st_valid_free().

Referenced by contact_business(), imap_folder_create(), imap_folder_name_escaped(), imap_folder_rename(), pop_retr(), pop_top(), register_print_message(), register_print_step1(), register_print_step2(), and teacher_process().

stringer_t* st_swap (**stringer_t** * *target*, **uchr_t** *pattern*, **uchr_t** *replacement*)

Replace all instances of one character in a managed string with another.

Parameters:

target the input string which will be transformed by the character replacement.

pattern the character to be searched and replaced in the target string.

replacement the character to be substituted for the pattern character in the target string.

Returns:

a pointer to the target managed string.

Definition at line 114 of file replace.c.

References log_check, log_pedantic, and st_empty_out().

Referenced by process_kill().

magma/core/strings/shortcuts.c File Reference

A collection of shortcuts used to call various string functions using sensible default values.

```
#include "magma.h"
```

Functions

- **placer_t pl_null** (void)
- *Return a zero-length placer pointing to NULL data.* **placer_t pl_init** (void *data, size_t len)
- *Return a placer wrapping a data buffer of given size.* **placer_t pl_clone** (placer_t place)
- **placer_t pl_set** (placer_t place, placer_t set)
- void * **pl_data_get** (placer_t place)
- *Get a pointer to the data referenced by a placer.* **chr_t * pl_char_get** (placer_t place)
- *Get a character pointer to the data referenced by a placer.* **int_t pl_length_int** (placer_t place)
- *Get the length, in bytes, of a placer as an integer.* size_t **pl_length_get** (placer_t place)
- *Get the length, in bytes, of a placer.* **bool_t pl_empty** (placer_t place)
- *Determine whether or not the specified placer is empty.* **bool_t pl_starts_with_char** (placer_t place, chr_t c)
- *Determine if a placer begins with a specified character.* **bool_t pl_inc** (placer_t *place, bool_t more)

Advance the placer one character forward beyond an expected character.

Detailed Description

A collection of shortcuts used to call various string functions using sensible default values.

\$Author\$ \$Date\$ \$Revision:\$

Definition in file **shortcuts.c**.

Function Documentation

chr_t* pl_char_get (placer_t place)

Get a character pointer to the data referenced by a placer.

shortcuts.c

Parameters:

place the input placer.

Returns:

NULL on failure or a character pointer to the block of data associated with the specified placer on success.

Definition at line 59 of file shortcuts.c.

References [st_char_get\(\)](#).

Referenced by [cache_config\(\)](#), [con_write_pl\(\)](#), [ecies_key_private\(\)](#), [ecies_key_public\(\)](#), [imap_fetch_body_header\(\)](#), [imap_narrow_messages\(\)](#), [imap_search_messages_range\(\)](#), [ip_str_subnet\(\)](#), [lib_load_openssl\(\)](#), [line_pl_pl\(\)](#), [molten_parse\(\)](#), [nvp_parse\(\)](#), [pl_get_embraced\(\)](#), [pl_shrink_before_characters\(\)](#), [pl_skip_characters\(\)](#), [pl_skip_to_characters\(\)](#), [pl_starts_with_char\(\)](#), [pl_trim\(\)](#), [pl_trim_end\(\)](#), [pl_trim_start\(\)](#), [pop_process\(\)](#), [portal_message_body\(\)](#), [portal_upload\(\)](#), [relay_config\(\)](#), [servers_config\(\)](#), [smtp_auth_login\(\)](#), [smtp_auth_plain\(\)](#), [smtp_parse_helo_domain\(\)](#), [smtp_parse_mail_from_path\(\)](#), [smtp_parse_rcpt_to\(\)](#), [smtp_process\(\)](#), and [str_tok_get_bl\(\)](#).

placer_t pl_clone (placer_t *place*)

Definition at line 35 of file shortcuts.c.

References pl_init().

void* pl_data_get (placer_t *place*)

Get a pointer to the data referenced by a placer.

Parameters:

place the input placer.

Returns:

NULL on failure or a pointer to the block of data associated with the specified placer on success.

Definition at line 49 of file shortcuts.c.

References st_data_get().

Referenced by ecies_key_private(), ecies_key_public(), imap_build_array(), imap_build_array_isliteral(), imap_fetch_body(), imap_fetch_body_header(), imap_folder_create(), imap_folder_rename(), imap_parse_address_part(), nvp_parse(), smtp_check_filters(), tok_get_pl(), and uint64_conv_pl().

bool_t pl_empty (placer_t *place*)

Determine whether or not the specified placer is empty.

Parameters:

place the input placer.

Returns:

true if the placer is empty or zero-length, or false otherwise.

Definition at line 89 of file shortcuts.c.

References st_empty().

Referenced by cache_config(), client_read_line(), con_read_line(), http_data_value_parse(), http_process(), imap_build_array(), imap_build_array_isliteral(), imap_fetch_body(), imap_fetch_body_header(), imap_narrow_messages(), imap_parse_address(), imap_parse_literal(), imap_process(), imap_search_messages_date(), imap_search_messages_header(), imap_search_messages_range(), mail_add_forward_headers(), mail_add_outbound_headers(), mail_mime_child(), mail_mime_count(), mail_mod_subject(), molten_parse(), multipart_get_boundary(), nvp_parse(), pl_get_embraced(), pl_inc(), pl_shrink_before_characters(), pl_skip_characters(), pl_skip_to_characters(), pl_starts_with_char(), pop_process(), portal_upload(), relay_config(), servers_config(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(), smtp_parse_rcpt_to(), and smtp_process().

bool_t pl_inc (placer_t * *place*, bool_t *more*)

Advance the placer one character forward beyond an expected character.

Parameters:

place the input placer.

more if true, the placer must contain more data, and vice versa.

Returns:

true if *more* was true and the placer contains more data, or if *more* was false and the placer ended; false otherwise.

Definition at line 119 of file shortcuts.c.

References `pl_empty()`.

placer_t pl_init (void * *data*, size_t *len*)

Return a placer wrapping a data buffer of given size.

Parameters:

data a pointer to the data to be wrapped.

len the length, in bytes, of the data.

Returns:

a placer pointing to the specified data.

Definition at line 30 of file shortcuts.c.

References FOREIGNDATA, JOINTED, PLACER_T, `placer_t`, and STACK.

Referenced by `bracket_extract_pl()`, `ecies_decrypt()`, `ecies_encrypt()`, `get_header_value_noopt()`, `http_parse_context()`, `http_parse_origin()`, `http_parse_pairs()`, `imap_build_array()`, `imap_fetch_body()`, `imap_fetch_body_portion()`, `imap_fetch_bodystructure()`, `imap_fetch_envelope()`, `imap_parse_address_breaker()`, `imap_search_messages_date()`, `imap_search_messages_header()`, `line_pl_bl()`, `mail_header_pop()`, `mail_mime_child()`, `mail_mime_header()`, `mail_mime_part()`, `mail_mime_update()`, `mail_store_header()`, `pl_clone()`, `pl_trim()`, `pl_trim_end()`, `pl_trim_start()`, `smtp_check_filters()`, `str_tok_get_bl()`, `str_tok_get_count_bl()`, `tok_get_ns()`, and `tok_pop()`.

size_t pl_length_get (placer_t *place*)

Get the length, in bytes, of a placer.

Parameters:

place the input placer.

Returns:

the size, in bytes, of the specified placer.

Definition at line 79 of file shortcuts.c.

References `st_length_get()`.

Referenced by `cache_config()`, `client_read()`, `client_read_line()`, `con_read()`, `con_read_line()`, `con_write_pl()`, `ecies_key_private()`, `ecies_key_public()`, `imap_build_array()`, `imap_build_array_isliteral()`, `imap_fetch_body()`, `imap_folder_create()`, `imap_folder_rename()`, `imap_parse_address_part()`, `imap_parse_literal()`, `ip_str_subnet()`, `line_pl_pl()`, `molten_parse()`, `nvp_parse()`, `pl_shrink_before_characters()`, `pl_trim()`, `pl_trim_end()`, `pl_trim_start()`, `pop_process()`, `portal_message_body()`, `portal_upload()`, `relay_config()`, `servers_config()`, `smtp_auth_login()`, `smtp_auth_plain()`, `smtp_check_filters()`, `smtp_parse_helo_domain()`, `smtp_parse_mail_from_path()`, `smtp_parse_rcpt_to()`, `smtp_process()`, `str_tok_get_bl()`, `str_tok_get_count_bl()`, `tok_get_pl()`, and `uint64_conv_pl()`.

int_t pl_length_int (placer_t *place*)

Get the length, in bytes, of a placer as an integer.

Parameters:

place the input placer.

Returns:

the size, in bytes, of the specified placer.

Definition at line 69 of file shortcuts.c.

References `st_length_int()`.

Referenced by `lib_load_openssl()`, `smtp_parse_helo_domain()`, `smtp_parse_mail_from_path()`, and `smtp_parse_rcpt_to()`.

placer_t pl_null (void)

Return a zero-length placer pointing to NULL data.

Returns:

a zero-length placer pointing to NULL data.

Definition at line 19 of file shortcuts.c.

References `FOREIGNDATA`, `JOINTED`, `PLACER_T`, `placer_t`, and `STACK`.

Referenced by `bracket_extract_pl()`, `client_read()`, `client_read_line()`, `con_init_network_buffer()`, `con_read()`, `con_read_line()`, `get_header_opt()`, `imap_fetch_body()`, `imap_fetch_body_portion()`, `imap_fetch_envelope()`, `imap_narrow_messages()`, `imap_parse_address_breaker()`, `imap_parse_literal()`, `imap_search_messages_date()`, `imap_search_messages_header()`, `imap_search_messages_range()`, `imap_starttls()`, `line_pl_bl()`, `mail_header_pop()`, `mail_mime_child()`, `nvp_parse()`, `pl_trim()`, `pl_trim_end()`, `pl_trim_start()`, `pop_starttls()`, `portal_upload()`, `smtp_check_filters()`, `smtp_starttls()`, `str_tok_get_bl()`, `tok_get_ns()`, and `tok_pop()`.

placer_t pl_set (placer_t *place*, placer_t *set*)

Definition at line 39 of file shortcuts.c.

References `placer_t`.

Referenced by `mail_mime_split()`.

bool_t pl_starts_with_char (placer_t *place*, chr_t *c*)

Determine if a placer begins with a specified character.

Parameters:

place the input placer.

c the character to be compared with the first byte of the placer's data.

Returns:

true if the placer begins with the given character or false otherwise.

Definition at line 100 of file shortcuts.c.

References `pl_char_get()`, and `pl_empty()`.

Referenced by `nvp_parse()`.

magma/core/strings/strings.h File Reference

Function declarations and types used by the different modules involved with handling stringers and null terminated strings.

Data Structures

- struct **multi_t**

Defines

- #define **st_append**(s, append) st_append_opts(1024, s, append)
- #define **st_alloc**(len) st_alloc_opts(MANAGED_T | CONTIGUOUS | HEAP, len)
- #define **st_merge**(...) st_merge_opts(MANAGED_T | CONTIGUOUS | HEAP, __VA_ARGS__)
- #define **st_vaprint**(format, args) st_vaprint_opts(MANAGED_T | CONTIGUOUS | HEAP, format, args)
- #define **CONSTANT**(string) (**stringer_t***)((**constant_t***)"\\x41\\x01\\x00\\x00" string)
- #define **NULLER**(d) (**stringer_t***)&((**nuller_t**){ .opts = (NULLER_T | JOINTED | STACK | FOREIGNDATA), .data = d })
- #define **BLOCK**(d, l) (**stringer_t***)&((**block_t**){ .opts = (BLOCK_T | JOINTED | STACK | FOREIGNDATA), .data = d, .length = l })
- #define **PLACER**(d, l) (**stringer_t***)&((**placer_t**){ .opts = (PLACER_T | JOINTED | STACK | FOREIGNDATA), .data = d, .length = l })
- #define **MANAGED**(d, l, a) (**stringer_t***)&((**managed_t**){ .opts = (MANAGED_T | JOINTED | STACK | FOREIGNDATA), .data = d, .length = l, .avail = a })
- #define **BLOCKBUF**(l) (**stringer_t***)&((**block_t**){ .opts = (BLOCK_T | CONTIGUOUS | STACK), .data = &((**chr_t** []){ [0 ... 1] = 0 }), .length = l })
- #define **MANAGEDBUF**(l) (**stringer_t***)&((**managed_t**){ .opts = (MANAGED_T | CONTIGUOUS | STACK), .data = &((**chr_t** []){ [0 ... 1] = 0 }), .length = 0, .avail = l })

Typedefs

- typedef void **stringer_t**

Enumerations

- enum { **CONSTANT_T** = 1, **PLACER_T** = 2, **NULLER_T** = 4, **BLOCK_T** = 8, **MANAGED_T** = 16, **MAPPED_T** = 32, **CONTIGUOUS** = 64, **JOINTED** = 128, **STACK** = 256, **HEAP** = 512, **SECURE** = 1024, **FOREIGNDATA** = 4096 }

Functions

- struct **__attribute__**((packed))
- **chr_t** * **ns_alloc** (size_t len)
- *nuller.c* **chr_t** * **ns_append** (**chr_t** *s, **chr_t** *append)
- *Append one string to another and return the result.* **chr_t** * **ns_dupe** (**chr_t** *s)
- *Duplicate a null-terminated string.* **bool_t** **ns_empty** (**chr_t** *s)
- *Return whether a null-terminated string is empty or not.* **bool_t** **ns_empty_out** (**chr_t** *s, **chr_t** **ptr, size_t *len)
- *Return whether a null-terminated string is empty or not, while storing its address and length.* void **ns_free** (**chr_t** *s)
- *Free a null-terminated string.* void **ns_cleanup** (**chr_t** *s)
- *A checked null-terminated string free front-end function.* **chr_t** * **ns_import** (void *block, size_t len)
- *Get a block of memory as a null-terminated string.* size_t **ns_length_get** (const **chr_t** *s)
- *Return the length of a null-terminated string.* int **ns_length_int** (**chr_t** *s)
- *Return the length of a null-terminated string as an int, capped at INT_MAX.* void **ns_wipe** (**chr_t** *s, size_t len)

- Zero out a null-terminated string. `const chr_t * st_info_type (uint32_t opts)`
- Get a readable description of a managed string's data type. `const chr_t * st_info_layout (uint32_t opts)`
- Get a readable description of a managed string's data layout. `const chr_t * st_info_allocator (uint32_t opts)`
- Get a readable description of a managed string's allocator type. `chr_t * st_info_opts (uint32_t opts, chr_t *s, size_t len)`
- Get a readable description of all of a managed string's option flags. `bool_t st_valid_free (uint32_t opts)`
- Determine whether a managed string is allowed to be freed. `bool_t st_valid_opts (uint32_t opts)`
- Check to see that a managed string has a valid combination of allocation options. `bool_t st_valid_avail (uint32_t opts)`
- Determine whether a managed string tracks the total allocated buffer size. `bool_t st_valid_append (uint32_t opts)`
- Determine whether the managed string options allow for data appending. `bool_t st_valid_placer (uint32_t opts)`
- A sanity check to determine whether the managed string is a valid placer. `bool_t st_valid_joined (uint32_t opts)`
- Determine whether the managed string options are a validly joined. `bool_t st_valid_tracked (uint32_t opts)`
- Determine whether a managed string provides for data length tracking. `bool_t st_valid_destination (uint32_t opts)`
- Determine whether the managed string options allow for a data store operation. `int_t st_length_int (stringer_t *s)`
- `size_t st_avail_get (stringer_t *s)`
- Return the total data buffer size allocated for a managed string. `size_t st_length_get (stringer_t *s)`
- Return the length of the data in a managed string. `size_t st_avail_set (stringer_t *s, size_t avail)`
- Set the total data buffer size allocated for a managed string. `size_t st_length_set (stringer_t *s, size_t len)`
- Set the data length for a managed string that supports data length tracking. `chr_t * st_char_get (stringer_t *s)`
- `data.c uchr_t * st_uchar_get (stringer_t *s)`
- Retrieve an unsigned character pointer to a managed string's data buffer. `void * st_data_get (stringer_t *s)`
- Retrieve the data associated with a managed string. `void st_data_set (stringer_t *s, void *data)`
- Set the underlying data of a jointed managed string. `bool_t st_empty (stringer_t *s)`
- Determine whether the specified managed string is empty or not. `bool_t st_empty_out (stringer_t *s, uchr_t **ptr, size_t *len) __attribute__((nonnull(2)))`
- `bool_t void st_wipe (stringer_t *s)`
- Wipe all of a managed string's allocated memory and if applicable, reset its length. `void st_free (stringer_t *s)`
- Free a managed string. `void st_cleanup (stringer_t *s)`
- A checked managed string free front-end function. `stringer_t * st_dupe (stringer_t *s)`
- Duplicate a managed string. `stringer_t * st_import (const void *s, size_t len)`
- Create a new (contiguous managed) managed string on the heap to hold a copy of the specified data buffer. `stringer_t * st_copy_in (stringer_t *s, void *buf, size_t len)`
- Copy data into a managed string. `stringer_t * st_realloc (stringer_t *s, size_t len)`
- Reallocate a managed string to a specified size. `stringer_t * st_output (stringer_t *output, size_t len)`
- Return a suitable managed string to store data of a specified length. `stringer_t * st_nullify (chr_t *input, size_t len)`
- Create a null-terminated string out of a block of memory and return it as a newly allocated nuller. `stringer_t * st_alloc_opts (uint32_t opts, size_t len)`
- Allocate a managed string with a specified options mask. `stringer_t * st_dupe_opts (uint32_t opts, stringer_t *s)`
- Create a duplicate copy of a managed string with a specified set of allocation options. `stringer_t * st_merge_opts (uint32_t opts, chr_t *format,...)`
- Concatenate a variable number of user-supplied multi-type strings. `stringer_t * st_append_opts (size_t align, stringer_t *s, stringer_t *append)`
- Append one managed string to another, with aligned memory boundaries. `chr_t * pl_char_get (placer_t place)`
- `shortcuts.c void * pl_data_get (placer_t place)`
- Get a pointer to the data referenced by a placer. `bool_t pl_empty (placer_t place)`

- Determine whether or not the specified placer is empty. **placer_t pl_init** (void *data, size_t len)
- Return a placer wrapping a data buffer of given size. **placer_t pl_clone** (placer_t place)
- size_t **pl_length_get** (placer_t place)
- Get the length, in bytes, of a placer. **int_t pl_length_int** (placer_t place)
- Get the length, in bytes, of a placer as an integer. **placer_t pl_null** (void)
- Return a zero-length placer pointing to NULL data. **placer_t pl_set** (placer_t place, placer_t set)
- **bool_t pl_starts_with_char** (placer_t place, chr_t c)
- Determine if a placer begins with a specified character. **bool_t pl_inc** (placer_t *place, bool_t more)
- Advance the placer one character forward beyond an expected character. **bool_t st_opt_get** (stringer_t *s, uint32_t opt)
- **opts.c** **int_t st_opt_set** (stringer_t *s, uint32_t opt, bool_t enabled)
- Enable or disable a set of option(s) for a managed string, with validity testing. **stringer_t * st_aprint** (chr_t *format,...) **__attribute__((format(printf**
- **print.c** **stringer_t stringer_t * st_aprint_opts** (uint32_t opts, chr_t *format,...) **__attribute__((format(printf**
- **stringer_t stringer_t stringer_t * st_quick** (stringer_t *s, chr_t *format,...) **__attribute__((format(printf**
- **stringer_t stringer_t stringer_t size_t st_sprint** (stringer_t *s, chr_t *format,...) **__attribute__((format(printf**
- **stringer_t stringer_t stringer_t size_t stringer_t * st_vaprint_opts** (uint32_t opts, chr_t *format, va_list args)
- Return a managed string containing sprintf()-style formatted data. **size_t st_vsprint** (stringer_t *s, chr_t *format, va_list args)
- Print to a managed string and return the number of bytes written. **int_t st_replace** (stringer_t **target, stringer_t *pattern, stringer_t *replacement)
- **replace.c** **stringer_t * st_swap** (stringer_t *target, uchr_t pattern, uchr_t replacement)
- Replace all instances of one character in a managed string with another. **int32_t cmp_mt_mt** (multi_t one, multi_t two)
- **multi.c** **bool_t ident_mt_mt** (multi_t one, multi_t two)
- Check to see if the values of two multi-type objects are identical. **multi_t mt_dupe** (multi_t multi)
- Duplicate a multi-type object. **void mt_free** (multi_t multi)
- Free the data underlying a multi-type data object (for managed and null-terminated strings). **char * mt_get_char** (multi_t multi)
- Get a character pointer to the value of a multi-type object. **size_t mt_get_length** (multi_t multi)
- Get the length of the data associated with a multi-type object. **multi_t mt_get_null** (void)
- Return an empty multi-type object. **uint64_t mt_get_number** (multi_t multi)
- Get the value of a numerical multi-type object. **M_TYPE mt_get_type** (multi_t multi)
- Get the data type associated with a multi-type object. **bool_t mt_is_empty** (multi_t multi)
- Determine whether a multi-type object is empty. **bool_t mt_is_number** (multi_t multi)
- Determine whether a multi-type object is a number. **multi_t mt_set_type** (multi_t multi, M_TYPE target)

Set the data type of a multi-type data object. Variables

- **constant_t**
- **nuller_t**
- **block_t**
- **placer_t**
- **managed_t**
- **mapped_t**

Detailed Description

Function declarations and types used by the different modules involved with handling stringers and null terminated strings.

Definition in file **strings.h**.

Define Documentation

```
#define BLOCK(d, l) (stringer_t *)&((block_t){ .opts = (BLOCK_T | JOINTED | STACK |  
FOREIGNDATA), .data = d, .length = l })
```

Definition at line 190 of file strings.h.

```
#define BLOCKBUF(l) (stringer_t *)&((block_t){ .opts = (BLOCK_T | CONTIGUOUS | STACK), .data  
= &((chr_t []){ [ 0 ... l ] = 0 }), .length = l })
```

Definition at line 199 of file strings.h.

```
#define CONSTANT(string) (stringer_t *)((constant_t *)"\x41\x01\x00\x00" string)
```

Definition at line 184 of file strings.h.

Referenced by cache_config(), cache_set_value(), config_load_cmdline_settings(), config_load_database_settings(), config_load_file_settings(), config_output_value_generic(), config_value_set(), get_temp_file_handle(), lib_load(), mail_add_forward_headers(), mail_add_inbound_headers(), mail_mod_subject(), meta_folders_by_name(), relay_config(), relay_set_value(), sanity_check(), serial_reset(), servers_config(), servers_output_settings(), servers_set_value(), servers_validate(), smtp_parse_mail_from_path(), stats_sum_errors(), and virus_check().

```
#define MANAGED(d, l, a) (stringer_t *)&((managed_t){ .opts = (MANAGED_T | JOINTED |  
STACK | FOREIGNDATA), .data = d, .length = l, .avail = a })
```

Definition at line 196 of file strings.h.

```
#define MANAGEDBUF(l) (stringer_t *)&((managed_t){ .opts = (MANAGED_T | CONTIGUOUS |  
STACK), .data = &((chr_t []){ [ 0 ... l ] = 0 }), .length = 0, .avail = l })
```

Definition at line 202 of file strings.h.

Referenced by contact_abuse_checks(), contact_abuse_increment_history(), contact_business(), http_response_allow_cross(), http_response_cookie(), http_response_header(), http_response_options(), imap_id(), imap_login(), imap_search(), lib_load(), lock_get(), lock_release(), mail_add_inbound_headers(), mail_add_outbound_headers(), pop_pass(), portal_meta(), process_kill(), register_abuse_check_blocklist(), register_abuse_checks(), register_abuse_increment_history(), register_data_insert_user(), register_session_cache(), register_session_get(), scramble_decrypt(), scramble_encrypt(), smtp_bounce(), smtp_check_greylist(), smtp_check_rbl(), smtp_rcpt_to(), smtp_reply(), user_lock(), and user_unlock().

```
#define NULLER(d) (stringer_t *)&((nuller_t){ .opts = (NULLER_T | JOINTED | STACK |  
FOREIGNDATA), .data = d })
```

Definition at line 187 of file strings.h.

Referenced by args_parse(), cache_config(), cmp_mt_mt(), config_free(), config_key_lookup(), config_output_value(), config_output_value_generic(), config_value_set(), contact_business(), contact_business_add_error(), contact_print_form(), contact_print_message(), contact_process(), http_content_load_directory(), http_content_load_fonts(), http_content_start(), http_data_get(), http_get_template(), http_load_file(), http_parse_context(), http_response(), imap_list(), imap_lsub(), imap_status(), ip_str_subnet(), lib_load(), lib_symbols(), log_start(), mail_mime_get_smtp_envelope(), multipart_get_boundary(), portal_endpoint_auth(), portal_endpoint_config_edit(), portal_endpoint_contacts_add(), portal_endpoint_contacts_edit(), portal_endpoint_error(), portal_endpoint_folders_add(), portal_endpoint_folders_list(), portal_endpoint_folders_remove(), portal_endpoint_folders_rename(), portal_endpoint_folders_tags(), portal_endpoint_messages_flag(), portal_endpoint_messages_send(), portal_endpoint_messages_tag(), portal_endpoint_response(), portal_parse_flags(), portal_parse_sections(), portal_upload(), register_abuse_checks(), register_abuse_increment_history(), register_captcha_random_font(), register_print_message(), register_session_cache(), register_session_get(), relay_config(), sanity_check(), servers_config(), servers_output_settings(), servers_set_value(), servers_validate(), spool_check_file(), st_info_opts(), stats_get_name_pos(), and stats_sum_errors().

```
#define PLACER(d, l) (stringer_t *)&((placer_t){ .opts = (PLACER_T | JOINTED | STACK |  
FOREIGNDATA), .data = d, .length = l })
```

Definition at line 193 of file strings.h.

Referenced by args_parse(), cache_config(), config_free(), config_load_database_settings(), config_output_value(), config_output_value_generic(), config_validate_settings(), config_value_set(), contact_business(), contact_business_add_error(), contact_details_fetch(), contact_edit(), contact_print_form(), contact_print_message(), contact_process(), contacts_fetch(), credential_address(), credential_alloc_auth(), credential_alloc_mail(), credential_username(), http_content_load_directory(), http_content_load_fonts(), http_content_start(), http_load_file(), http_parse_header(), http_parse_method(), http_parse_origin(), http_parse_pairs(), http_print_400(), http_print_403(), http_print_404(), http_print_405(), http_print_500(), http_print_500_log(), http_print_501(), http_response(), http_response_allow_cross(), http_response_connection(), http_response_cookie(), ident_mt_mt(), imap_command_log_safe(), imap_command_parser(), imap_compare(), imap_fetch(), imap_fetch_body(), imap_fetch_body_mime(), imap_fetch_bodystructure(), imap_fetch_envelope(), imap_fetch_message(), imap_flag_action(), imap_folder_compare(), imap_folder_create(), imap_folder_name_escaped(), imap_folder_remove(), imap_folder_rename(), imap_get_flag(), imap_list(), imap_lsub(), imap_parse_dataitems(), imap_search_messages_date(), imap_search_messages_date_compare(), imap_search_messages_inner(), imap_status(), lock_get(), magma_folder_fetch(), magma_folder_find_full_name(), magma_folder_find_name(), magma_folder_name(), mail_add_forward_headers(), mail_add_outbound_headers(), mail_add_required_headers(), mail_count_received(), mail_discover_encoding(), mail_discover_insertion_point(), mail_discover_type(), mail_get_boundary(), mail_get_chunk(), mail_header_fetch_all(), mail_header_fetch_cleaned(), mail_insert_chunk_base64(), mail_insert_chunk_text(), mail_load_header(), mail_load_message(), mail_mime_boundary(), mail_mime_content_encoding(), mail_mime_content_id(), mail_mime_encoding(), mail_mime_type(), mail_mime_type_group(), mail_mime_type_parameters(), mail_mime_type_sub(), mail_mod_subject(), mail_modify_part(), mail_signature_add(), messages_fetch(), meta_data_fetch_alerts(), meta_data_fetch_mailbox_aliases(), molten_compare(), pop_compare(), pop_retr(), pop_top(), portal_endpoint(), portal_endpoint_attachments_progress(), portal_endpoint_compare(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_cookies(), portal_endpoint_error(), portal_endpoint_folders_list(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(), portal_endpoint_messages_tag(), portal_endpoint_response(), portal_endpoint_search(), portal_message_attachments(), portal_message_body(), portal_message_header(), portal_parse_action(), portal_parse_context(), portal_parse_flags(), portal_parse_sections(), portal_process(), portal_upload(), qp_encode(), register_business_step2(), register_captcha_random_font(), register_print_message(), register_print_step1(), register_print_step2(), relay_config(), scramble_decrypt(), servers_config(), sess_token(), signal_shutdown(), smtp_accept_message(), smtp_auth_login(),

smtp_auth_plain(), smtp_check_filters(), smtp_client_close(), smtp_client_send_data(), smtp_compare(), smtp_data_outbound(), smtp_fetch_autoreply(), smtp_get_action(), smtp_parse_mail_from_path(), smtp_update_receive_stats(), spool_check_file(), st_replace(), stamp_counter_check(), stamp_counter_increment(), symmetric_vector(), teacher_data_get(), teacher_data_save(), teacher_process(), tran_commit(), tran_rollback(), tran_start(), url_encode(), user_config_fetch(), virus_check(), warehouse_fetch_domains(), xml_get_xpath_int16(), xml_get_xpath_int32(), xml_get_xpath_int64(), xml_get_xpath_int8(), xml_get_xpath_uint16(), xml_get_xpath_uint32(), xml_get_xpath_uint64(), and xml_get_xpath_uint8().

#define st_alloc(len) st_alloc_opts(MANAGED_T | CONTIGUOUS | HEAP, len)

Definition at line 179 of file strings.h.

Referenced by base64_decode(), base64_decode_mod(), base64_encode(), base64_encode_mod(), client_connect(), con_init_network_buffer(), decompress_block_lzo(), decompress_bzip(), decompress_lzo(), decompress_zlib(), deserialize_st(), digest_hash(), file_load(), hex_decode_st(), hex_encode_st(), hex_encode_st_debug(), http_load_file(), imap_build_array(), imap_parse_address_part(), imap_parse_literal(), imap_parse_qstring(), ip_presentation(), ip_reversed(), ip_standard(), ip_subnet(), mail_add_required_headers(), mail_build_signature(), mail_extract_address(), mail_extract_tag(), mail_header_fetch_cleaned(), mail_load_message(), mail_mime_generate_boundary(), portal_smtp_merge_headers(), qp_decode(), rand_choices(), sanity_check(), serialize_int16(), serialize_int32(), serialize_int64(), serialize_uint16(), serialize_uint32(), serialize_uint64(), st_output(), st_replace(), symmetric_decrypt(), symmetric_encrypt(), url_decode(), zbase32_decode(), and zbase32_encode().

#define st_append(s, append) st_append_opts(1024, s, append)

Definition at line 178 of file strings.h.

Referenced by args_parse(), get_temp_file_handle(), http_response_allow_cross(), http_response_cookie(), magma_folder_name(), qp_encode(), st_info_opts(), and url_encode().

#define st_merge(...) st_merge_opts(MANAGED_T | CONTIGUOUS | HEAP, __VA_ARGS__)

Definition at line 180 of file strings.h.

Referenced by contact_business(), dkim_create(), http_content_load_directory(), http_content_load_fonts(), http_print_301(), imap_build_array_isliteral(), imap_fetch_body(), imap_fetch_body_header(), imap_fetch_body_mime(), imap_fetch_body_tag(), imap_fetch_bodystructure(), imap_fetch_message(), imap_folder_create(), imap_folder_name_escaped(), imap_folder_rename(), imap_narrow_folders(), imap_parse_address(), imap_range_build(), imap_search_messages_date(), mail_add_required_headers(), mail_build_signature(), mail_get_boundary(), mail_header_fetch_all(), mail_insert_chunk_base64(), mail_insert_chunk_text(), mail_mime_boundary(), mail_mime_encode_part(), mail_mime_get_smtp_envelope(), mail_mod_subject(), meta_folders_name(), meta_get(), portal_endpoint_contacts_copy(), portal_folder_mail_add(), portal_message_attachments(), portal_smtp_create_data(), portal_smtp_merge_headers(), register_business_step2(), register_data_insert_user(), sess_token(), smtp_bounce(), and smtp_reply().

#define st_vaprint(format, args) st_vaprint_opts(MANAGED_T | CONTIGUOUS | HEAP, format, args)

Definition at line 181 of file strings.h.

Referenced by client_print().

Typedef Documentation

typedef void stringer_t

Definition at line 82 of file strings.h.

Enumeration Type Documentation

anonymous enum

Enumerator:

CONSTANT_T
PLACER_T
NULLER_T
BLOCK_T
MANAGED_T
MAPPED_T
CONTIGUOUS
JOINTED
STACK
HEAP
SECURE
FOREIGNDATA

Definition at line 19 of file strings.h.

Function Documentation

struct __attribute__((packed)) [read]

Definition at line 74 of file strings.h.

References `__attribute__::data`, and `__attribute__::length`.

int32_t cmp_mt_mt (multi_t *one*, multi_t *two*)

multi.c

multi.c

Note:

The types of the two input objects must match, or be a comparison between a managed string and null-terminated string.

Parameters:

one the first multi-type object to be compared.

two the second multi-type object to be compared.

Returns:

Definition at line 686 of file multi.c.

References multi_t::binary, multi_t::dbl, multi_t::fl, multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, log_options, log_pedantic, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, mt_get_type(), multi_t::ns, NULLER, multi_t::st, st_cmp_cs_eq(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by tree_cmp().

bool_t ident_mt_mt (multi_t *one*, multi_t *two*)

Check to see if the values of two multi-type objects are identical.

Note:

The types of the two input objects must match, or be a comparison between a managed string and null-terminated string.

Parameters:

one the first multi-type object to be compared.

two the second multi-type object to be compared.

Returns:

true if the two objects have identical values; false if they don't, or on failure.

Definition at line 568 of file multi.c.

References multi_t::binary, multi_t::dbl, multi_t::fl, multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, log_options, log_pedantic, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, mt_get_type(), multi_t::ns, ns_length_get(), PLACER, multi_t::st, st_cmp_cs_eq(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by hashed_bucket_find_key(), hashed_delete(), linked_delete(), and linked_find().

multi_t mt_dupe (multi_t *multi*)

Duplicate a multi-type object.

Parameters:

multi the multi-type object to be examined.

Returns:

a copy of the input object (a deep copy for managed strings and null-terminated strings).

Definition at line 485 of file multi.c.

References multi_t::binary, CONTIGUOUS, multi_t::dbl, multi_t::fl, HEAP, multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, MANAGED_T, mt_get_null(), multi_t::ns, ns_dupe(), multi_t::st, st_dupe_opts(), type(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by hashed_bucket_alloc(), linked_record_alloc(), and tree_insert().

void mt_free (multi_t *multi*)

Free the data underlying a multi-type data object (for managed and null-terminated strings).

Parameters:

multi the multi-type object to be freed.

Returns:

This function returns no value.

Definition at line 463 of file multi.c.

References M_TYPE_NULLER, M_TYPE_STRINGER, multi_t::ns, ns_free(), multi_t::st, st_free(), multi_t::type, and multi_t::val.

Referenced by hashed_delete(), hashed_free(), hashed_truncate(), linked_record_free(), tree_delete(), tree_insert(), and tree_truncate().

char* mt_get_char (multi_t *multi*)

Get a character pointer to the value of a multi-type object.

Parameters:

multi the multi-type object to be examined.

a a pointer to the value of the input object, or NULL on failure.

Definition at line 176 of file multi.c.

References multi_t::binary, multi_t::bl, multi_t::dbl, multi_t::fl, multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, log_options, M_LOG_INFO, M_LOG_STACK_TRACE, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, multi_t::ns, multi_t::st, st_char_get(), type(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by hashed_bucket().

size_t mt_get_length (multi_t *multi*)

Get the length of the data associated with a multi-type object.

Parameters:

multi the multi-type object to be examined.

Returns:

the length in bytes of the data associated with the input object, or 0 on failure.

Definition at line 250 of file multi.c.

References log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, multi_t::ns, ns_length_get(), multi_t::st, st_length_get(), type(), multi_t::type, and multi_t::val.

Referenced by hashed_bucket().

multi_t mt_get_null (void)

Return an empty multi-type object.

Returns:

an empty multi-type object.

Definition at line 22 of file multi.c.

References EMPTY, and multi_t::type.

Referenced by hashed_cursor_key_active(), hashed_cursor_key_next(), inx_cursor_key_active(), inx_cursor_key_next(), linked_cursor_key_active(), linked_cursor_key_next(), linked_record_get_key(), mt_dupe(), tree_cursor_key_next(), and tree_cursor_next().

uint64_t mt_get_number (multi_t *multi*)

Get the value of a numerical multi-type object.

Parameters:

multi the multi-type object to be examined.

Returns:

the numerical value of the input object, or 0 on failure.

Definition at line 121 of file multi.c.

References multi_t::dbl, EMPTY, multi_t::fl, multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, type(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by hashed_bucket().

M_TYPE mt_get_type (multi_t *multi*)

Get the data type associated with a multi-type object.

Parameters:

multi the multi-type object to be examined.

Returns:

the data type of the input object, or EMPTY on failure.

Definition at line 319 of file multi.c.

References EMPTY, log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_PLACER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, type(), and multi_t::type.

Referenced by cmp_mt_mt(), and ident_mt_mt().

bool_t mt_is_empty (multi_t *multi*)

Determine whether a multi-type object is empty.

Note:

All numeric (including boolean and floating point) types will always return false.

Parameters:

multi the multi-type object to be examined.

Returns:

true if the object is empty (by type or value), or false otherwise.

Definition at line 36 of file multi.c.

References multi_t::bl, EMPTY, log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, multi_t::ns, ns_empty(), multi_t::st, st_empty(), type(), multi_t::type, and multi_t::val.

Referenced by config_load_cmdline_settings(), config_load_file_settings(), and tree_insert().

bool_t mt_is_number (multi_t *multi*)

Determine whether a multi-type object is a number.

Parameters:

multi the multi-type object to be examined. return true if the input object is any integer, boolean, or floating point; false otherwise.

Definition at line 85 of file multi.c.

References EMPTY, log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, type(), and multi_t::type.

Referenced by hashed_bucket().

multi_t mt_set_type (multi_t *multi*, M_TYPE *target*)

Set the data type of a multi-type data object.

Parameters:

multi the multi-type object to be adjusted.

target the value of the new data type for the input object.

Returns:

a copy of the modified multi-type data object.

Definition at line 392 of file multi.c.

References EMPTY, log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64,

M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_PLACER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, type(), and multi_t::type.

Referenced by nvp_parse().

chr_t* ns_alloc (size_t len)

nuller.c

nuller.c

Parameters:

len the length of the buffer to be allocated.

Returns:

NULL on failure or if len was 0; a pointer to the newly allocated memory otherwise.

Definition at line 71 of file nuller.c.

References log_pedantic, mm_alloc(), and ns_wipe().

Referenced by deserialize_ns(), mail_message_path(), and ns_append().

chr_t* ns_append (chr_t * s, chr_t * append)

Append one string to another and return the result.

Parameters:

s a pointer to a leading null-terminated string to which the other string will be appended.

append a pointer to a null-terminated string to be appended to the leading string.

Returns:

NULL if a memory allocation failure occurred, or a pointer to the resulting string on success.

Definition at line 173 of file nuller.c.

References log_pedantic, mm_copy(), ns_alloc(), ns_dupe(), ns_free(), and ns_length_get().

Referenced by portal_endpoint_error(), and portal_endpoint_response().

void ns_cleanup (chr_t * s)

A checked null-terminated string free front-end function.

See also:

ns_free()

Parameters:

s the null-terminated string to be freed.

Returns:

This function returns no value.

Definition at line 115 of file nuller.c.

References ns_free().

chr_t* ns_dupe (chr_t * s)

Duplicate a null-terminated string.

Parameters:

s the null-terminated string to be duplicated.

Returns:

NULL on failure, or a pointer to a copy of the input string.

Definition at line 129 of file nuller.c.

References `length`, `log_info`, `log_pedantic`, `mm_alloc()`, `mm_copy()`, and `ns_length_get()`.

Referenced by `cache_set_value()`, `config_value_set()`, `mt_dupe()`, `ns_append()`, `relay_set_value()`, `servers_set_value()`, and `xml_get_xpath_ns()`.

bool_t ns_empty (chr_t * s)

Return whether a null-terminated string is empty or not.

Parameters:

s the input as a null-terminated string.

Returns:

true if string is NULL or zero length; false otherwise.

Definition at line 32 of file nuller.c.

References `ns_length_get()`.

Referenced by `cache_free()`, `cache_output_settings()`, `cache_set_value()`, `cache_validate()`, `config_output_value()`, `config_output_value_generic()`, `config_value_set()`, `mt_is_empty()`, `relay_free()`, `relay_output_settings()`, `relay_set_value()`, `relay_validate()`, `servers_free()`, `servers_output_settings()`, `servers_set_value()`, `servers_validate()`, `sql_start()`, and `st_merge_opts()`.

bool_t ns_empty_out (chr_t * s, chr_t ** ptr, size_t * len)

Return whether a null-terminated string is empty or not, while storing its address and length.

Parameters:

s the input as a null-terminated string.

ptr a pointer address to receive a copy of the string's location.

len a pointer to a variable to receive the length of the string, in bytes.

Returns:

true if string is NULL or zero length; false otherwise.

Definition at line 44 of file nuller.c.

References `ns_length_get()`.

void ns_free (chr_t * s)

Free a null-terminated string.

Parameters:

s the string to be freed.

Returns:

This function returns no value.

Definition at line 98 of file nuller.c.

References log_pedantic, and mm_free().

Referenced by adjust_message_encryption(), cache_free(), cache_set_value(), config_free(), config_value_set(), ip_str_subnet(), mail_copy_message(), mail_load_header(), mail_load_message(), mail_message_path(), mail_remove_message(), mail_store_message(), mail_store_message_data(), mt_free(), ns_append(), ns_cleanup(), portal_endpoint_error(), portal_endpoint_response(), relay_free(), relay_set_value(), servers_free(), and servers_set_value().

chr_t* ns_import (void * *block*, size_t *len*)

Get a block of memory as a null-terminated string.

Parameters:

block a pointer to the buffer containing the data to be copied.

len the length, in bytes, of the data buffer to be copied.

Returns:

NULL on failure, or a pointer to the newly allocated null-terminated string on success.

Definition at line 154 of file nuller.c.

References log_pedantic, mm_alloc(), and mm_copy().

Referenced by adjust_message_encryption(), cache_set_value(), config_value_set(), ip_str_subnet(), relay_set_value(), and servers_set_value().

size_t ns_length_get (const chr_t * *s*)

Return the length of a null-terminated string.

Parameters:

s the input as a null-terminated string.

Returns:

the length in bytes of the string.

Definition at line 21 of file nuller.c.

Referenced by con_reverse_lookup(), con_write_ns(), config_fetch_host_number(), config_fetch_settings(), ecies_key_private_hex(), ecies_key_public_hex(), get_temp_file_handle(), host_platform(), host_version(), http_load_file(), ident_mt_mt(), imap_build_array(), imap_fetch_bodystructure(), imap_fetch_message(), imap_search_messages_date(), int16_conv_ns(), int32_conv_ns(), int64_conv_ns(), int8_conv_ns(), ip_presentation(), ip_str_addr(), ip_str_subnet(), lib_load_openssl(), line_pl_ns(), log_internal(), log_rotate(), log_start(), mail_mime_get_media_type(), mail_path_finder(), meta_folders_name(), mt_get_length(), ns_append(), ns_dupe(), ns_empty(), ns_empty_out(), ns_length_int(), portal_endpoint(), portal_endpoint_attachments_add(), portal_endpoint_error(), portal_endpoint_folders_add(), portal_endpoint_folders_rename(), portal_endpoint_response(), portal_parse_json_str_array(), rand_choices(), register_data_insert_user(), register_print_step1(), register_print_step2(), servers_network_start(), spool_path(),

st_merge_opts(), stmt_rebuild(), stmt_start(), tank_start(), teacher_add_cookie(), uint16_conv_ns(),
uint32_conv_ns(), uint64_conv_ns(), uint8_conv_ns(), virus_check(), xml_get_xpath_int16(),
xml_get_xpath_int32(), xml_get_xpath_int64(), xml_get_xpath_int8(), xml_get_xpath_st(),
xml_get_xpath_uint16(), xml_get_xpath_uint32(), xml_get_xpath_uint64(), and xml_get_xpath_uint8().

int ns_length_int (chr_t * s)

Return the length of a null-terminated string as an int, capped at INT_MAX.

Parameters:

s the null-terminated string to be evaluated.

Returns:

the length of the string.

Definition at line 54 of file nuller.c.

References log_pedantic, and ns_length_get().

Referenced by lib_load_bzip().

void ns_wipe (chr_t * s, size_t len)

Zero out a null-terminated string.

Parameters:

s the string to be wiped.

len the number of bytes to be wiped at the start of the string.

Returns:

This function returns no parameters.

Definition at line 209 of file nuller.c.

References log_pedantic, and mm_set().

Referenced by ns_alloc().

chr_t* pl_char_get (placer_t place)

shortcuts.c

shortcuts.c

Parameters:

place the input placer.

Returns:

NULL on failure or a character pointer to the block of data associated with the specified placer on success.

Definition at line 59 of file shortcuts.c.

References st_char_get().

Referenced by cache_config(), con_write_pl(), ecies_key_private(), ecies_key_public(),
imap_fetch_body_header(), imap_narrow_messages(), imap_search_messages_range(), ip_str_subnet(),
lib_load_openssl(), line_pl_pl(), molten_parse(), nvp_parse(), pl_get_embraced(),

pl_shrink_before_characters(), pl_skip_characters(), pl_skip_to_characters(), pl_starts_with_char(), pl_trim(), pl_trim_end(), pl_trim_start(), pop_process(), portal_message_body(), portal_upload(), relay_config(), servers_config(), smtp_auth_login(), smtp_auth_plain(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(), smtp_parse_rcpt_to(), smtp_process(), and str_tok_get_bl().

placer_t pl_clone (placer_t *place*)

Definition at line 35 of file shortcuts.c.

References pl_init().

void* pl_data_get (placer_t *place*)

Get a pointer to the data referenced by a placer.

Parameters:

place the input placer.

Returns:

NULL on failure or a pointer to the block of data associated with the specified placer on success.

Definition at line 49 of file shortcuts.c.

References st_data_get().

Referenced by ecies_key_private(), ecies_key_public(), imap_build_array(), imap_build_array_isliteral(), imap_fetch_body(), imap_fetch_body_header(), imap_folder_create(), imap_folder_rename(), imap_parse_address_part(), nvp_parse(), smtp_check_filters(), tok_get_pl(), and uint64_conv_pl().

bool_t pl_empty (placer_t *place*)

Determine whether or not the specified placer is empty.

Parameters:

place the input placer.

Returns:

true if the placer is empty or zero-length, or false otherwise.

Definition at line 89 of file shortcuts.c.

References st_empty().

Referenced by cache_config(), client_read_line(), con_read_line(), http_data_value_parse(), http_process(), imap_build_array(), imap_build_array_isliteral(), imap_fetch_body(), imap_fetch_body_header(), imap_narrow_messages(), imap_parse_address(), imap_parse_literal(), imap_process(), imap_search_messages_date(), imap_search_messages_header(), imap_search_messages_range(), mail_add_forward_headers(), mail_add_outbound_headers(), mail_mime_child(), mail_mime_count(), mail_mod_subject(), molten_parse(), multipart_get_boundary(), nvp_parse(), pl_get_embraced(), pl_inc(), pl_shrink_before_characters(), pl_skip_characters(), pl_skip_to_characters(), pl_starts_with_char(), pop_process(), portal_upload(), relay_config(), servers_config(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(), smtp_parse_rcpt_to(), and smtp_process().

bool_t pl_inc (placer_t * *place*, bool_t *more*)

Advance the placer one character forward beyond an expected character.

Parameters:

place the input placer.

more if true, the placer must contain more data, and vice versa.

Returns:

true if more was true and the placer contains more data, or if more was false and the placer ended; false otherwise.

Definition at line 119 of file shortcuts.c.

References pl_empty().

placer_t pl_init (void * *data*, size_t *len*)

Return a placer wrapping a data buffer of given size.

Parameters:

data a pointer to the data to be wrapped.

len the length, in bytes, of the data.

Returns:

a placer pointing to the specified data.

Definition at line 30 of file shortcuts.c.

References FOREIGNDATA, JOINTED, PLACER_T, placer_t, and STACK.

Referenced by bracket_extract_pl(), ecies_decrypt(), ecies_encrypt(), get_header_value_noopt(), http_parse_context(), http_parse_origin(), http_parse_pairs(), imap_build_array(), imap_fetch_body(), imap_fetch_body_portion(), imap_fetch_bodystructure(), imap_fetch_envelope(), imap_parse_address_breaker(), imap_search_messages_date(), imap_search_messages_header(), line_pl_bl(), mail_header_pop(), mail_mime_child(), mail_mime_header(), mail_mime_part(), mail_mime_update(), mail_store_header(), pl_clone(), pl_trim(), pl_trim_end(), pl_trim_start(), smtp_check_filters(), str_tok_get_bl(), str_tok_get_count_bl(), tok_get_ns(), and tok_pop().

size_t pl_length_get (placer_t *place*)

Get the length, in bytes, of a placer.

Parameters:

place the input placer.

Returns:

the size, in bytes, of the specified placer.

Definition at line 79 of file shortcuts.c.

References st_length_get().

Referenced by cache_config(), client_read(), client_read_line(), con_read(), con_read_line(), con_write_pl(), ecies_key_private(), ecies_key_public(), imap_build_array(), imap_build_array_isliteral(), imap_fetch_body(), imap_folder_create(), imap_folder_rename(), imap_parse_address_part(), imap_parse_literal(), ip_str_subnet(), line_pl_pl(), molten_parse(), nvp_parse(), pl_shrink_before_characters(), pl_trim(), pl_trim_end(), pl_trim_start(), pop_process(), portal_message_body(), portal_upload(), relay_config(), servers_config(), smtp_auth_login(), smtp_auth_plain(), smtp_check_filters(), and smtp_parse_helo_domain(),

smtp_parse_mail_from_path(), smtp_parse_rcpt_to(), smtp_process(), str_tok_get_bl(), str_tok_get_count_bl(), tok_get_pl(), and uint64_conv_pl().

int_t pl_length_int (placer_t *place*)

Get the length, in bytes, of a placer as an integer.

Parameters:

place the input placer.

Returns:

the size, in bytes, of the specified placer.

Definition at line 69 of file shortcuts.c.

References st_length_int().

Referenced by lib_load_openssl(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(), and smtp_parse_rcpt_to().

placer_t pl_null (void)

Return a zero-length placer pointing to NULL data.

Returns:

a zero-length placer pointing to NULL data.

Definition at line 19 of file shortcuts.c.

References FOREIGNDATA, JOINED, PLACER_T, placer_t, and STACK.

Referenced by bracket_extract_pl(), client_read(), client_read_line(), con_init_network_buffer(), con_read(), con_read_line(), get_header_opt(), imap_fetch_body(), imap_fetch_body_portion(), imap_fetch_envelope(), imap_narrow_messages(), imap_parse_address_breaker(), imap_parse_literal(), imap_search_messages_date(), imap_search_messages_header(), imap_search_messages_range(), imap_starttls(), line_pl_bl(), mail_header_pop(), mail_mime_child(), nvp_parse(), pl_trim(), pl_trim_end(), pl_trim_start(), pop_starttls(), portal_upload(), smtp_check_filters(), smtp_starttls(), str_tok_get_bl(), tok_get_ns(), and tok_pop().

placer_t pl_set (placer_t *place*, placer_t *set*)

Definition at line 39 of file shortcuts.c.

References placer_t.

Referenced by mail_mime_split().

bool_t pl_starts_with_char (placer_t *place*, chr_t *c*)

Determine if a placer begins with a specified character.

Parameters:

place the input placer.

c the character to be compared with the first byte of the placer's data.

Returns:

true if the placer begins with the given character or false otherwise.

Definition at line 100 of file shortcuts.c.

References `pl_char_get()`, and `pl_empty()`.

Referenced by `nvp_parse()`.

stringer_t* st_alloc_opts (uint32_t *opts*, size_t *len*)

Allocate a managed string with a specified options mask.

See also:

`st_valid_options()`

Note:

All requested allocation masks should conform to the validation imposed by `st_valid_opts()`. The supported types are: placer, nuller, block, managed, and mapped. The following logic is applied to requested string allocation options: 1. Any allocation options specified for strings to be allocated on the stack are IGNORED. 2. All jointed strings allocate memory for the header and data separately and then link them EXCEPT: Jointed placers only allocate space for a header. Jointed mapped strings allocate the data with an aligned `mmap()` operation. 3. Contiguous strings allocate space for the header and data together in a single contiguous block. 4. After allocation, the length field is set for block strings. 5. After allocation, the available field is set for managed strings. 6. Placers can only be jointed; mapped strings can only be contiguous.

Parameters:

opts the allocation options mask to be used by the newly allocated string.

len the length, in bytes, of the managed string to be allocated.

Returns:

NULL on failure or a pointer to the newly allocated managed string on success.

Definition at line 414 of file allocation.c.

References `align()`, `block_t`, `BLOCK_T`, `CONTIGUOUS`, `HEAP`, `JOINTED`, `log_pedantic`, `magma`, `MAGMA_SPOOL_DATA`, `managed_t`, `MANAGED_T`, `mapped_t`, `MAPPED_T`, `MEMORYBUF`, `mm_alloc()`, `mm_free()`, `mm_sec_alloc()`, `mm_sec_free()`, `mm_set()`, `nuller_t`, `NULLER_T`, `magma_t::page_length`, `placer_t`, `PLACER_T`, `SECURE`, `spool_mktemp()`, `st_info_opts()`, `st_valid_opts()`, and `STACK`.

Referenced by `base64_decode_opts()`, `base64_encode_opts()`, `credential_address()`, `credential_alloc_auth()`, `credential_alloc_mail()`, `dkim_create()`, `ecies_key_private_hex()`, `http_body()`, `mail_add_forward_headers()`, `mail_message_cleanup()`, `mail_mime_split()`, `meta_data_user_build_storage_keys()`, `qp_encode()`, `register_data_insert_user()`, `smtp_check_greylist()`, `smtp_data_read()`, `st_append_opts()`, `st_dupe_opts()`, `st_import()`, `st_merge_opts()`, `st_nullify()`, `st_vaprint_opts()`, and `url_encode()`.

stringer_t* st_append_opts (size_t *align*, stringer_t * *s*, stringer_t * *append*)

Append one managed string to another, with aligned memory boundaries.

Parameters:

align an alignment value to be used for managed string (re)allocation.

s the managed string to be extended, which will be allocated for the caller if passed as NULL.

append the managed string to be appended to *s*.

Returns:

NULL on failure, or a pointer to the appended result on success.

Definition at line 363 of file allocation.c.

References HEAP, JOINTED, log_pedantic, MANAGED_T, mm_copy(), st_alloc_opts(), st_avail_get(), st_data_get(), st_length_get(), st_length_set(), st_realloc(), and st_valid_append().

Referenced by http_body(), imap_id(), imap_search(), imap_status(), and mail_add_forward_headers().

stringer_t* st_aprint (chr_t * *format*, ...)

print.c

stringer_t stringer_t* st_aprint_opts (uint32_t *opts*, chr_t * *format*, ...)

size_t st_avail_get (stringer_t * *s*)

Return the total data buffer size allocated for a managed string.

Parameters:

s the input managed string.

Returns:

0 on failure, or the total buffer size in bytes on success.

Definition at line 151 of file length.c.

References log_pedantic, managed_t, MANAGED_T, mapped_t, MAPPED_T, MEMORYBUF, st_info_opts(), st_length_get(), and st_valid_opts().

Referenced by base64_decode(), base64_decode_mod(), base64_encode(), base64_encode_mod(), client_read(), client_read_line(), con_read(), con_read_line(), digest_hash(), file_load(), file_read(), hex_decode_st(), hex_encode_st(), host_platform(), host_version(), imap_parse_address_put(), ip_presentation(), ip_reversed(), ip_standard(), ip_subnet(), mail_add_required_headers(), rand_write(), sanity_check(), serialize_int16(), serialize_int32(), serialize_int64(), serialize_ns(), serialize_st(), serialize_uint16(), serialize_uint32(), serialize_uint64(), st_append_opts(), st_copy_in(), st_dupe_opts(), st_output(), st_trim(), st_vsprint(), st_wipe(), time_print_gmt(), and time_print_local().

size_t st_avail_set (stringer_t * *s*, size_t *avail*)

Set the total data buffer size allocated for a managed string.

Parameters:

s the input managed string.

avail the new buffer size for the managed string.

Returns:

0 on failure, or the managed string's new buffer size in bytes on success.

Definition at line 188 of file length.c.

References log_pedantic, managed_t, MANAGED_T, mapped_t, MAPPED_T, MEMORYBUF, st_info_opts(), and st_valid_avail().

chr_t* st_char_get (stringer_t * s)

data.c

data.c

See also:

st_data_get()

Parameters:

s the input managed string.

Returns:

NULL on failure or for an improperly constructed string; otherwise, a pointer to the string's data.

Definition at line 158 of file data.c.

References **st_data_get()**.

Referenced by **adjust_message_encryption()**, **cache_add()**, **cache_append()**, **cache_config()**, **cache_delete()**, **cache_get()**, **cache_get_u64()**, **cache_increment()**, **cache_output_settings()**, **cache_set()**, **cache_set_u64()**, **cache_set_value()**, **cache_silent_add()**, **cipher_name()**, **client_read()**, **client_read_line()**, **con_read()**, **con_read_line()**, **con_write_st()**, **config_load_cmdline_settings()**, **config_load_database_settings()**, **config_load_file_settings()**, **config_output_value()**, **config_output_value_generic()**, **config_value_set()**, **contact_abuse_checks()**, **contact_abuse_increment_history()**, **contact_business_valid_email()**, **contact_detail_delete()**, **contact_detail_upsert()**, **contact_insert()**, **contact_print_form()**, **contact_update()**, **credential_address()**, **credential_username()**, **decrypt_user_messages()**, **deserialize_int16()**, **deserialize_int32()**, **deserialize_int64()**, **deserialize_ns()**, **deserialize_st()**, **deserialize_uint16()**, **deserialize_uint32()**, **deserialize_uint64()**, **digest_name()**, **double_conv()**, **dspam_check()**, **dspam_train()**, **encrypt_user_messages()**, **float_conv()**, **folder_exists()**, **get_header_value_noopt()**, **get_temp_file_handle()**, **hex_encode_st_debug()**, **http_body()**, **http_content_load_directory()**, **http_data_header_parse()**, **http_data_value_decode()**, **http_data_value_parse()**, **http_load_file()**, **http_page_get()**, **http_parse_context()**, **http_parse_header()**, **http_parse_method()**, **http_parse_pairs()**, **http_print_301()**, **http_response_allow_cross()**, **http_response_cookie()**, **http_response_header()**, **http_response_options()**, **imap_append()**, **imap_append_message()**, **imap_build_array()**, **imap_capability()**, **imap_check()**, **imap_close()**, **imap_command_log_safe()**, **imap_copy()**, **imap_create()**, **imap_delete()**, **imap_examine()**, **imap_expunge()**, **imap_fetch()**, **imap_fetch_body()**, **imap_fetch_body_header()**, **imap_fetch_body_portion()**, **imap_fetch_bodystructure()**, **imap_fetch_envelope()**, **imap_fetch_parse_partial()**, **imap_fetch_return_header()**, **imap_folder_compare()**, **imap_id()**, **imap_idle()**, **imap_init()**, **imap_invalid()**, **imap_list()**, **imap_login()**, **imap_logout()**, **imap_lsub()**, **imap_narrow_messages()**, **imap_noop()**, **imap_parse_address_breaker()**, **imap_parse_address_part()**, **imap_parse_address_put()**, **imap_parse_literal()**, **imap_parse_qstring()**, **imap_process()**, **imap_rename()**, **imap_search()**, **imap_search_messages_date()**, **imap_search_messages_header()**, **imap_select()**, **imap_starttls()**, **imap_status()**, **imap_store()**, **imap_subscribe()**, **imap_unsubscribe()**, **ip_presentation()**, **ip_reversed()**, **ip_standard()**, **lib_load()**, **line_pl_st()**, **lock_get()**, **lock_release()**, **magma_folder_insert()**, **magma_folder_rename()**, **mail_add_forward_headers()**, **mail_add_outbound_headers()**, **mail_add_required_headers()**, **mail_build_signature()**, **mail_create_directory()**, **mail_db_insert_duplicate_message()**, **mail_db_insert_message()**, **mail_discover_encoding()**, **mail_discover_insertion_point()**, **mail_discover_type()**, **mail_extract_address()**, **mail_extract_tag()**, **mail_get_boundary()**, **mail_get_chunk()**, **mail_header_end()**, **mail_header_fetch_all()**, **mail_header_fetch_cleaned()**, **mail_header_pop()**, **mail_headers()**, **mail_insert_chunk_base64()**, **mail_insert_chunk_text()**, **mail_load_header()**, **mail_load_message()**, **mail_load_message_top()**, **mail_message()**, **mail_message_cleanup()**, **mail_message_path()**, **mail_mime_boundary()**, **mail_mime_child()**, **mail_mime_content_encoding()**, **mail_mime_content_id()**, **mail_mime_count()**, **mail_mime_encoding()**, **mail_mime_generate_boundary()**, **mail_mime_header()**, **mail_mime_part()**, **mail_mime_type()**, **mail_mime_type_group()**, **mail_mime_type_parameters_key()**, **mail_mime_type_parameters_value()**, **mail_mime_type_sub()**, **mail_mime_update()**, **mail_mod_subject()**, **mail_modify_part()**, **mail_setup_basic()**, **mail_signature_add()**, **meta_data_check_mailbox()**, **meta_data_delete_tag()**, **meta_data_fetch_mailbox_aliases()**, **meta_data_fetch_messages()**, **meta_data_insert_folder()**, **meta_data_insert_tag()**, **meta_data_update_folder_name()**, **meta_data_user_build()**,

meta_data_user_save_storage_keys(), mt_get_char(), pl_char_get(), pop_pass(), pop_retr(), pop_top(), portal_config_collection(), portal_config_entry(), portal_contact_details(), portal_endpoint(), portal_endpoint_alert_acknowledge(), portal_endpoint_alert_list(), portal_endpoint_aliases(), portal_endpoint_attachments_add(), portal_endpoint_attachments_remove(), portal_endpoint_auth(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(), portal_endpoint_contacts_list(), portal_endpoint_contacts_load(), portal_endpoint_contacts_move(), portal_endpoint_contacts_remove(), portal_endpoint_folders_add(), portal_endpoint_folders_list(), portal_endpoint_folders_remove(), portal_endpoint_folders_rename(), portal_endpoint_folders_tags(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(), portal_endpoint_messages_load(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(), portal_endpoint_messages_send(), portal_endpoint_messages_tag(), portal_endpoint_messages_tags(), portal_message_attachments(), portal_message_body(), portal_message_header(), portal_message_tags_array(), portal_meta(), portal_settings_changepass(), portal_settings_identity(), portal_upload(), portal_validate_request(), process_kill(), register_abuse_check_blocklist(), register_abuse_checks(), register_abuse_increment_history(), register_business_step1(), register_business_validate_password(), register_business_validate_username(), register_captcha_generate(), register_data_check_username(), register_data_insert_user(), register_print_captcha(), register_session_cache(), register_session_get(), relay_config(), relay_output_settings(), relay_set_value(), serial_get(), serial_increment(), serial_reset(), serialize_int16(), serialize_int32(), serialize_int64(), serialize_ns(), serialize_st(), serialize_uint16(), serialize_uint32(), serialize_uint64(), servers_config(), servers_output_settings(), servers_set_value(), sess_get(), signal_shutdown(), smtp_add_bypass_entry(), smtp_bounce(), smtp_check_authorized_from(), smtp_check_filters(), smtp_check_greylist(), smtp_check_rbl(), smtp_check_receive_quota(), smtp_client_connect(), smtp_client_send_data(), smtp_client_send_helo(), smtp_client_send_mailfrom(), smtp_client_send_nullfrom(), smtp_client_send_rcptto(), smtp_data_finish(), smtp_data_outbound(), smtp_data_read(), smtp_ehlo(), smtp_fetch_authorization(), smtp_fetch_inbound(), smtp_helo(), smtp_init(), smtp_insert_spamsig(), smtp_rcpt_to(), smtp_reply(), smtp_rollout(), smtp_update_receive_stats(), spf_check(), spool_check(), spool_cleanup(), spool_mktemp(), spool_start(), spool_stop(), sql_query_conn(), ssl_start(), st_info_opts(), st_merge_opts(), st_replace(), st_trim(), teacher_add_cookie(), teacher_print_message(), teacher_process(), time_print_gmt(), time_print_local(), tok_pop_init_st(), user_config_delete(), user_config_upsert(), virus_engine_create(), and virus_start().

void st_cleanup (stringer_t * s)

A checked managed string free front-end function.

See also:

st_free()

Parameters:

s the managed string to be freed.

Returns:

This function returns no value.

Definition at line 98 of file allocation.c.

References st_free().

Referenced by client_close(), con_destroy(), credential_alloc_auth(), credential_free(), dkim_create(), http_data_free(), http_free_content(), http_print_301(), http_response_header(), http_response_options(), http_session_reset(), imap_command_parser(), imap_fetch_body(), imap_fetch_body_header(), imap_fetch_body_mime(), imap_fetch_body_tag(), imap_fetch_bodystructure(), imap_fetch_envelope(), imap_fetch_response_free(), imap_folder_create(), imap_folder_rename(), imap_login(), imap_session_destroy(), magma_folder_find_full_name(), mail_add_forward_headers(), mail_add_outbound_headers(), mail_add_required_headers(), mail_build_signature(), mail_cache_destroy(), mail_cache_reset(), mail_cache_set(), mail_destroy(), mail_destroy_header(), mail_destroy_message(), mail_get_boundary(), mail_insert_chunk_base64(), mail_mime_boundary(), mail_mime_free(),

mail_mime_get_smtp_envelope(), message_free(), meta_data_fetch_user(), meta_data_user_build_storage_keys(), meta_get(), meta_user_build(), meta_user_destroy(), pop_session_destroy(), pop_user(), portal_endpoint_contacts_copy(), portal_endpoint_messages_list(), portal_folder_contacts_add(), portal_folder_mail_add(), portal_folder_mail_remove(), portal_message_attachments(), portal_message_body(), portal_message_header(), portal_smtp_merge_headers(), register_business_step2(), register_data_insert_user(), register_session_cache(), register_session_free(), sess_destroy(), sess_get(), sess_release_attachment(), sess_token(), smtp_auth_login(), smtp_auth_plain(), smtp_bounce(), smtp_check_filters(), smtp_check_greylist(), smtp_ehlo(), smtp_free_inbound(), smtp_free_outbound(), smtp_free_recipients(), smtp_helo(), smtp_list_free_filter(), smtp_reply(), smtp_session_destroy(), smtp_session_reset(), st_info_opts(), st_merge_opts(), teacher_data_delete(), teacher_data_free(), and teacher_process().

stringer_t* st_copy_in (stringer_t * s, void * buf, size_t len)

Copy data into a managed string.

Parameters:

s the managed string to store the copied contents of the data.
buf a pointer to the buffer containing the data to be copied.
len the length of the data to be copied.

Returns:

NULL on failure, or a pointer to the managed string on success.

Definition at line 235 of file allocation.c.

References log_pedantic, mm_copy(), st_avail_get(), st_data_get(), and st_length_set().

Referenced by credential_alloc_auth(), ecies_key_private_hex(), and meta_data_user_build_storage_keys().

void* st_data_get (stringer_t * s)

Retrieve the data associated with a managed string.

Parameters:

s the input managed string.

Returns:

NULL on failure or for an improperly constructed string; otherwise, a pointer to the string's data.

Definition at line 110 of file data.c.

References block_t, BLOCK_T, constant_t, CONSTANT_T, debug_hook(), log_pedantic, managed_t, MANAGED_T, mapped_t, MAPPED_T, MEMORYBUF, nuller_t, NULLER_T, placer_t, PLACER_T, st_info_opts(), and st_valid_opts().

Referenced by adjust_message_encryption(), alert_alloc(), alias_alloc(), base64_decode(), base64_decode_mod(), base64_encode(), base64_encode_mod(), client_read(), client_read_line(), compress_bzip(), compress_import(), compress_lzo(), compress_zlib(), con_read(), con_read_line(), contact_alloc(), contact_detail_alloc(), contact_name(), credential_alloc_auth(), credential_alloc_mail(), decompress_block_lzo(), decompress_bzip(), decompress_lzo(), decompress_zlib(), digest_hash(), dkim_check(), dkim_create(), domain_alloc(), file_load(), file_read(), hex_decode_st(), hex_encode_st(), http_body(), http_parse_origin(), imap_command_parser(), imap_fetch_bodystructure(), int16_conv_st(), int32_conv_st(), int64_conv_st(), int8_conv_st(), magma_folder_alloc(), mail_add_forward_headers(), mail_add_outbound_headers(), mail_discover_insertion_point(), mail_extract_address(), mail_get_boundary(), mail_insert_chunk_base64(), mail_load_message(), mail_mime_header(), mail_mime_part(),

mail_mod_subject(), mail_modify_part(), message_alloc(), meta_data_user_build_storage_keys(), meta_folder_stats_tag_alloc(), pl_data_get(), pop_num_parse(), pop_pass_parse(), pop_top_parse(), pop_user_parse(), qp_decode(), qp_encode(), rand_choices(), rand_write(), register_data_insert_user(), sanity_check(), scramble_encrypt(), scramble_import(), sess_get(), smtp_check_greylist(), smtp_data_finish(), smtp_data_read(), sql_query_conn(), st_append_opts(), st_char_get(), st_copy_in(), st_dupe_opts(), st_empty(), st_empty_out(), st_import(), st_merge_opts(), st_nullify(), st_replace(), st_uchar_get(), st_vaprint_opts(), st_vsprint(), st_wipe(), symmetric_decrypt(), symmetric_encrypt(), symmetric_vector(), tank_store(), tok_get_count_st(), tok_get_st(), uint16_conv_st(), uint32_conv_st(), uint64_conv_st(), uint8_conv_st(), url_decode(), url_encode(), user_config_entry_alloc(), virus_check(), xml_encode(), zbase32_decode(), and zbase32_encode().

void st_data_set (stringer_t * s, void * data)

Set the underlying data of a jointed managed string.

Parameters:

s the managed string to be adjusted.
data the data buffer to be attached to the input managed string.

Note:

The underlying data of *s* will be released, unless it contains foreign data.

Returns:

This function does not return a value.

Definition at line 53 of file data.c.

References block_t, BLOCK_T, FOREIGNDATA, JOINTED, log_pedantic, managed_t, MANAGED_T, MAPPED_T, MEMORYBUF, mm_free(), mm_sec_free(), mm_sec_secured(), nuller_t, NULLER_T, placer_t, PLACER_T, SECURE, st_info_opts(), and st_valid_jointed().

Referenced by contact_name(), credential_alloc_mail(), smtp_data_finish(), smtp_data_read(), and st_dupe_opts().

stringer_t* st_dupe (stringer_t * s)

Duplicate a managed string.

See also:

st_dupe_opts()

Note:

The allocation options of the duplicated string will be the same as that of the source string.

Parameters:

s the managed string to be duplicated.

Returns:

NULL on failure, or a copy of the input managed string on success.

Definition at line 349 of file allocation.c.

References st_dupe_opts().

Referenced by ar_dupe(), contact_folder_rename(), get_temp_file_handle(), http_load_file(), http_print_301(), imap_folder_create(), imap_folder_rename(), mail_add_forward_headers(), mail_add_outbound_headers(), mail_cache_get(), meta_data_user_build(), meta_user_build(), register_business_step1(), register_business_step2(), register_print_message(), register_print_step1(), and register_print_step2().

register_print_step3(), register_session_get(), smtp_add_recipient(), smtp_data_outbound(),
smtp_fetch_inbound(), smtp_forward_message(), and teacher_process().

stringer_t* st_dupe_opts (uint32_t opts, stringer_t * s)

Create a duplicate copy of a managed string with a specified set of allocation options.

See also:

st_valid_opts()

Note:

Both the source and destination managed strings must have option values that pass **st_valid_opts()**. The following procedure occurs when creating the duplicate managed string: If the destination is a placer, 0 bytes are allocated for the duplicate's underlying data. If the destination is a constant, nuller, or block, its underlying data buffer will be of equal size to the source's data length. If the destination is managed or mapped, and so is the source, its underlying data buffer will be of equal size to the source's available size; but if the source is neither, the underlying data buffer of the destination managed or mapped string will equal the size of the source's data length. A deep copy will be made of the source data to the destination if the destination is NOT a placer.

Parameters:

opts the allocation options mask to be used for the newly created managed string.
s the target managed string to be duplicated.

Returns:

NULL on failure or a pointer to a copy of the duplicated managed string on success.

Definition at line 275 of file allocation.c.

References BLOCK_T, CONSTANT_T, log_pedantic, MANAGED_T, MAPPED_T, MEMORYBUF, mm_copy(), NULLER_T, PLACER_T, st_alloc_opts(), st_avail_get(), st_data_get(), st_data_set(), st_info_opts(), st_length_get(), st_length_set(), st_valid_opts(), and st_valid_tracked().

Referenced by args_parse(), cache_set_value(), config_value_set(), get_temp_file_handle(), http_data_value_parse(), http_load_file(), http_parse_header(), http_parse_method(), http_response_connection(), imap_fetch_response_add(), imap_folder_create(), imap_folder_rename(), imap_login(), magma_folder_name(), mail_add_forward_headers(), mail_cache_set(), mt_dupe(), portal_message_body(), relay_set_value(), servers_set_value(), sess_create(), smtp_fetch_inbound(), smtp_relay_message(), and st_dupe().

bool_t st_empty (stringer_t * s)

Determine whether the specified managed string is empty or not.

Parameters:

s the input managed string.

Returns:

true if string is NULL or uninitialized or empty; false otherwise.

Definition at line 20 of file data.c.

References st_data_get(), and st_length_get().

Referenced by cache_free(), cache_output_settings(), cache_set_value(), cache_validate(), cipher_name(), client_read_line(), compress_import(), con_read_line(), config_load_cmdline_settings(), config_load_database_settings(), config_load_file_settings(), and config_output_value(),

config_output_value_generic(), config_value_set(), contact_create(), contact_edit(), contact_find_detail(),
 contact_find_name(), contact_folder_create(), contact_folder_rename(), contact_update(),
 credential_alloc_auth(), digest_hash(), digest_name(), double_conv(), float_conv(), lock_get(), lock_release(),
 magma_folder_find_full_name(), magma_folder_find_name(), magma_folder_name(),
 mail_add_outbound_headers(), mail_add_required_headers(), mail_header_fetch_all(),
 mail_header_fetch_cleaned(), mail_header_pop(), mail_insert_chunk_text(), mail_mime_child(),
 mail_mime_count(), mail_mime_part(), mail_mime_split(), message_folder_create(),
 meta_data_check_mailbox(), meta_data_user_build(), meta_data_user_build_storage_keys(), meta_get(),
 meta_remove(), meta_user_build(), meta_user_prune(), mt_is_empty(), pl_empty(), pop_pass(), relay_free(),
 relay_output_settings(), relay_set_value(), relay_validate(), scramble_import(), servers_free(),
 servers_network_start(), servers_output_settings(), servers_set_value(), servers_validate(), smtp_auth_login(),
 smtp_auth_plain(), st_info_opts(), st_merge_opts(), and user_config_edit().

bool_t st_empty_out (stringer_t * s, uchr_t ** ptr, size_t * len)

void st_free (stringer_t * s)

Free a managed string.

See also:

st_valid_free(), st_valid_opts()

Note:

Any managed string to be freed should conform with the validity checks enforced by **st_valid_free()/st_valid_opts()** *NO* managed string should be passed to **st_free()** if it was allocated on the stack, or contains foreign data. The following logic is carried out depending on the allocation options of the string to be freed: Jointed placer: Free header, if secure or on heap Free underlying data if it is not foreign Jointed nuller: Free header and underlying data Jointed block: Free header and underlying data Jointed managed: Free header and underlying data Jointed mapped: Free the header and (munmap) underlying data Contiguous nuller: Free header (this includes the underlying data because they are already merged) Contiguous block: Free header (this includes the underlying data because they are already merged) Contiguous managed: Free header (this includes the underlying data because they are already merged)

Parameters:

s the managed string to be freed.

Returns:

This function returns no value.

HIGH: Finish this logic out. Stack allocations are skipped, unless they are jointed. In that case the data is freed unless it carries a foreigner flag. Note its possible for a jointed string to have a NULL data reference. We should be picking up on that too. Contiguous heap buffers are just destroyed.

Do we need to differentiate between stack structures, and heap data blocks needing to be freed, or not?

Definition at line 34 of file allocation.c.

References block_t, BLOCK_T, CONTIGUOUS, data, debug_hook(), FOREIGNDATA, HEAP, JOINTED, length, log_pedantic, managed_t, MANAGED_T, mapped_t, MAPPED_T, MEMORYBUF, mm_free(), mm_sec_free(), nuller_t, NULLER_T, placer_t, PLACER_T, SECURE, st_info_opts(), and st_valid_free().

Referenced by ar_free(), base64_decode_opts(), base64_encode_opts(), cache_free(), cache_set_value(), client_print(), compress_bzip(), compress_lzo(), compress_zlib(), config_free(), config_load_cmdline_settings(), config_load_file_settings(), config_value_set(), contact_business(), contact_folder_rename(), contact_free(), contact_print_form(), and contact_print_message(),

credential_alloc_auth(), decompress_block_lzo(), decompress_bzip(), decompress_lzo(), decompress_zlib(), digest_hash(), dspam_check(), dspam_train(), ecies_key_private_hex(), get_temp_file_handle(), http_content_load_directory(), http_content_load_fonts(), http_content_refresh(), http_content_start(), http_data_header_parse_line(), http_data_value_parse(), http_load_file(), http_print_301(), imap_build_array(), imap_command_parser(), imap_fetch_body(), imap_fetch_body_tag(), imap_fetch_bodystructure(), imap_fetch_response_add(), imap_folder_create(), imap_folder_name_escaped(), imap_folder_rename(), imap_id(), imap_list(), imap_lsub(), imap_narrow_folders(), imap_parse_address(), imap_parse_address_part(), imap_parse_literal(), imap_range_build(), imap_search(), imap_search_messages_body(), imap_search_messages_date(), imap_search_messages_header(), imap_search_messages_text(), imap_status(), ip_presentation(), ip_standard(), ip_subnet(), mail_add_forward_headers(), mail_add_outbound_headers(), mail_add_required_headers(), mail_build_signature(), mail_create_message(), mail_discover_encoding(), mail_discover_insertion_point(), mail_discover_type(), mail_extract_tag(), mail_header_fetch_all(), mail_insert_chunk_base64(), mail_insert_chunk_text(), mail_load_header(), mail_load_message(), mail_message_cleanup(), mail_mime_content_encoding(), mail_mime_content_id(), mail_mime_encode_part(), mail_mime_encoding(), mail_mime_generate_boundary(), mail_mime_get_smtp_envelope(), mail_mime_split(), mail_mime_type(), mail_mime_type_group(), mail_mime_type_parameters(), mail_mime_type_sub(), mail_mod_subject(), mail_modify_part(), meta_data_fetch_all_tags(), meta_data_user_build(), meta_data_user_build_storage_keys(), meta_data_user_save_storage_keys(), meta_folders_by_name(), meta_folders_name(), meta_get(), mt_free(), nvp_alloc(), nvp_parse(), pop_pass(), pop_user(), portal_endpoint_attachments_progress(), portal_endpoint_folders_add(), portal_endpoint_folders_rename(), portal_endpoint_messages_list(), portal_endpoint_messages_send(), portal_endpoint_search(), portal_parse_json_str_array(), portal_print_login(), portal_smtp_create_data(), protocol_secure(), rand_choices(), register_data_fetch_blocklist(), register_data_insert_user(), register_print_captcha(), register_print_message(), register_print_step1(), register_print_step2(), register_print_step3(), register_session_cache(), register_session_get(), relay_free(), relay_set_value(), sanity_check(), scramble_decrypt(), scramble_encrypt(), serial_get(), serial_increment(), serial_reset(), servers_free(), servers_set_value(), sess_get(), smtp_accept_message(), smtp_auth_login(), smtp_auth_plain(), smtp_bounce(), smtp_check_receive_quota(), smtp_data(), smtp_data_outbound(), smtp_data_read(), smtp_forward_message(), smtp_mail_from(), smtp_rcpt_to(), smtp_reply(), smtp_rollout(), smtp_update_receive_stats(), spool_cleanup(), spool_mktemp(), spool_start(), spool_stop(), ssl_start(), st_cleanup(), st_replace(), st_vaprint_opts(), stamp_counter_check(), statistics_process(), symmetric_decrypt(), symmetric_encrypt(), teacher_add_cookie(), teacher_data_get(), teacher_data_save(), teacher_print_form(), teacher_print_message(), virus_stop(), warehouse_fetch_patterns(), and zbbase32_decode().

stringer_t* st_import (const void * s, size_t len)

Create a new (contiguous managed) managed string on the heap to hold a copy of the specified data buffer.

Parameters:

s the address of the buffer to be duplicated.
len the length, in bytes, of the copied buffer.

Returns:

NULL on failure, or a pointer to the newly allocated managed string on success.

Definition at line 214 of file allocation.c.

References CONTIGUOUS, HEAP, MANAGED_T, mm_copy(), st_alloc_opts(), st_data_get(), and st_length_set().

Referenced by cache_get(), con_reverse_lookup(), dspam_check(), ecies_key_public_hex(), get_temp_file_handle(), http_data_header_parse_line(), http_load_file(), imap_fetch_body(), imap_fetch_body_tag(), imap_fetch_bodystructure(), imap_fetch_message(), imap_fetch_return_header(), imap_parse_astring(), imap_parse_nstring(), imap_search_messages_date(), mail_add_outbound_headers(), mail_add_required_headers(), mail_header_fetch_all(), mail_load_header(), mail_load_message(),

mail_mime_content_encoding(), mail_mime_content_id(), mail_mime_type_group(),
 mail_mime_type_parameters_key(), mail_mime_type_parameters_value(), mail_mime_type_sub(),
 meta_data_user_build_storage_keys(), meta_data_user_save_storage_keys(), meta_folders_name(),
 nvp_parse(), pop_pass_parse(), pop_user_parse(), portal_endpoint_attachments_add(),
 portal_endpoint_folders_add(), portal_endpoint_folders_rename(), portal_endpoint_messages_list(),
 portal_parse_json_str_array(), portal_upload(), register_captcha_generate(), res_field_string(),
 servers_network_start(), smtp_parse_auth(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(),
 smtp_parse_rcpt_to(), tank_load(), teacher_add_cookie(), xml_dump_doc(), xml_get_xpath_st(), and
 xml_node_get_content_st().

const chr_t* st_info_allocator (uint32_t opts)

Get a readable description of a managed string's allocator type.

Parameters:

opts a value containing the managed string's option mask.

Returns:

a pointer to a null-terminated string containing a description of the managed string's allocator type.

Definition at line 101 of file info.c.

References HEAP, SECURE, st_option_allocators, and STACK.

Referenced by st_info_opts().

const chr_t* st_info_layout (uint32_t opts)

Get a readable description of a managed string's data layout.

Parameters:

opts a value containing the managed string's option mask.

Returns:

a pointer to a null-terminated string containing a description of the managed string's data layout.

Definition at line 80 of file info.c.

References CONTIGUOUS, JOINTED, and st_option_layouts.

Referenced by st_info_opts().

chr_t* st_info_opts (uint32_t opts, chr_t * s, size_t len)

Get a readable description of all of a managed string's option flags.

Parameters:

opts a value containing the managed string's option mask.

s a pointer to a null-terminated string to receive the options description.

len the length, in bytes, of the description output buffer.

Returns:

NULL on failure, or a pointer to the output buffer containing the managed string's options description on success.

LOW: Turn this into a loop.

Definition at line 127 of file info.c.

References FOREIGNDATA, NULLER, st_append, st_char_get(), st_cleanup(), st_empty(), st_info_allocator(), st_info_layout(), st_info_type(), st_length_int(), and st_option_flags.

Referenced by st_alloc_opts(), st_avail_get(), st_avail_set(), st_data_get(), st_data_set(), st_dupe_opts(), st_free(), st_length_get(), st_length_set(), st_merge_opts(), st_opt_get(), st_opt_set(), st_realloc(), st_vaprint_opts(), st_vsprint(), and st_wipe().

const chr_t* st_info_type (uint32_t opts)

Get a readable description of a managed string's data type.

Parameters:

opts a value containing the managed string's option mask.

Returns:

a pointer to a null-terminated string containing a description of the managed string's data type.

Definition at line 47 of file info.c.

References BLOCK_T, CONSTANT_T, MANAGED_T, MAPPED_T, NULLER_T, PLACER_T, and st_option_types.

Referenced by st_info_opts().

size_t st_length_get (stringer_t * s)

Return the length of the data in a managed string.

Parameters:

s the input managed string.

Returns:

0 on failure, or the length of the managed string's data in bytes.

Definition at line 20 of file length.c.

References block_t, BLOCK_T, constant_t, CONSTANT_T, data, log_pedantic, managed_t, MANAGED_T, mapped_t, MAPPED_T, MEMORYBUF, nuller_t, NULLER_T, placer_t, PLACER_T, st_info_opts(), and st_valid_opts().

Referenced by adjust_message_encryption(), alert_alloc(), alias_alloc(), base64_decode(), base64_decode_mod(), base64_encode(), base64_encode_mod(), cache_add(), cache_append(), cache_config(), cache_delete(), cache_get(), cache_get_u64(), cache_increment(), cache_set(), cache_set_u64(), cache_set_value(), cache_silent_add(), client_read(), client_read_line(), compress_bzip(), compress_import(), compress_lzo(), compress_zlib(), con_read(), con_read_line(), con_write_st(), config_value_set(), contact_alloc(), contact_business_valid_email(), contact_detail_alloc(), contact_detail_delete(), contact_detail_upsert(), contact_folder_create(), contact_folder_rename(), contact_insert(), contact_print_form(), contact_print_message(), contact_update(), credential_address(), credential_alloc_auth(), credential_alloc_mail(), credential_username(), decompress_block_lzo(), deserialize_int16(), deserialize_int32(), deserialize_int64(), deserialize_ns(), deserialize_st(), deserialize_uint16(), deserialize_uint32(), deserialize_uint64(), digest_hash(), dkim_check(), dkim_create(), domain_alloc(), double_conv(), dspam_train(), float_conv(), get_header_value_noopt(), hex_decode_st(), hex_encode_st(), hex_encode_st_debug(), http_body(), http_data_header_parse(), http_data_value_decode(), http_load_file(),

http_page_get(), http_parse_context(), http_parse_header(), http_parse_pairs(), http_response(),
 imap_append_message(), imap_build_array(), imap_command_parser(), imap_copy(), imap_fetch_body(),
 imap_fetch_body_header(), imap_fetch_bodystructure(), imap_fetch_envelope(), imap_fetch_message(),
 imap_fetch_parse_partial(), imap_folder_compare(), imap_id(), imap_init(), imap_narrow_folders(),
 imap_parse_address_breaker(), imap_parse_address_part(), imap_parse_address_put(), imap_process(),
 imap_search(), imap_search_messages_date(), imap_search_messages_header(), imap_starttls(), imap_status(),
 imap_valid_folder_name(), int16_conv_st(), int32_conv_st(), int64_conv_st(), int8_conv_st(), ip_subnet(),
 line_pl_st(), magma_folder_alloc(), magma_folder_insert(), magma_folder_rename(),
 mail_add_forward_headers(), mail_add_outbound_headers(), mail_add_required_headers(),
 mail_build_signature(), mail_db_insert_duplicate_message(), mail_db_insert_message(),
 mail_discover_encoding(), mail_discover_insertion_point(), mail_discover_type(), mail_get_boundary(),
 mail_get_chunk(), mail_header_end(), mail_header_fetch_all(), mail_header_fetch_cleaned(),
 mail_header_pop(), mail_headers(), mail_insert_chunk_base64(), mail_insert_chunk_text(),
 mail_load_header(), mail_load_message_top(), mail_message(), mail_message_cleanup(),
 mail_mime_boundary(), mail_mime_child(), mail_mime_content_encoding(), mail_mime_content_id(),
 mail_mime_count(), mail_mime_encoding(), mail_mime_header(), mail_mime_part(), mail_mime_type(),
 mail_mime_type_group(), mail_mime_type_parameters_key(), mail_mime_type_parameters_value(),
 mail_mime_type_sub(), mail_mime_update(), mail_mod_subject(), mail_modify_part(), mail_signature_add(),
 mail_store_message(), message_alloc(), meta_data_check_mailbox(), meta_data_delete_tag(),
 meta_data_insert_folder(), meta_data_insert_tag(), meta_data_update_folder_name(), meta_data_user_build(),
 meta_data_user_save_storage_keys(), meta_folder_stats_tag_alloc(), mt_get_length(), nvp_parse(),
 pl_length_get(), pop_num_parse(), pop_pass_parse(), pop_retr(), pop_top(), pop_top_parse(), pop_user_parse(),
 portal_endpoint_alert_acknowledge(), portal_endpoint_attachments_add(),
 portal_endpoint_attachments_progress(), portal_endpoint_attachments_remove(),
 portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(),
 portal_endpoint_contacts_list(), portal_endpoint_contacts_load(), portal_endpoint_contacts_move(),
 portal_endpoint_contacts_remove(), portal_endpoint_folders_add(), portal_endpoint_folders_list(),
 portal_endpoint_folders_remove(), portal_endpoint_folders_rename(), portal_endpoint_folders_tags(),
 portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(),
 portal_endpoint_messages_load(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(),
 portal_endpoint_messages_send(), portal_endpoint_messages_tag(), portal_endpoint_search(),
 portal_message_attachments(), portal_print_login(), portal_settings_changepass(), portal_upload(),
 portal_validate_request(), qp_decode(), rand_write(), register_abuse_check_blocklist(),
 register_business_validate_password(), register_business_validate_username(), register_captcha_generate(),
 register_data_check_username(), register_data_insert_user(), register_print_captcha(),
 register_print_message(), register_print_step1(), register_print_step2(), register_print_step3(), relay_config(),
 relay_set_value(), scramble_decrypt(), scramble_encrypt(), scramble_import(), serialize_int16(),
 serialize_int32(), serialize_int64(), serialize_ns(), serialize_st(), serialize_uint16(), serialize_uint32(),
 serialize_uint64(), servers_config(), servers_set_value(), smtp_accept_message(), smtp_auth_plain(),
 smtp_check_authorized_from(), smtp_check_filters(), smtp_check_greylist(), smtp_check_receive_quota(),
 smtp_client_connect(), smtp_client_send_data(), smtp_client_send_helo(), smtp_client_send_mailfrom(),
 smtp_client_send_rcptto(), smtp_data_outbound(), smtp_fetch_authorization(), smtp_fetch_inbound(),
 smtp_insert_spamsig(), smtp_update_receive_stats(), sql_query_conn(), st_append_opts(), st_avail_get(),
 st_dupe_opts(), st_empty(), st_empty_out(), st_length_int(), st_merge_opts(), st_realloc(), st_trim(),
 statistics_process(), symmetric_key(), tank_store(), teacher_print_form(), teacher_print_message(),
 teacher_process(), tok_get_count_st(), tok_get_st(), tok_pop_init_st(), uint16_conv_st(), uint32_conv_st(),
 uint64_conv_st(), uint8_conv_st(), url_decode(), user_config_delete(), user_config_entry_alloc(),
 user_config_upsert(), and virus_check().

int_t st_length_int (stringer_t * s)

Return the length of a managed string as an integer.

Parameters:

s the managed string as type stringer_t *.

Returns:

the string length as an integer, capped at INT_MAX.

Definition at line 75 of file length.c.

References log_pedantic, st_length_get(), and st_valid_opts().

Referenced by cache_add(), cache_append(), cache_config(), cache_delete(), cache_get(), cache_get_u64(), cache_increment(), cache_output_settings(), cache_set(), cache_set_u64(), cipher_name(), config_load_cmdline_settings(), config_load_database_settings(), config_load_file_settings(), config_output_value(), config_output_value_generic(), contact_abuse_checks(), contact_abuse_increment_history(), contact_detail_delete(), contact_detail_upsert(), digest_name(), double_conv(), float_conv(), http_body(), http_data_value_parse(), http_load_file(), http_parse_header(), http_parse_method(), http_print_301(), http_response_allow_cross(), http_response_cookie(), http_response_header(), http_response_options(), imap_append(), imap_append_message(), imap_capability(), imap_check(), imap_close(), imap_command_log_safe(), imap_copy(), imap_create(), imap_delete(), imap_examine(), imap_expunge(), imap_fetch(), imap_id(), imap_idle(), imap_init(), imap_invalid(), imap_list(), imap_login(), imap_logout(), imap_lsub(), imap_narrow_messages(), imap_noop(), imap_process(), imap_rename(), imap_search(), imap_select(), imap_status(), imap_store(), imap_subscribe(), imap_unsubscribe(), lock_get(), lock_release(), mail_create_directory(), mail_message_path(), mail_signature_add(), mail_store_message(), meta_data_fetch_mailbox_aliases(), meta_data_user_build(), pl_length_int(), pop_pass(), portal_config_entry(), process_kill(), register_business_step1(), register_session_cache(), register_session_get(), relay_config(), relay_output_settings(), serial_get(), serial_increment(), serial_reset(), servers_config(), servers_output_settings(), servers_set_value(), smtp_bounce(), smtp_check_filters(), smtp_check_greylist(), smtp_check_rbl(), smtp_client_connect(), smtp_client_send_data(), smtp_client_send_mailfrom(), smtp_client_send_nullfrom(), smtp_client_send_rcptto(), smtp_ehlo(), smtp_fetch_authorization(), smtp_fetch_inbound(), smtp_helo(), smtp_init(), smtp_rcpt_to(), smtp_reply(), spf_check(), spool_check(), spool_cleanup(), spool_mktemp(), spool_start(), spool_stop(), sql_query_conn(), st_info_opts(), user_config_delete(), user_config_upsert(), and virus_start().

size_t st_length_set (stringer_t * s, size_t len)

Set the data length for a managed string that supports data length tracking.

Parameters:

s the input managed string.

len the new value of the managed string's data length.

Returns:

0 on error, or the new data length on success.

Definition at line 104 of file length.c.

References log_pedantic, managed_t, MANAGED_T, mapped_t, MAPPED_T, MEMORYBUF, placer_t, PLACER_T, st_info_opts(), and st_valid_tracked().

Referenced by base64_decode(), base64_decode_mod(), base64_encode(), base64_encode_mod(), client_read(), client_read_line(), con_read(), con_read_line(), contact_name(), credential_address(), credential_alloc_mail(), credential_username(), decompress_block_lzo(), decompress_bzip(), decompress_lzo(), decompress_zlib(), deserialize_st(), digest_hash(), file_load(), file_read(), hex_decode_st(), hex_encode_st(), hex_encode_st_debug(), http_data_value_decode(), http_load_file(), http_parse_header(), imap_build_array(), imap_parse_address_part(), imap_parse_address_put(), imap_parse_literal(), imap_parse_qstring(), imap_search(), imap_starttls(), imap_valid_folder_name(), ip_presentation(), ip_reversed(), ip_standard(), mail_add_required_headers(), mail_extract_address(), mail_extract_tag(), mail_header_fetch_cleaned(), mail_load_header(), mail_load_message(), mail_load_message_top(), mail_message_cleanup(), mail_mime_generate_boundary(), pop_starttls(), qp_decode(), rand_choices(), rand_write(), serialize_int16(), serialize_int32(), serialize_int64(), serialize_ns(), serialize_st(),

serialize_uint16(), serialize_uint32(), serialize_uint64(), smtp_data_finish(), smtp_data_read(), smtp_starttls(), st_append_opts(), st_copy_in(), st_dupe_opts(), st_import(), st_merge_opts(), st_nullify(), st_replace(), st_trim(), st_vaprint_opts(), st_vsprintf(), st_wipe(), symmetric_decrypt(), symmetric_encrypt(), symmetric_key(), symmetric_vector(), time_print_gmt(), time_print_local(), url_decode(), url_encode(), zbase32_decode(), and zbase32_encode().

stringer_t* st_merge_opts (uint32_t *opts*, chr_t * *format*, ...)

Concatenate a variable number of user-supplied multi-type strings.

Parameters:

opts the options value of the resulting managed string that will be allocated for the caller.

format a format string consisting of the characters 's' for specifying that the next variable argument will be a managed string or 'n' if it is a null-terminated string.

... a variable argument list containing the strings to be concatenated to one another.

Returns:

a newly allocated string containing the concatenations of all specified managed and null-terminated strings.

Definition at line 115 of file allocation.c.

References length, log_pedantic, MEMORYBUF, mm_copy(), ns_empty(), ns_length_get(), st_alloc_opts(), st_char_get(), st_cleanup(), st_data_get(), st_empty(), st_info_opts(), st_length_get(), st_length_set(), st_valid_destination(), and st_valid_tracked().

Referenced by credential_alloc_auth(), mail_add_forward_headers(), mail_add_inbound_headers(), mail_add_outbound_headers(), mail_add_required_headers(), and spool_path().

stringer_t* st_nullify (chr_t * *input*, size_t *len*)

Create a null-terminated string out of a block of memory and return it as a newly allocated nuller.

Parameters:

input a pointer to the data block to be copied.

len the size, in bytes, of the data block to be copied before being terminated with a null byte.

Returns:

NULL on failure, or a pointer to the new null-terminated string on success.

Definition at line 719 of file allocation.c.

References CONTIGUOUS, HEAP, MANAGED_T, mm_copy(), st_alloc_opts(), st_data_get(), and st_length_set().

Referenced by portal_message_body().

bool_t st_opt_get (stringer_t * *s*, uint32_t *opt*)

opts.c

opts.c

Parameters:

s the managed string to be checked. *opt* a bitmask of the managed string options to be tested.

Returns:

-1 on error, 0 if *opt* is not set, and 1 if *opt* is set.

Definition at line 53 of file *opts.c*.

References *log_pedantic*, *MEMORYBUF*, *st_info_opts()*, and *st_valid_opts()*.

Referenced by *contact_free()*, and *contact_name()*.

int_t st_opt_set (stringer_t * *s*, uint32_t *opt*, bool_t *enabled*)

Enable or disable a set of option(s) for a managed string, with validity testing.

Parameters:

s the input managed string.

opt the bitmask of option(s) to be enabled or disabled for the managed string.

enabled if true, set the option(s); if false, disable them.

Returns:

-1 on error, 0 if successfully disabled, or 1 if successfully enabled.

Definition at line 22 of file *opts.c*.

References *log_pedantic*, *MEMORYBUF*, *st_info_opts()*, and *st_valid_opts()*.

Referenced by *contact_name()*.

stringer_t* st_output (stringer_t * *output*, size_t *len*)

Return a suitable managed string to store data of a specified length.

Note:

If a NULL output string is specified, one will be allocated for the caller.

Parameters:

output the input managed string (can be NULL).

len the size, in bytes, of the requested data buffer.

Returns:

NULL on failure or if input was not large enough or a pointer to a validated output managed string.

Definition at line 690 of file *allocation.c*.

References *log_pedantic*, *st_alloc*, *st_avail_get()*, and *st_valid_destination()*.

Referenced by *symmetric_key()*, and *symmetric_vector()*.

stringer_t stringer_t stringer_t* st_quick (stringer_t * *s*, chr_t * *format*, ...)

stringer_t* st_realloc (stringer_t * *s*, size_t *len*)

Reallocate a managed string to a specified size.

Note:

The caller can request a new size that is either smaller (truncated) or larger than the original string. Only these types of strings can be reallocated: nuller, block, managed, and mapped. The following logic is

applied to string reallocation requests: For jointed strings, a new data buffer is allocated of the requested length, the original contents copied in, the data field of the new string is set, and the original data buffer freed. The stringer header returned to the caller resides at the same address as the original. For contiguous strings, a new data buffer is allocated for both the requested length and the new stringer header, and the data of both parts are copied from the original string to the resulting one. For block strings, the length field of the resulting string is set to the requested length. For managed strings, the length field of the resulting string is set to the original length, but the available field is adjusted accordingly. For mapped strings, an aligned `mremap()` call is made, and both the length and available fields are set to the new length.

Parameters:

s the managed string to have its size reallocated.
len the new size, in bytes, of the resulting managed string.

Returns:

NULL on failure or a pointer to the newly reallocated managed string on success.

Definition at line 552 of file `allocation.c`.

References `block_t`, `BLOCK_T`, `CONTIGUOUS`, `data`, `JOINTED`, `log_pedantic`, `magma`, `managed_t`, `MANAGED_T`, `mapped_t`, `MAPPED_T`, `magma_t::memory`, `MEMORYBUF`, `mm_alloc()`, `mm_copy()`, `mm_free()`, `mm_sec_alloc()`, `mm_sec_free()`, `nuller_t`, `NULLER_T`, `magma_t::page_length`, `magma_t::secure`, `SECURE`, `st_info_opts()`, `st_length_get()`, `st_valid_opts()`, and `system_limit_cur()`.

Referenced by `serialize_int16()`, `serialize_int32()`, `serialize_int64()`, `serialize_ns()`, `serialize_st()`, `serialize_uint16()`, `serialize_uint32()`, `serialize_uint64()`, `smtp_data_read()`, and `st_append_opts()`.

`int_t st_replace (stringer_t ** target, stringer_t * pattern, stringer_t * replacement)`

`replace.c`

Definition at line 24 of file `replace.c`.

References `log_pedantic`, `PLACER`, `st_alloc`, `st_char_get()`, `st_cmp_cs_starts()`, `st_data_get()`, `st_empty_out()`, `st_free()`, `st_length_set()`, and `st_valid_free()`.

Referenced by `contact_business()`, `imap_folder_create()`, `imap_folder_name_escaped()`, `imap_folder_rename()`, `pop_retr()`, `pop_top()`, `register_print_message()`, `register_print_step1()`, `register_print_step2()`, and `teacher_process()`.

`stringer_t stringer_t stringer_t size_t st_sprint (stringer_t * s, chr_t * format, ...)`

`stringer_t* st_swap (stringer_t * target, uchr_t pattern, uchr_t replacement)`

Replace all instances of one character in a managed string with another.

Parameters:

target the input string which will be transformed by the character replacement.
pattern the character to be searched and replaced in the target string.
replacement the character to be substituted for the pattern character in the target string.

Returns:

a pointer to the target managed string.

Definition at line 114 of file `replace.c`.

References `log_check`, `log_pedantic`, and `st_empty_out()`.

Referenced by `process_kill()`.

uchar_t* st_uchar_get (stringer_t * s)

Retrieve an unsigned character pointer to a managed string's data buffer.

See also:

st_data_get()

Parameters:

s the input managed string.

Returns:

NULL on failure or for an improperly constructed string; otherwise, a pointer to the string's data.

Definition at line 169 of file data.c.

References **st_data_get()**.

Referenced by **http_parse_origin()**.

bool_t st_valid_append (uint32_t opts)

Determine whether the managed string options allow for data appending.

Parameters:

opts the options value for the managed string.

Returns:

true if permitted, or false if options are invalid, the string is not jointed, or either managed or mapped.

Definition at line 55 of file validate.c.

References **JOINTED**, **MANAGED_T**, **MAPPED_T**, and **st_valid_opts()**.

Referenced by **st_append_opts()**.

bool_t st_valid_avail (uint32_t opts)

Determine whether a managed string tracks the total allocated buffer size.

Parameters:

opts the managed string's options value.

Returns:

false if buffer size isn't tracked; true otherwise (the managed string is managed or mapped).

Definition at line 124 of file validate.c.

References **MANAGED_T**, **MAPPED_T**, and **st_valid_opts()**.

Referenced by **base64_decode()**, **base64_decode_mod()**, **base64_encode()**, **base64_encode_mod()**, **digest_hash()**, **hex_decode_st()**, **hex_encode_st()**, **ip_subnet()**, **rand_write()**, and **st_avail_set()**.

bool_t st_valid_destination (uint32_t opts)

Determine whether the managed string options allow for a data store operation.

Parameters:

opts the options value for the managed string.

Returns:

true if permitted, or false if options are invalid, or indicate the managed string is constant.

Definition at line 39 of file validate.c.

References CONSTANT_T, and st_valid_opts().

Referenced by base64_decode(), base64_decode_mod(), base64_encode(), base64_encode_mod(), digest_hash(), hex_decode_st(), hex_encode_st(), ip_presentation(), ip_reversed(), ip_standard(), ip_subnet(), rand_write(), st_merge_opts(), st_output(), st_vaprint_opts(), and st_vsprint().

bool_t st_valid_free (uint32_t *opts*)

Determine whether a managed string is allowed to be freed.

Parameters:

opts the options value of the managed string to be evaluated.

Returns:

true if the managed string can be freed; false otherwise (it is allocated on the stack or contains foreign data).

Definition at line 73 of file validate.c.

References FOREIGNDATA, PLACER_T, st_valid_opts(), and STACK.

Referenced by st_free(), and st_replace().

bool_t st_valid_jointed (uint32_t *opts*)

Determine whether the managed string options are a validly jointed.

Parameters:

opts the options value to test.

Returns:

false if the options are invalid or if the string isn't jointed; true if the string is jointed.

Definition at line 107 of file validate.c.

References JOINTED, and st_valid_opts().

Referenced by st_data_set().

bool_t st_valid_opts (uint32_t *opts*)

Check to see that a managed string has a valid combination of allocation options.

Note:

The following rules are enforced: 1. Each managed string must only be one of the following: a. constant, nuller, block, placer, managed, or mapped. b. jointed or contiguous. c. allocated on the stack, heap, or secure. 2. A placer cannot be contiguous. 3. A constant must be contiguous and be allocated on the stack. 4. Mapped strings must be contiguous and on the heap.

Parameters:

opts the managed string option mask to be validated.

Returns:

true if the options represent valid managed string allocation options, or false if they do not.

Definition at line 149 of file validate.c.

References `bits_count()`, `BLOCK_T`, `CONSTANT_T`, `CONTIGUOUS`, `HEAP`, `JOINTED`, `MANAGED_T`, `MAPPED_T`, `NULLER_T`, `PLACER_T`, `SECURE`, and `STACK`.

Referenced by `st_alloc_opts()`, `st_avail_get()`, `st_data_get()`, `st_dupe_opts()`, `st_length_get()`, `st_length_int()`, `st_opt_get()`, `st_opt_set()`, `st_realloc()`, `st_valid_append()`, `st_valid_avail()`, `st_valid_destination()`, `st_valid_free()`, `st_valid_joined()`, `st_valid_placer()`, `st_valid_tracked()`, and `st_wipe()`.

bool_t st_valid_placer (uint32_t opts)

A sanity check to determine whether the managed string is a valid placer.

Note:

The following criteria must be satisfied: placer bit set, jointed bit set, either stack, heap, or secure set, and no other bits other than these mentioned should be set.

Parameters:

opts the managed string options value to be evaluated.

Returns:

true if the option value reflects a valid placer, or false otherwise.

Definition at line 22 of file validate.c.

References `HEAP`, `JOINTED`, `PLACER_T`, `SECURE`, `st_valid_opts()`, and `STACK`.

bool_t st_valid_tracked (uint32_t opts)

Determine whether a managed string provides for data length tracking.

Parameters:

opts the options value of the managed string.

Returns:

false if no length tracking is available; true if there is (string is a placer, managed, or mapped).

Definition at line 91 of file validate.c.

References `MANAGED_T`, `MAPPED_T`, `PLACER_T`, and `st_valid_opts()`.

Referenced by `base64_decode()`, `base64_decode_mod()`, `base64_encode()`, `base64_encode_mod()`, `digest_hash()`, `hex_decode_st()`, `hex_encode_st()`, `ip_presentation()`, `ip_reversed()`, `ip_standard()`, `rand_write()`, `st_dupe_opts()`, `st_length_set()`, `st_merge_opts()`, `st_vaprint_opts()`, `st_vsprint()`, `st_wipe()`, `symmetric_key()`, and `symmetric_vector()`.

stringer_t stringer_t stringer_t size_t stringer_t* st_vaprint_opts (uint32_t opts, chr_t * format, va_list args)

Return a managed string containing sprintf()-style formatted data.

Parameters:

opts an options value to be passed to the allocation of the resulting managed string.
format a format string for the output string data.
args a variable argument list of parameters to be formatted as output.

Returns:

NULL on failure or a managed string containing the final formatted data on success.
Definition at line 106 of file print.c.

References `length`, `log_pedantic`, `MEMORYBUF`, `st_alloc_opts()`, `st_data_get()`, `st_free()`, `st_info_opts()`, `st_length_set()`, `st_valid_destination()`, and `st_valid_tracked()`.

Referenced by `st_aprint()`, and `st_aprint_opts()`.

size_t st_vsprintf (stringer_t * s, chr_t * format, va_list args)

Print to a managed string and return the number of bytes written.

See also:

`vsnprintf()`

Note:

If the destination string pointer is NULL the function will simply return how much room would have been necessary.

Parameters:

s a pointer to the managed string that will receive the output of the print operation.
format a format string specifying the arguments of the print operation.
args a `va_list` containing the parameters to the print format string.

Returns:

-1 on failure, or the number of characters printed to the string, excluding the terminating null byte.
Definition at line 24 of file print.c.

References `length`, `log_pedantic`, `MEMORYBUF`, `st_avail_get()`, `st_data_get()`, `st_info_opts()`, `st_length_set()`, `st_valid_destination()`, and `st_valid_tracked()`.

Referenced by `st_quick()`, and `st_sprint()`.

bool_t void st_wipe (stringer_t * s)

Wipe all of a managed string's allocated memory and if applicable, reset its length.

Parameters:

s the managed string to be wiped.

Returns:

This function returns no value.
Definition at line 179 of file data.c.

References `log_pedantic`, `MEMORYBUF`, `mm_wipe()`, `st_avail_get()`, `st_data_get()`, `st_info_opts()`, `st_length_set()`, `st_valid_opts()`, and `st_valid_tracked()`.

Referenced by `ip_standard()`, and `pop_pass()`.

Variable Documentation

block_t

Definition at line 59 of file strings.h.

Referenced by st_alloc_opts(), st_data_get(), st_data_set(), st_free(), st_length_get(), and st_realloc().

constant_t

Definition at line 48 of file strings.h.

Referenced by st_data_get(), and st_length_get().

managed_t

Definition at line 72 of file strings.h.

Referenced by st_alloc_opts(), st_avail_get(), st_avail_set(), st_data_get(), st_data_set(), st_free(), st_length_get(), st_length_set(), and st_realloc().

mapped_t

Definition at line 80 of file strings.h.

Referenced by st_alloc_opts(), st_avail_get(), st_avail_set(), st_data_get(), st_free(), st_length_get(), st_length_set(), and st_realloc().

nuller_t

Definition at line 53 of file strings.h.

Referenced by st_alloc_opts(), st_data_get(), st_data_set(), st_free(), st_length_get(), and st_realloc().

placer_t

Definition at line 65 of file strings.h.

Referenced by alert_alloc(), alias_alloc(), ar_field_pl(), cache_config(), contact_alloc(), contact_detail_alloc(), domain_alloc(), get_header_opt(), get_header_value_noopt(), http_data_value_parse(), http_parse_context(), http_parse_header(), http_parse_method(), http_parse_pairs(), http_response_allow_cross(), imap_build_array(), imap_fetch_body(), imap_fetch_body_header(), imap_fetch_body_part(), imap_fetch_envelope(), imap_folder_create(), imap_folder_remove(), imap_folder_rename(), imap_narrow_messages(), imap_parse_address(), imap_parse_address_breaker(), imap_parse_address_part(), imap_search_messages_date(), imap_search_messages_date_compare(), imap_search_messages_header(), imap_search_messages_range(), ip_str_subnet(), lib_load_openssl(), magma_folder_alloc(), mail_add_forward_headers(), mail_add_outbound_headers(), mail_mime_split(), mail_mime_type_parameters(), mail_mime_update(), mail_mod_subject(), message_alloc(), meta_folder_stats_tag_alloc(), multipart_get_boundary(), nvp_parse(), pl_init(), pl_null(), pl_set(), portal_get_upload_attachment(), portal_upload(), relay_config(), sanity_check(), servers_config(), smtp_auth_plain(), smtp_check_authorized_from(), smtp_check_filters(), smtp_parse_mail_from_path(), spf_check(), st_alloc_opts(), st_data_get(), st_data_set(), st_free(), st_length_get(), st_length_set(), stamp_counter_check(), str_tok_get_bl(), str_tok_get_count_bl(), and user_config_entry_alloc().

magma/core/strings/validate.c File Reference

A collection of functions used to validate stringer allocation option combinations.

```
#include "magma.h"
```

Functions

- **bool_t st_valid_placer** (uint32_t opts)
 - *A sanity check to determine whether the managed string is a valid placer.* **bool_t st_valid_destination** (uint32_t opts)
 - *Determine whether the managed string options allow for a data store operation.* **bool_t st_valid_append** (uint32_t opts)
 - *Determine whether the managed string options allow for data appending.* **bool_t st_valid_free** (uint32_t opts)
 - *Determine whether a managed string is allowed to be freed.* **bool_t st_valid_tracked** (uint32_t opts)
 - *Determine whether a managed string provides for data length tracking.* **bool_t st_valid_jointed** (uint32_t opts)
 - *Determine whether the managed string options are a validly jointed.* **bool_t st_valid_avail** (uint32_t opts)
 - *Determine whether a managed string tracks the total allocated buffer size.* **bool_t st_valid_opts** (uint32_t opts)
-
- Check to see that a managed string has a valid combination of allocation options.*

Detailed Description

A collection of functions used to validate stringer allocation option combinations.

\$Author\$ \$Date\$ \$Revision:\$

Definition in file **validate.c**.

Function Documentation

bool_t st_valid_append (uint32_t *opts*)

Determine whether the managed string options allow for data appending.

Parameters:

opts the options value for the managed string.

Returns:

true if permitted, or false if options are invalid, the string is not jointed, or either managed or mapped.

Definition at line 55 of file validate.c.

References JOINTED, MANAGED_T, MAPPED_T, and st_valid_opts().

Referenced by st_append_opts().

bool_t st_valid_avail (uint32_t *opts*)

Determine whether a managed string tracks the total allocated buffer size.

Parameters:

opts the managed string's options value.

Returns:

false if buffer size isn't tracked; true otherwise (the managed string is managed or mapped).

Definition at line 124 of file validate.c.

References MANAGED_T, MAPPED_T, and st_valid_opts().

Referenced by base64_decode(), base64_decode_mod(), base64_encode(), base64_encode_mod(), digest_hash(), hex_decode_st(), hex_encode_st(), ip_subnet(), rand_write(), and st_avail_set().

bool_t st_valid_destination (uint32_t opts)

Determine whether the managed string options allow for a data store operation.

Parameters:

opts the options value for the managed string.

Returns:

true if permitted, or false if options are invalid, or indicate the managed string is constant.

Definition at line 39 of file validate.c.

References CONSTANT_T, and st_valid_opts().

Referenced by base64_decode(), base64_decode_mod(), base64_encode(), base64_encode_mod(), digest_hash(), hex_decode_st(), hex_encode_st(), ip_presentation(), ip_reversed(), ip_standard(), ip_subnet(), rand_write(), st_merge_opts(), st_output(), st_vaprint_opts(), and st_vsprint().

bool_t st_valid_free (uint32_t opts)

Determine whether a managed string is allowed to be freed.

Parameters:

opts the options value of the managed string to be evaluated.

Returns:

true if the managed string can be freed; false otherwise (it is allocated on the stack or contains foreign data).

Definition at line 73 of file validate.c.

References FOREIGNDATA, PLACER_T, st_valid_opts(), and STACK.

Referenced by st_free(), and st_replace().

bool_t st_valid_jointed (uint32_t opts)

Determine whether the managed string options are a validly jointed.

Parameters:

opts the options value to test.

Returns:

false if the options are invalid or if the string isn't jointed; true if the string is jointed.

Definition at line 107 of file validate.c.

References JOINTED, and st_valid_opts().

Referenced by `st_data_set()`.

bool_t st_valid_opts (uint32_t opts)

Check to see that a managed string has a valid combination of allocation options.

Note:

The following rules are enforced: 1. Each managed string must only be one of the following: a. constant, nuller, block, placer, managed, or mapped. b. jointed or contiguous. c. allocated on the stack, heap, or secure. 2. A placer cannot be contiguous. 3. A constant must be contiguous and be allocated on the stack. 4. Mapped strings must be contiguous and on the heap.

Parameters:

opts the managed string option mask to be validated.

Returns:

true if the options represent valid managed string allocation options, or false if they do not.

Definition at line 149 of file `validate.c`.

References `bits_count()`, `BLOCK_T`, `CONSTANT_T`, `CONTIGUOUS`, `HEAP`, `JOINTED`, `MANAGED_T`, `MAPPED_T`, `NULLER_T`, `PLACER_T`, `SECURE`, and `STACK`.

Referenced by `st_alloc_opts()`, `st_avail_get()`, `st_data_get()`, `st_dupe_opts()`, `st_length_get()`, `st_length_int()`, `st_opt_get()`, `st_opt_set()`, `st_realloc()`, `st_valid_append()`, `st_valid_avail()`, `st_valid_destination()`, `st_valid_free()`, `st_valid_jointed()`, `st_valid_placer()`, `st_valid_tracked()`, and `st_wipe()`.

bool_t st_valid_placer (uint32_t opts)

A sanity check to determine whether the managed string is a valid placer.

Note:

The following criteria must be satisfied: placer bit set, jointed bit set, either stack, heap, or secure set, and no other bits other than these mentioned should be set.

Parameters:

opts the managed string options value to be evaluated.

Returns:

true if the option value reflects a valid placer, or false otherwise.

Definition at line 22 of file `validate.c`.

References `HEAP`, `JOINTED`, `PLACER_T`, `SECURE`, `st_valid_opts()`, and `STACK`.

bool_t st_valid_tracked (uint32_t opts)

Determine whether a managed string provides for data length tracking.

Parameters:

opts the options value of the managed string.

Returns:

false if no length tracking is available; true if there is (string is a placer, managed, or mapped).

Definition at line 91 of file `validate.c`.

References `MANAGED_T`, `MAPPED_T`, `PLACER_T`, and `st_valid_opts()`.

Referenced by `base64_decode()`, `base64_decode_mod()`, `base64_encode()`, `base64_encode_mod()`, `digest_hash()`, `hex_decode_st()`, `hex_encode_st()`, `ip_presentation()`, `ip_reversed()`, `ip_standard()`, `rand_write()`, `st_dupe_opts()`, `st_length_set()`, `st_merge_opts()`, `st_vaprint_opts()`, `st_vsprint()`, `st_wipe()`, `symmetric_key()`, and `symmetric_vector()`.

magma/core/thread/keys.c File Reference

Functions for handling thread local storage keys.

```
#include "magma.h"
```

Functions

- `int tkey_init` (`pthread_key_t *key`, `void(*destructor)(void *)`)
 - *Create a thread-specific data key.* `void * tkey_get` (`pthread_key_t key`)
 - *Get the calling thread's value for a pthread key.* `int tkey_set` (`pthread_key_t key`, `void *value`)
- Set the calling thread's value for a pthread key.*
-

Detailed Description

Functions for handling thread local storage keys.

Definition in file **keys.c**.

Function Documentation

void* tkey_get (pthread_key_t key)

Get the calling thread's value for a pthread key.

keys.c

See also:

`pthread_getspecific()`

Parameters:

key the pthread key to be queried.

Returns:

NULL on failure or a pointer to the key's thread-specific data value on success.

Definition at line 40 of file keys.c.

int tkey_init (pthread_key_t * key, void(*)(void *) destructor)

Create a thread-specific data key.

See also:

`pthread_key_create()`

Parameters:

key a pointer to pthread key to be initialized for thread-local storage.

destructor if not NULL, an optional function pointer to be called at thread exit to release the data.

Returns:

0 on success, or an error number on failure.

Definition at line 22 of file keys.c.

References `log_pedantic`.

int `key_set` (`pthread_key_t` *key*, `void *` *value*)

Set the calling thread's value for a pthread key.

See also:

`pthread_setspecific()`

Parameters:

key the pthread key to have its thread-local value modified.

value a pointer to the new thread-specific value of the specified key.

Returns:

0 on success, or an error number on failure.

Definition at line 51 of file `keys.c`.

References `log_pedantic`.

magma/core/thread/mutex.c File Reference

Functions for thread coordination via a mutex.

```
#include "magma.h"
```

Functions

- `int mutex_init` (`pthread_mutex_t *lock`, `pthread_mutexattr_t *attr`)
- *Initialize a pthread mutex with the given attributes.* `int mutex_lock` (`pthread_mutex_t *lock`)
- *Acquire a pthread mutex, blocking if necessary.* `int mutex_unlock` (`pthread_mutex_t *lock`)
- *Unlock a pthread mutex.* `int mutex_destroy` (`pthread_mutex_t *lock`)

Free a pthread mutex.

Detailed Description

Functions for thread coordination via a mutex.

Definition in file **mutex.c**.

Function Documentation

`int mutex_destroy` (`pthread_mutex_t * lock`)

Free a pthread mutex.

mutex.c

See also:

`pthread_mutex_destroy()`

Parameters:

lock a pointer to the mutex to be freed.

Returns:

0 on success, or an error number on failure.

Definition at line 82 of file `mutex.c`.

References `log_pedantic`, and `MEMORYBUF`.

Referenced by `con_destroy()`, `meta_user_create()`, `meta_user_destroy()`, `pool_free()`, `protocol_secure()`, `queue_shutdown()`, `sess_destroy()`, `ssl_stop()`, `stacker_free()`, and `stats_shutdown()`.

`int mutex_init` (`pthread_mutex_t * lock`, `pthread_mutexattr_t * attr`)

Initialize a pthread mutex with the given attributes.

See also:

`pthread_mutex_init()`

Parameters:

lock a pointer to a mutex that will be initialized.

attr a pointer to a mutex attributes holder, or NULL to use system default values.

Returns:

0 on success, or an error number on failure.

Definition at line 22 of file mutex.c.

References log_pedantic, and MEMORYBUF.

Referenced by con_init(), meta_user_create(), pool_alloc(), queue_init(), ssl_start(), stacker_alloc(), and stats_init().

int mutex_lock (pthread_mutex_t * *lock*)

Acquire a pthread mutex, blocking if necessary.

See also:

pthread_mutex_lock()

Parameters:

lock a pointer to the mutex to be locked.

Returns:

0 on success, or an error number on failure.

Definition at line 42 of file mutex.c.

References log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, and MEMORYBUF.

Referenced by con_decrement_refs(), con_increment_refs(), con_reverse_check(), con_reverse_domain(), con_reverse_enqueue(), con_reverse_status(), dequeue(), log_disable(), log_enable(), log_internal(), meta_user_ref_add(), meta_user_ref_dec(), meta_user_ref_stamp(), meta_user_ref_total(), meta_user_rlock(), meta_user_wlock(), mm_sec_alloc(), mm_sec_free(), mm_sec_stats(), pattern_check(), pattern_stop(), pattern_update(), pool_get_failures(), pool_get_obj(), pool_pull(), pool_release(), pool_set_obj(), pool_swap_obj(), portal_endpoint_attachments_add(), portal_endpoint_attachments_remove(), portal_endpoint_messages_compose(), portal_endpoint_messages_send(), portal_get_upload_attachment(), requeue(), sess_number(), sess_ref_add(), sess_ref_dec(), sess_ref_stamp(), sess_ref_total(), sess_refresh_check(), sess_refresh_flush(), sess_refresh_stamp(), sess_trigger(), signal_refresh(), spool_check(), spool_check_file(), spool_error_stats(), spool_mktemp(), spool_start(), ssl_locking_callback(), stacker_nodes(), stacker_pop(), stacker_push(), statistics_refresh(), stats_adjust_by_name(), stats_adjust_by_num(), stats_decrement_by_name(), stats_decrement_by_num(), stats_get_name(), stats_get_name_pos(), stats_get_value_by_name(), stats_get_value_by_num(), stats_increment_by_name(), stats_increment_by_num(), stats_set_by_name(), stats_set_by_num(), status(), status_get(), status_set(), tank_cycle(), and xml_error().

int mutex_unlock (pthread_mutex_t * *lock*)

Unlock a pthread mutex.

See also:

pthread_mutex_unlock()

Parameters:

lock a pointer to the mutex to be unlocked.

Returns:

0 on success, or an error number on failure.

Definition at line 62 of file mutex.c.

References log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, and MEMORYBUF.

Referenced by con_decrement_refs(), con_increment_refs(), con_reverse_check(), con_reverse_domain(), con_reverse_enqueue(), con_reverse_status(), dequeue(), log_disable(), log_enable(), log_internal(), meta_user_ref_add(), meta_user_ref_dec(), meta_user_ref_stamp(), meta_user_ref_total(), meta_user_unlock(), mm_sec_alloc(), mm_sec_free(), mm_sec_stats(), pattern_check(), pattern_stop(), pattern_update(), pool_get_failures(), pool_get_obj(), pool_pull(), pool_release(), pool_set_obj(), pool_swap_obj(), portal_endpoint_attachments_add(), portal_endpoint_attachments_remove(), portal_endpoint_messages_compose(), portal_endpoint_messages_send(), portal_get_upload_attachment(), requeue(), sess_number(), sess_ref_add(), sess_ref_dec(), sess_ref_stamp(), sess_ref_total(), sess_refresh_check(), sess_refresh_flush(), sess_refresh_stamp(), sess_trigger(), signal_refresh(), spool_check(), spool_check_file(), spool_error_stats(), spool_mktemp(), spool_start(), ssl_locking_callback(), stacker_nodes(), stacker_pop(), stacker_push(), statistics_refresh(), stats_adjust_by_name(), stats_adjust_by_num(), stats_decrement_by_name(), stats_decrement_by_num(), stats_get_name(), stats_get_name_pos(), stats_get_value_by_name(), stats_get_value_by_num(), stats_increment_by_name(), stats_increment_by_num(), stats_set_by_name(), stats_set_by_num(), status(), status_get(), status_set(), tank_cycle(), and xml_error().

magma/core/thread/rwlock.c File Reference

Functions for thread coordination via a read/write lock.

```
#include "magma.h"
```

Functions

- `int rwlock_init (pthread_rwlock_t *lock, pthread_rwlockattr_t *attr)`
- *Initialize a pthread read/write lock.* `int rwlock_lock_write (pthread_rwlock_t *lock)`
- *Attempt to acquire a pthread read/write lock for writing, blocking if necessary.* `int rwlock_lock_read (pthread_rwlock_t *lock)`
- *Attempt to acquire a pthread read/write lock for reading, blocking if necessary.* `int rwlock_unlock (pthread_rwlock_t *lock)`
- *Unlock a pthread read/write lock.* `int rwlock_destroy (pthread_rwlock_t *lock)`
- *Free a pthread read/write lock and free its resources.* `int rwlock_attr_init (pthread_rwlockattr_t *attr)`
- *Initialize a pthread read/write lock attributes object with default values.* `int rwlock_attr_destroy (pthread_rwlockattr_t *attr)`
- *Free a pthread read/write lock attributes object.* `int rwlock_attr_setkind (pthread_rwlockattr_t *attr, int pref)`
- *Set the kind of a pthread read/write lock attributes object.* `int rwlock_attr_getkind (pthread_rwlockattr_t *attr, int *pref)`

Get the kind of a pthread read/write lock attributes object.

Detailed Description

Functions for thread coordination via a read/write lock.

Definition in file **rwlock.c**.

Function Documentation

int **rwlock_attr_destroy (pthread_rwlockattr_t * *attr*)**

Free a pthread read/write lock attributes object.

rwlock.c

See also:

`pthread_rwlockattr_destroy()`

Parameters:

attr a pointer to the read/write lock attributes object to be freed.

Returns:

0 on success or an error number on failure.

Definition at line 126 of file **rwlock.c**.

References `log_pedantic`.

int **rwlock_attr_getkind (pthread_rwlockattr_t * *attr*, int * *pref*)**

Get the kind of a pthread read/write lock attributes object.

See also:

`pthread_rwlockattr_getkind_np()`

Parameters:

attr a pointer to the read/write lock attributes object to be queried.

pref a pointer to an integer that will receive the kind of the read/write lock.

Returns:

0 on success or an error number on failure.

Definition at line 160 of file `rwlock.c`.

References `log_pedantic`.

int `rwlock_attr_init` (`pthread_rwlockattr_t` * *attr*)

Initialize a pthread read/write lock attributes object with default values.

See also:

`pthread_rwlockattr_init()`

Parameters:

attr a pointer to the read/write lock attributes object to be initialized.

Returns:

0 on success or an error number on failure.

Definition at line 110 of file `rwlock.c`.

References `log_pedantic`.

int `rwlock_attr_setkind` (`pthread_rwlockattr_t` * *attr*, int *pref*)

Set the kind of a pthread read/write lock attributes object.

See also:

`pthread_rwlockattr_setkind_np()`

Parameters:

attr a pointer to the read/write lock attributes object to be adjusted.

pref the kind of the read/write lock: `PTHREAD_RWLOCK_PREFER_READER_NP`, `PTHREAD_RWLOCK_PREFER_WRITER_NP`, or `PTHREAD_RWLOCK_PREFER_WRITER_NONRECURSIVE_NP`.

Returns:

0 on success or an error number on failure.

Definition at line 143 of file `rwlock.c`.

References `log_pedantic`.

int `rwlock_destroy` (`pthread_rwlock_t` * *lock*)

Free a pthread read/write lock and free its resources.

See also:

`pthread_rwlock_destroy()`

Parameters:

lock a pointer to the read/write lock to be destroyed.

Returns:

0 on success or an error number on failure.

Definition at line 94 of file `rwlock.c`.

References `log_pedantic`.

Referenced by `inx_free()`, `magma_folder_free()`, and `message_free()`.

`int pthread_rwlock_init (pthread_rwlock_t * lock, pthread_rwlockattr_t * attr)`

Initialize a pthread read/write lock.

See also:

`pthread_rwlock_init()`

Parameters:

lock a pointer to the read/write lock to be initialized.

attr an optional pointer to a set of lock attributes, or the default attributes if NULL is specified.

Returns:

0 on success or an error number on failure.

Definition at line 22 of file `rwlock.c`.

References `log_pedantic`.

Referenced by `inx_alloc()`, `magma_folder_alloc()`, and `message_alloc()`.

`int pthread_rwlock_lock_read (pthread_rwlock_t * lock)`

Attempt to acquire a pthread read/write lock for reading, blocking if necessary.

See also:

`pthread_rwlock_rdlock()`

Parameters:

lock the read-write lock to be tried.

Returns:

0 on success or an error number on failure.

Definition at line 58 of file `rwlock.c`.

References `log_pedantic`.

Referenced by `inx_auto_read()`, `inx_lock_read()`, `neue_rlock()`, `register_abuse_check_blocklist()`, and `spool_mktemp()`.

`int pthread_rwlock_lock_write (pthread_rwlock_t * lock)`

Attempt to acquire a pthread read/write lock for writing, blocking if necessary.

See also:

`pthread_rwlock_wrlock()`

Parameters:

lock a pointer to the read/write lock to be acquired.

Returns:

0 on success or an error number on failure.

Definition at line 40 of file `rwlock.c`.

References `log_pedantic`.

Referenced by `inx_auto_write()`, `inx_lock_write()`, `neue_wlock()`, `register_blocklist_update()`, and `spool_cleanup()`.

int `rwlock_unlock` (`pthread_rwlock_t * lock`)

Unlock a pthread read/write lock.

See also:

`pthread_rwlock_unlock()`

Parameters:

lock a pointer to the read/write lock to be unlocked.

Returns:

0 on success or an error number on failure.

Definition at line 76 of file `rwlock.c`.

References `log_pedantic`.

Referenced by `inx_auto_unlock()`, `inx_unlock()`, `neue_unlock()`, `register_abuse_check_blocklist()`, `register_blocklist_update()`, `spool_cleanup()`, and `spool_mktemp()`.

magma/core/thread/thread.c File Reference

Functions for spawning new threads, and retrieving their exit statuses.

```
#include "magma.h"
```

Functions

- `pthread_t thread_get_thread_id` (void)
 - *Get the id of the calling thread.* `int_t thread_launch` (pthread_t *thread, void *function, void *data)
 - *Launch a thread to execute a specified function.* `pthread_t * thread_alloc` (void *function, void *data)
 - *Launch a function in a freshly created thread.* `int_t thread_join` (pthread_t thread)
 - *Block until a specified thread finishes execution.* `int_t thread_result` (pthread_t thread, void **result)
 - *Block until a specified thread finishes execution and store its exit value.* `int_t thread_signal` (pthread_t thread, int_t signal)
 - *Send a specified signal to a thread.* `int_t thread_cancel` (pthread_t thread)
 - *Send a cancellation request to a thread.* `void thread_cancel_enable` (void)
 - *Set the calling thread to be cancelable.* `void thread_cancel_disable` (void)
- Set the calling thread to be not cancelable.*
-

Detailed Description

Functions for spawning new threads, and retrieving their exit statuses.

Definition in file **thread.c**.

Function Documentation

pthread_t* thread_alloc (void * *function*, void * *data*)

Launch a function in a freshly created thread.

thread.c

See also:

`pthread_create_new()`

Parameters:

function a pointer to the function to be executed.

data a pointer to data to be passed to the executed function.

Returns:

NULL on failure, or a pointer to the newly created pthread on success.

Definition at line 63 of file thread.c.

References `log_pedantic`, `mm_alloc()`, `mm_free()`, and `thread_launch()`.

int_t thread_cancel (pthread_t *thread*)

Send a cancellation request to a thread.

Parameters:

thread the pthread id of the thread to be canceled.

Returns:

0 on success or a non-zero error number on failure.

Definition at line 133 of file thread.c.

References log_pedantic.

void thread_cancel_disable (void)

Set the calling thread to be not cancelable.

Returns:

This function returns no value.

Definition at line 164 of file thread.c.

Referenced by process_maint().

void thread_cancel_enable (void)

Set the calling thread to be cancelable.

Returns:

This function returns no value.

Definition at line 155 of file thread.c.

Referenced by process_maint().

pthread_t thread_get_thread_id (void)

Get the id of the calling thread.

Returns:

the id of the calling thread.

Definition at line 19 of file thread.c.

Referenced by rand_start(), rand_thread_start(), spool_mktemp(), and ssl_thread_id_callback().

int_t thread_join (pthread_t *thread*)

Block until a specified thread finishes execution.

Note:

pthread_join()

Parameters:

thread the thread to wait upon.

Returns:

0 on success, or an error number on failure.

Definition at line 88 of file thread.c.

References log_pedantic.

Referenced by process_stop(), and queue_shutdown().

int_t thread_launch (pthread_t * *thread*, void * *function*, void * *data*)

Launch a thread to execute a specified function.

See also:

pthread_create()

Parameters:

thread a pointer to a pthread object to receive the new thread id.

function a pointer to a function to be executed inside the new thread.

data a pointer to be data to be passed in as input to the called function.

Returns:

0 on success, or a non-zero error code on failure.

Definition at line 32 of file thread.c.

References log_pedantic, magma, magma_t::system, and magma_t::thread_stack_size.

Referenced by process_start(), queue_init(), and thread_alloc().

int_t thread_result (pthread_t *thread*, void ** *result*)

Block until a specified thread finishes execution and store its exit value.

Note:

pthread_join()

Parameters:

thread the thread to wait upon.

result a pointer to a block of memory to receive the target thread exit value.

Returns:

0 on success, or an error number on failure.

Definition at line 106 of file thread.c.

References log_pedantic.

int_t thread_signal (pthread_t *thread*, int_t *signal*)

Send a specified signal to a thread.

Parameters:

thread the target thread id.

signal the number of the signal to be delivered.

Returns:

0 on success or an error number on failure.

Definition at line 123 of file thread.c.

Referenced by process_stop(), and queue_signal().

magma/engine/context/thread.c File Reference

The thread start and stop functions.

```
#include "magma.h"
```

Functions

- void **thread_stop** (void)
- *Prepare a thread to exit by destroying its mysql and openssl-specific and associated mail cache data.* **bool_t thread_start** (void)

Setup a new thread by setting up signal handling and preparing it for mysql processing.

Detailed Description

The thread start and stop functions.

Definition in file **thread.c**.

Function Documentation

bool_t thread_start (void)

Setup a new thread by setting up signal handling and preparing it for mysql processing.

thread.c

Returns:

true on success or false on failure.

Definition at line 32 of file thread.c.

References `signal_thread_start()`, and `sql_thread_start()`.

Referenced by `dequeue()`, and `process_maint()`.

void thread_stop (void)

Prepare a thread to exit by destroying its mysql and openssl-specific and associated mail cache data.

Returns:

This function returns no value.

Definition at line 19 of file thread.c.

References `mail_cache_thread_stop()`, `sql_thread_stop()`, and `ssl_thread_stop()`.

Referenced by `dequeue()`, and `process_maint()`.

magma/core/thread/thread.h File Reference

Interface for providing pthread functionality.

Functions

- `pthread_t * thread_alloc` (void *function, void *data)
 - *thread.c* `int_t thread_cancel` (pthread_t thread)
 - *Send a cancellation request to a thread.* void `thread_cancel_disable` (void)
 - *Set the calling thread to be not cancelable.* void `thread_cancel_enable` (void)
 - *Set the calling thread to be cancelable.* pthread_t `thread_get_thread_id` (void)
 - *Get the id of the calling thread.* int_t `thread_join` (pthread_t thread)
 - *Block until a specified thread finishes execution.* int_t `thread_launch` (pthread_t *thread, void *function, void *data)
 - *Launch a thread to execute a specified function.* int_t `thread_result` (pthread_t thread, void **result)
 - *Block until a specified thread finishes execution and store its exit value.* int_t `thread_signal` (pthread_t thread, int_t signal)
 - *Send a specified signal to a thread.* int `rwlock_attr_destroy` (pthread_rwlockattr_t *attr)
 - *rwlock.c* int `rwlock_attr_getkind` (pthread_rwlockattr_t *attr, int *pref)
 - *Get the kind of a pthread read/write lock attributes object.* int `rwlock_attr_init` (pthread_rwlockattr_t *attr)
 - *Initialize a pthread read/write lock attributes object with default values.* int `rwlock_attr_setkind` (pthread_rwlockattr_t *attr, int pref)
 - *Set the kind of a pthread read/write lock attributes object.* int `rwlock_destroy` (pthread_rwlock_t *lock)
 - *Free a pthread read/write lock and free its resources.* int `rwlock_init` (pthread_rwlock_t *lock, pthread_rwlockattr_t *attr)
 - *Initialize a pthread read/write lock.* int `rwlock_lock_read` (pthread_rwlock_t *lock)
 - *Attempt to acquire a pthread read/write lock for reading, blocking if necessary.* int `rwlock_lock_write` (pthread_rwlock_t *lock)
 - *Attempt to acquire a pthread read/write lock for writing, blocking if necessary.* int `rwlock_unlock` (pthread_rwlock_t *lock)
 - *Unlock a pthread read/write lock.* void * `tkey_get` (pthread_key_t key)
 - *keys.c* int `tkey_init` (pthread_key_t *key, void(*destructor)(void *))
 - *Create a thread-specific data key.* int `tkey_set` (pthread_key_t key, void *value)
 - *Set the calling thread's value for a pthread key.* int `mutex_destroy` (pthread_mutex_t *lock)
 - *mutex.c* int `mutex_init` (pthread_mutex_t *lock, pthread_mutexattr_t *attr)
 - *Initialize a pthread mutex with the given attributes.* int `mutex_lock` (pthread_mutex_t *lock)
 - *Acquire a pthread mutex, blocking if necessary.* int `mutex_unlock` (pthread_mutex_t *lock)
- Unlock a pthread mutex.*
-

Detailed Description

Interface for providing pthread functionality.

Definition in file `thread.h`.

Function Documentation

`int mutex_destroy (pthread_mutex_t * lock)`

mutex.c

mutex.c

See also:

`pthread_mutex_destroy()`

Parameters:

lock a pointer to the mutex to be freed.

Returns:

0 on success, or an error number on failure.

Definition at line 82 of file mutex.c.

References `log_pedantic`, and `MEMORYBUF`.

Referenced by `con_destroy()`, `meta_user_create()`, `meta_user_destroy()`, `pool_free()`, `protocol_secure()`, `queue_shutdown()`, `sess_destroy()`, `ssl_stop()`, `stacker_free()`, and `stats_shutdown()`.

int mutex_init (pthread_mutex_t * *lock*, pthread_mutexattr_t * *attr*)

Initialize a pthread mutex with the given attributes.

See also:

`pthread_mutex_init()`

Parameters:

lock a pointer to a mutex that will be initialized.

attr a pointer to a mutex attributes holder, or NULL to use system default values.

Returns:

0 on success, or an error number on failure.

Definition at line 22 of file mutex.c.

References `log_pedantic`, and `MEMORYBUF`.

Referenced by `con_init()`, `meta_user_create()`, `pool_alloc()`, `queue_init()`, `ssl_start()`, `stacker_alloc()`, and `stats_init()`.

int mutex_lock (pthread_mutex_t * *lock*)

Acquire a pthread mutex, blocking if necessary.

See also:

`pthread_mutex_lock()`

Parameters:

lock a pointer to the mutex to be locked.

Returns:

0 on success, or an error number on failure.

Definition at line 42 of file mutex.c.

References `log_options`, `M_LOG_PEDANTIC`, `M_LOG_STACK_TRACE`, and `MEMORYBUF`.

Referenced by `con_decrement_refs()`, `con_increment_refs()`, `con_reverse_check()`, `con_reverse_domain()`, `con_reverse_enqueue()`, `con_reverse_status()`, `dequeue()`, `log_disable()`, `log_enable()`, `log_internal()`, `meta_user_ref_add()`, `meta_user_ref_dec()`, `meta_user_ref_stamp()`, `meta_user_ref_total()`, `meta_user_rlock()`,

meta_user_wlock(), mm_sec_alloc(), mm_sec_free(), mm_sec_stats(), pattern_check(), pattern_stop(), pattern_update(), pool_get_failures(), pool_get_obj(), pool_pull(), pool_release(), pool_set_obj(), pool_swap_obj(), portal_endpoint_attachments_add(), portal_endpoint_attachments_remove(), portal_endpoint_messages_compose(), portal_endpoint_messages_send(), portal_get_upload_attachment(), requeue(), sess_number(), sess_ref_add(), sess_ref_dec(), sess_ref_stamp(), sess_ref_total(), sess_refresh_check(), sess_refresh_flush(), sess_refresh_stamp(), sess_trigger(), signal_refresh(), spool_check(), spool_check_file(), spool_error_stats(), spool_mktemp(), spool_start(), ssl_locking_callback(), stacker_nodes(), stacker_pop(), stacker_push(), statistics_refresh(), stats_adjust_by_name(), stats_adjust_by_num(), stats_decrement_by_name(), stats_decrement_by_num(), stats_get_name(), stats_get_name_pos(), stats_get_value_by_name(), stats_get_value_by_num(), stats_increment_by_name(), stats_increment_by_num(), stats_set_by_name(), stats_set_by_num(), status(), status_get(), status_set(), tank_cycle(), and xml_error().

int mutex_unlock (pthread_mutex_t * lock)

Unlock a pthread mutex.

See also:

pthread_mutex_unlock()

Parameters:

lock a pointer to the mutex to be unlocked.

Returns:

0 on success, or an error number on failure.

Definition at line 62 of file mutex.c.

References log_options, M_LOG_PEDANTIC, M_LOG_STACK_TRACE, and MEMORYBUF.

Referenced by con_decrement_refs(), con_increment_refs(), con_reverse_check(), con_reverse_domain(), con_reverse_enqueue(), con_reverse_status(), dequeue(), log_disable(), log_enable(), log_internal(), meta_user_ref_add(), meta_user_ref_dec(), meta_user_ref_stamp(), meta_user_ref_total(), meta_user_unlock(), mm_sec_alloc(), mm_sec_free(), mm_sec_stats(), pattern_check(), pattern_stop(), pattern_update(), pool_get_failures(), pool_get_obj(), pool_pull(), pool_release(), pool_set_obj(), pool_swap_obj(), portal_endpoint_attachments_add(), portal_endpoint_attachments_remove(), portal_endpoint_messages_compose(), portal_endpoint_messages_send(), portal_get_upload_attachment(), requeue(), sess_number(), sess_ref_add(), sess_ref_dec(), sess_ref_stamp(), sess_ref_total(), sess_refresh_check(), sess_refresh_flush(), sess_refresh_stamp(), sess_trigger(), signal_refresh(), spool_check(), spool_check_file(), spool_error_stats(), spool_mktemp(), spool_start(), ssl_locking_callback(), stacker_nodes(), stacker_pop(), stacker_push(), statistics_refresh(), stats_adjust_by_name(), stats_adjust_by_num(), stats_decrement_by_name(), stats_decrement_by_num(), stats_get_name(), stats_get_name_pos(), stats_get_value_by_name(), stats_get_value_by_num(), stats_increment_by_name(), stats_increment_by_num(), stats_set_by_name(), stats_set_by_num(), status(), status_get(), status_set(), tank_cycle(), and xml_error().

int rwlock_attr_destroy (pthread_rwlockattr_t * attr)

rwlock.c

rwlock.c

See also:

pthread_rwlockattr_destroy()

Parameters:

attr a pointer to the read/write lock attributes object to be freed.

Returns:

0 on success or an error number on failure.

Definition at line 126 of file `rwlock.c`.

References `log_pedantic`.

int `rwlock_attr_getkind` (`pthread_rwlockattr_t` * *attr*, int * *pref*)

Get the kind of a pthread read/write lock attributes object.

See also:

`pthread_rwlockattr_getkind_np()`

Parameters:

attr a pointer to the read/write lock attributes object to be queried.

pref a pointer to an integer that will receive the kind of the read/write lock.

Returns:

0 on success or an error number on failure.

Definition at line 160 of file `rwlock.c`.

References `log_pedantic`.

int `rwlock_attr_init` (`pthread_rwlockattr_t` * *attr*)

Initialize a pthread read/write lock attributes object with default values.

See also:

`pthread_rwlockattr_init()`

Parameters:

attr a pointer to the read/write lock attributes object to be initialized.

Returns:

0 on success or an error number on failure.

Definition at line 110 of file `rwlock.c`.

References `log_pedantic`.

int `rwlock_attr_setkind` (`pthread_rwlockattr_t` * *attr*, int *pref*)

Set the kind of a pthread read/write lock attributes object.

See also:

`pthread_rwlockattr_setkind_np()`

Parameters:

attr a pointer to the read/write lock attributes object to be adjusted.

pref the kind of the read/write lock: `PTHREAD_RWLOCK_PREFER_READER_NP`, `PTHREAD_RWLOCK_PREFER_WRITER_NP`, or `PTHREAD_RWLOCK_PREFER_WRITER_NONRECURSIVE_NP`.

Returns:

0 on success or an error number on failure.

Definition at line 143 of file rwlock.c.

References log_pedantic.

int rwlock_destroy (pthread_rwlock_t * *lock*)

Free a pthread read/write lock and free its resources.

See also:

pthread_rwlock_destroy()

Parameters:

lock a pointer to the read/write lock to be destroyed.

Returns:

0 on success or an error number on failure.

Definition at line 94 of file rwlock.c.

References log_pedantic.

Referenced by inx_free(), magma_folder_free(), and message_free().

int rwlock_init (pthread_rwlock_t * *lock*, pthread_rwlockattr_t * *attr*)

Initialize a pthread read/write lock.

See also:

pthread_rwlock_init()

Parameters:

lock a pointer to the read/write lock to be initialized.

attr an optional pointer to a set of lock attributes, or the default attributes if NULL is specified.

Returns:

0 on success or an error number on failure.

Definition at line 22 of file rwlock.c.

References log_pedantic.

Referenced by inx_alloc(), magma_folder_alloc(), and message_alloc().

int rwlock_lock_read (pthread_rwlock_t * *lock*)

Attempt to acquire a pthread read/write lock for reading, blocking if necessary.

See also:

pthread_rwlock_rdlock()

Parameters:

lock the read-write lock to be tried.

Returns:

0 on success or an error number on failure.

Definition at line 58 of file rwlock.c.

References log_pedantic.

Referenced by inx_auto_read(), inx_lock_read(), neue_rlock(), register_abuse_check_blocklist(), and spool_mktemp().

int rwlock_lock_write (pthread_rwlock_t * *lock*)

Attempt to acquire a pthread read/write lock for writing, blocking if necessary.

See also:

pthread_rwlock_wrlock()

Parameters:

lock a pointer to the read/write lock to be acquired.

Returns:

0 on success or an error number on failure.

Definition at line 40 of file rwlock.c.

References log_pedantic.

Referenced by inx_auto_write(), inx_lock_write(), neue_wlock(), register_blocklist_update(), and spool_cleanup().

int rwlock_unlock (pthread_rwlock_t * *lock*)

Unlock a pthread read/write lock.

See also:

pthread_rwlock_unlock()

Parameters:

lock a pointer to the read/write lock to be unlocked.

Returns:

0 on success or an error number on failure.

Definition at line 76 of file rwlock.c.

References log_pedantic.

Referenced by inx_auto_unlock(), inx_unlock(), neue_unlock(), register_abuse_check_blocklist(), register_blocklist_update(), spool_cleanup(), and spool_mktemp().

pthread_t* thread_alloc (void * *function*, void * *data*)

thread.c

thread.c

See also:

pthread_create_new()

Parameters:

function a pointer to the function to be executed.
data a pointer to data to be passed to the executed function.

Returns:

NULL on failure, or a pointer to the newly created pthread on success.
Definition at line 63 of file thread.c.
References log_pedantic, mm_alloc(), mm_free(), and thread_launch().

int_t thread_cancel (pthread_t *thread*)

Send a cancellation request to a thread.

Parameters:

thread the pthread id of the thread to be canceled.

Returns:

0 on success or a non-zero error number on failure.
Definition at line 133 of file thread.c.
References log_pedantic.

void thread_cancel_disable (void)

Set the calling thread to be not cancelable.

Returns:

This function returns no value.
Definition at line 164 of file thread.c.
Referenced by process_maint().

void thread_cancel_enable (void)

Set the calling thread to be cancelable.

Returns:

This function returns no value.
Definition at line 155 of file thread.c.
Referenced by process_maint().

pthread_t thread_get_thread_id (void)

Get the id of the calling thread.

Returns:

the id of the calling thread.

Definition at line 19 of file thread.c.

Referenced by rand_start(), rand_thread_start(), spool_mktemp(), and ssl_thread_id_callback().

int_t thread_join (pthread_t *thread*)

Block until a specified thread finishes execution.

Note:

pthread_join()

Parameters:

thread the thread to wait upon.

Returns:

0 on success, or an error number on failure.

Definition at line 88 of file thread.c.

References log_pedantic.

Referenced by process_stop(), and queue_shutdown().

int_t thread_launch (pthread_t * *thread*, void * *function*, void * *data*)

Launch a thread to execute a specified function.

See also:

pthread_create()

Parameters:

thread a pointer to a pthread object to receive the new thread id.

function a pointer to a function to be executed inside the new thread.

data a pointer to be data to be passed in as input to the called function.

Returns:

0 on success, or a non-zero error code on failure.

Definition at line 32 of file thread.c.

References log_pedantic, magma, magma_t::system, and magma_t::thread_stack_size.

Referenced by process_start(), queue_init(), and thread_alloc().

int_t thread_result (pthread_t *thread*, void ** *result*)

Block until a specified thread finishes execution and store its exit value.

Note:

pthread_join()

Parameters:

thread the thread to wait upon.

result a pointer to a block of memory to receive the target thread exit value.

Returns:

0 on success, or an error number on failure.

Definition at line 106 of file thread.c.

References log_pedantic.

int_t thread_signal (pthread_t *thread*, int_t *signal*)

Send a specified signal to a thread.

Parameters:

thread the target thread id.

signal the number of the signal to be delivered.

Returns:

0 on success or an error number on failure.

Definition at line 123 of file thread.c.

Referenced by process_stop(), and queue_signal().

void* tkey_get (pthread_key_t *key*)

keys.c

keys.c

See also:

pthread_getspecific()

Parameters:

key the pthread key to be queried.

Returns:

NULL on failure or a pointer to the key's thread-specific data value on success.

Definition at line 40 of file keys.c.

int tkey_init (pthread_key_t * *key*, void(*)(void *) *destructor*)

Create a thread-specific data key.

See also:

pthread_key_create()

Parameters:

key a pointer to pthread key to be initialized for thread-local storage.

destructor if not NULL, an optional function pointer to be called at thread exit to release the data.

Returns:

0 on success, or an error number on failure.

Definition at line 22 of file keys.c.

References log_pedantic.

int tkey_set (pthread_key_t *key*, void * *value*)

Set the calling thread's value for a pthread key.

See also:

pthread_setspecific()

Parameters:

key the pthread key to have its thread-local value modified.

value a pointer to the new thread-specific value of the specified key.

Returns:

0 on success, or an error number on failure.

Definition at line 51 of file keys.c.

References log_pedantic.

magma/core/type.c File Reference

A function for translating the M_TYPE enum into an equivalent string, typically so that a type code can be recorded as a string in error messages.

```
#include "magma.h"
```

Functions

- `char * type (M_TYPE type)`

Detailed Description

A function for translating the M_TYPE enum into an equivalent string, typically so that a type code can be recorded as a string in error messages.

Definition in file `type.c`.

Function Documentation

`char* type (M_TYPE type)`

Takes a type code and returns the fully enumerated string associated with that type.

Parameters:

type The type code to evaluate.

Returns:

Null terminated string containing type name. String is stored in static buffer and returned as a pointer.

Definition at line 22 of file `type.c`.

References `M_TYPE_BLOCK`, `M_TYPE_BOOLEAN`, `M_TYPE_DOUBLE`, `M_TYPE_ENUM`, `M_TYPE_FLOAT`, `M_TYPE_INT16`, `M_TYPE_INT32`, `M_TYPE_INT64`, `M_TYPE_INT8`, `M_TYPE_MULTI`, `M_TYPE_NULLER`, `M_TYPE_PLACER`, `M_TYPE_STRINGER`, `M_TYPE_UINT16`, `M_TYPE_UINT32`, `M_TYPE_UINT64`, and `M_TYPE_UINT8`.

Referenced by `ar_dupe()`, `ar_free()`, `cache_free()`, `cache_output_settings()`, `cache_set_value()`, `cache_validate()`, `config_free()`, `imap_parse_array()`, `imap_parse_dataitems()`, `mail_modify_part()`, `meta_data_delete_folder()`, `meta_data_fetch_folders()`, `meta_data_insert_folder()`, `meta_data_update_folder_name()`, `mt_dupe()`, `mt_get_char()`, `mt_get_length()`, `mt_get_number()`, `mt_get_type()`, `mt_is_empty()`, `mt_is_number()`, `mt_set_type()`, `portal_message_attachments()`, `portal_message_body()`, `relay_free()`, `relay_output_settings()`, `relay_set_value()`, `relay_validate()`, `servers_free()`, `servers_output_settings()`, `servers_set_value()`, and `servers_validate()`.

magma/engine/config/cache/cache.c File Reference

Functions for handling the cache instance configurations.

```
#include "magma.h"
```

```
#include "keys.h"
```

Functions

- void **cache_free** (void)
- *Free magma's cache server configuration options.* **cache_t * cache_alloc** (uint32_t **number**)
- *Allocate a new cache server configuration entry and initialize it to its default values.* **bool_t cache_validate** (void)
- *Validate all of the configured magma cache server instances.* void **cache_output_settings** (void)
- *Log the full contents of the magma cache configuration settings.* **bool_t cache_set_value** (cache_keys_t *setting, **cache_t** *cache, **stringer_t** *value)
- *Set the value of a key for a specified cache server configuration entry.* **bool_t cache_config** (**stringer_t** *name, **stringer_t** *value)
- *Set the value of a cache server configuration entry by name.* void **cache_output_help** (void)

Log all cache key information to be returned via "magma -h".

Detailed Description

Functions for handling the cache instance configurations.

\$Author\$ \$Author\$ \$Revision\$

Definition in file **cache.c**.

Function Documentation

cache_t* cache_alloc (uint32_t *number*)

Allocate a new cache server configuration entry and initialize it to its default values.

Parameters:

number the index of the cache server instance in the global cache server array.

Returns:

NULL on failure, or a pointer to the requested cache server configuration entry on success.

Definition at line 69 of file cache.c.

References magma_t::cache, cache_keys, cache_set_value(), magma_t::iface, log_critical, magma, and mm_alloc().

Referenced by cache_config().

bool_t cache_config (**stringer_t** * *name*, **stringer_t** * *value*)

Set the value of a cache server configuration entry by name.

Note:

This function is designed to operate on lines from a configuration source, like a file or database. When a named server is referenced for the first time, a new cache server configuration instance will be allocated.

Parameters:

name a managed string containing the human-readable cache server configuration parameter to be set.
value a managed string containing the new value of the cache server config key.

Returns:

true if the value was successfully set, or false on failure.

Definition at line 430 of file cache.c.

References `bracket_extract_pl()`, `cache_alloc()`, `cache_keys`, `cache_set_value()`, `CONSTANT`, `log_critical`, `MAGMA_CACHE_INSTANCES`, `NULLER`, `pl_char_get()`, `pl_empty()`, `pl_length_get()`, `PLACER`, `placer_t`, `st_char_get()`, `st_cmp_ci_eq()`, `st_cmp_ci_starts()`, `st_length_get()`, `st_length_int()`, and `uint32_conv_bl()`.

Referenced by `config_load_cmdline_settings()`, `config_load_database_settings()`, and `config_load_file_settings()`.

void cache_free (void)

Free magma's cache server configuration options.

Note:

This function assumes all worker threads have finished, and should only be called during the normal shutdown process.

Parameters:

This function returns no value.

Definition at line 22 of file cache.c.

References `magma_t::cache`, `cache_keys`, `magma_t::iface`, `log_pedantic`, `M_TYPE_BLOCK`, `M_TYPE_BOOLEAN`, `M_TYPE_DOUBLE`, `M_TYPE_ENUM`, `M_TYPE_FLOAT`, `M_TYPE_INT16`, `M_TYPE_INT32`, `M_TYPE_INT64`, `M_TYPE_INT8`, `M_TYPE_NULLER`, `M_TYPE_STRINGER`, `M_TYPE_UINT16`, `M_TYPE_UINT32`, `M_TYPE_UINT64`, `M_TYPE_UINT8`, `magma`, `MAGMA_CACHE_INSTANCES`, `mm_free()`, `cache_keys_t::norm`, `ns_empty()`, `ns_free()`, `cache_keys_t::offset`, `st_empty()`, `st_free()`, `type()`, and `multi_t::type`.

Referenced by `config_free()`.

void cache_output_help (void)

Log all cache key information to be returned via "magma -h".

Returns:

This function returns no value.

Definition at line 489 of file cache.c.

References `multi_t::bl`, `cache_keys`, `config_output_value_generic()`, `log_info`, `log_options`, `M_LOG_FILE_DISABLE`, `M_LOG_FUNCTION_DISABLE`, `M_LOG_LINE_DISABLE`, `M_LOG_LINE_FEED_DISABLE`, `M_LOG_STACK_TRACE_DISABLE`, `M_LOG_TIME_DISABLE`, `cache_keys_t::norm`, `cache_keys_t::required`, and `multi_t::val`.

Referenced by `config_output_help()`.

void cache_output_settings (void)

Log the full contents of the magma cache configuration settings.

Note:

This logs all the details of each memcached instance configured to work with magma.

Returns:

This function returns no value.

Definition at line 212 of file cache.c.

References magma_t::cache, cache_keys, magma_t::iface, log_info, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_CACHE_INSTANCES, cache_keys_t::norm, ns_empty(), cache_keys_t::offset, st_char_get(), st_empty(), st_length_int(), multi_t::type, and type().

Referenced by config_output_settings().

bool_t cache_set_value (cache_keys_t * setting, cache_t * cache, stringer_t * value)

Set the value of a key for a specified cache server configuration entry.

Note:

This function will allocate space for a copy of the key value, and return an error if it is not able to convert it to the proper key data type.

Parameters:

setting a pointer to the cache server key to be set.

cache a pointer to the cache server configuration entry to be modified.

value a managed string containing the new value of the key.

Returns:

true if the value was successfully set, or false on failure.

Definition at line 287 of file cache.c.

References multi_t::binary, CONSTANT, CONTIGUOUS, HEAP, multi_t::i32, multi_t::i64, multi_t::i8, int32_conv_st(), int64_conv_st(), int8_conv_st(), log_critical, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, MANAGED_T, cache_keys_t::name, cache_keys_t::norm, multi_t::ns, ns_dupe(), ns_empty(), ns_free(), ns_import(), cache_keys_t::offset, multi_t::st, st_char_get(), st_cmp_ci_eq(), st_dupe_opts(), st_empty(), st_free(), st_length_get(), type(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, uint16_conv_st(), uint32_conv_st(), uint64_conv_st(), uint8_conv_st(), and multi_t::val.

Referenced by cache_alloc(), and cache_config().

bool_t cache_validate (void)

Validate all of the configured magma cache server instances.

Note:

This makes set that all required config keys have been set, and that at least one host has been configured.

Returns:

true if all cache servers have been validated, or false on failure.

Definition at line 98 of file cache.c.

References magma_t::cache, cache_keys, multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, magma_t::iface, log_critical, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_CACHE_INSTANCES, cache_keys_t::norm, ns_empty(), cache_keys_t::offset, st_empty(), multi_t::type, type(), multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by config_validate_settings().

magma/objects/mail/cache.c File Reference

Functions used to cache a mail message in its parsed form.

```
#include "magma.h"
```

Functions

- void **mail_cache_destroy** (void *holder)
- *Free a cached mail message.* **bool_t mail_cache_start** (void)
- *Initialize the thread-specific data key used for the mail message cache.* void **mail_cache_stop** (void)
- *Destroy the thread-specific data key used for the mail message cache.* void **mail_cache_thread_stop** (void)
- *Free the thread-specific data for the mail message cache.* **stringer_t * mail_cache_get** (uint64_t messagenum)
- *Attempt to retrieve the contents of a message from the thread's cached data.* void **mail_cache_reset** (void)
- *Reset the thread's mail cache and free any held message.* void **mail_cache_set** (uint64_t messagenum, **stringer_t** *text)

Set the contents of the thread's mail cache.

Detailed Description

Functions used to cache a mail message in its parsed form.

Definition in file **cache.c**.

Function Documentation

void mail_cache_destroy (void * holder)

Free a cached mail message.

cache.c

Parameters:

holder a pointer to the cached mail message object to be freed.

Returns:

This function returns no value.

Definition at line 22 of file cache.c.

References `mm_free()`, `st_cleanup()`, and `mail_cache_t::text`.

Referenced by `mail_cache_start()`, and `mail_cache_thread_stop()`.

stringer_t* mail_cache_get (uint64_t messagenum)

Attempt to retrieve the contents of a message from the thread's cached data.

Note:

The thread's cache only has the capacity to store a single message.

Parameters:

messagenum the id of the message to be retrieved.

Returns:

NULL on failure or a managed string containing the message data on success.

Definition at line 83 of file cache.c.

References mail_cache_t::messagenum, st_dupe(), and mail_cache_t::text.

Referenced by mail_load_message().

void mail_cache_reset (void)

Reset the thread's mail cache and free any held message.

Returns:

This function returns no value.

Definition at line 102 of file cache.c.

References mail_cache_t::messagenum, st_cleanup(), and mail_cache_t::text.

Referenced by imap_session_destroy(), and pop_session_destroy().

void mail_cache_set (uint64_t *messagenum*, stringer_t * *text*)

Set the contents of the thread's mail cache.

Parameters:

messagenum the numerical id of the message to be cached. a managed string containing the contents of the specified message to be cached.

Returns:

This function returns no value.

Definition at line 121 of file cache.c.

References CONTIGUOUS, HEAP, log_pedantic, MANAGED_T, mail_cache_t::messagenum, mm_alloc(), mm_free(), st_cleanup(), st_dupe_opts(), and mail_cache_t::text.

Referenced by mail_load_message().

bool_t mail_cache_start (void)

Initialize the thread-specific data key used for the mail message cache.

Returns:

true on success or false on failure.

Definition at line 38 of file cache.c.

References log_pedantic, and mail_cache_destroy().

Referenced by process_start().

void mail_cache_stop (void)

Destroy the thread-specific data key used for the mail message cache.

Returns:

This function returns no value.

Definition at line 52 of file cache.c.

References log_pedantic.

Referenced by process_stop().

void mail_cache_thread_stop (void)

Free the thread-specific data for the mail message cache.

Returns:

This function returns no value.

Definition at line 65 of file cache.c.

References mail_cache_destroy().

Referenced by thread_stop().

magma/providers/consumers/cache.c File Reference

Distributed cache interface.
`#include "magma.h"`

Functions

- **bool_t lib_load_cache** (void)
- *Initialize the memcached library and bind dynamically to the exported functions that are required.* **const char * lib_version_cache** (void)
- *Return the version string of the memcached library.* **bool_t cache_start** (void)
- **void cache_stop** (void)
- *Destroy the memcached client pool and all active connections.* **void cache_flush** (void)
- **stringer_t * cache_get** (**stringer_t** *key)
- *Retrieve data from memcached by key.* **uint64_t cache_get_u64** (**stringer_t** *key)
- *Retrieve a 64 bit value from memcached by key.* **int_t cache_set** (**stringer_t** *key, **stringer_t** *object, **time_t** expiration)
- *Set a value in memcached by key.* **int_t cache_set_u64** (**stringer_t** *key, **uint64_t** value, **time_t** expiration)
- *Set a 64 bit value in memcached by key.* **int_t cache_add** (**stringer_t** *key, **stringer_t** *object, **time_t** expiration)
- *Add a new key/value pair to the memcached server.* **int_t cache_silent_add** (**stringer_t** *key, **stringer_t** *object, **time_t** expiration)
- *Add a new key/value pair to the memcached server, but suppress any error messages.* **int_t cache_append** (**stringer_t** *key, **stringer_t** *object, **time_t** expiration)
- *Append data to the value of a cached key on a memcached server.* **int_t cache_delete** (**stringer_t** *key)
- *Delete a key from memcached immediately.* **uint64_t cache_increment** (**stringer_t** *key, **uint64_t** offset, **uint64_t** initial, **time_t** expiration)

Increment the value stored with a key by memcached by a specified offset.

Variables

- **pool_t * cache_pool** = NULL

Detailed Description

Distributed cache interface.

Definition in file **cache.c**.

Function Documentation

int_t cache_add (**stringer_t** * key, **stringer_t** * object, **time_t** expiration)

Add a new key/value pair to the memcached server.

cache.c

Parameters:

key a managed string with the name of the key to be added to the cache.

object a managed string containing the initial value of the new key.

expiration an expiration time, in seconds, for the newly cached data.

Definition at line 284 of file cache.c.

References `log_info`, `memcached_add_d`, `memcached_strerror_d`, `PL_RESERVED`, `pool_get_obj()`, `pool_pull()`, `pool_release()`, `st_char_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by `mail_load_header()`, and `mail_load_message()`.

int_t cache_append (stringer_t * *key*, stringer_t * *object*, time_t *expiration*)

Append data to the value of a cached key on a memcached server.

Parameters:

key a managed string with the name of the target memcached key.

object a managed string containing the value of the data being appended to the original key.

expiration an expiration time, in seconds, for the modified cached data.

Definition at line 329 of file cache.c.

References `log_info`, `memcached_add_d`, `memcached_append_d`, `memcached_strerror_d`, `PL_RESERVED`, `pool_get_obj()`, `pool_pull()`, `pool_release()`, `st_char_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by `stamp_counter_increment()`.

int_t cache_delete (stringer_t * *key*)

Delete a key from memcached immediately.

Note:

`memcached_delete()`

Parameters:

key the name of the key to be deleted from the memcached server.

Returns:

0 on failure, or `MEMCACHED_SUCCESS` on success.

Definition at line 359 of file cache.c.

References `log_info`, `memcached_delete_d`, `memcached_strerror_d`, `PL_RESERVED`, `pool_get_obj()`, `pool_pull()`, `pool_release()`, `st_char_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by `lock_release()`.

void cache_flush (void)

Definition at line 130 of file cache.c.

References `log_info`, `memcached_flush_d`, `memcached_strerror_d`, `PL_RESERVED`, `pool_get_obj()`, `pool_pull()`, and `pool_release()`.

Referenced by `main()`.

stringer_t* cache_get (stringer_t * *key*)

Retrieve data from memcached by key.

Parameters:

key a managed string containing a key to be passed to memcached.

Returns:

NULL on failure, or a pointer to a managed string containing the cached data on success.

Definition at line 151 of file cache.c.

References `data`, `length`, `log_info`, `memcached_get_d`, `memcached_strerror_d`, `mm_free()`, `PL_RESERVED`, `pool_get_obj()`, `pool_pull()`, `pool_release()`, `st_char_get()`, `st_import()`, `st_length_get()`, and `st_length_int()`.

Referenced by `mail_load_header()`, `mail_load_message()`, `register_session_get()`, `smtp_check_greylist()`, `smtp_fetch_autoreply()`, `stamp_counter_check()`, and `teacher_data_get()`.

uint64_t cache_get_u64 (stringer_t * key)

Retrieve a 64 bit value from memcached by key.

Parameters:

key a managed string containing a key to be passed to memcached.

Returns:

NULL on failure, or the 64 bit value cached data on success.

Definition at line 191 of file cache.c.

References `data`, `length`, `log_info`, `memcached_get_d`, `memcached_strerror_d`, `mm_free()`, `PL_RESERVED`, `pool_get_obj()`, `pool_pull()`, `pool_release()`, `st_char_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by `contact_abuse_checks()`, `register_abuse_checks()`, and `smtp_reply()`.

uint64_t cache_increment (stringer_t * key, uint64_t offset, uint64_t initial, time_t expiration)

Increment the value stored with a key by memcached by a specified offset.

See also:

`memcached_increment_with_initial()`

Parameters:

key a managed string containing the name of the memcached key to be adjusted.

offset the offset by which the specified key's value should be incremented.

initial the initial value to seed the key if it does not already exist.

expiration the time, in seconds, when the cached key will expire.

Returns:

0 on failure, or the new incremented value of the data associated with the key, on success.

Definition at line 385 of file cache.c.

References `log_pedantic`, `memcached_increment_with_initial_d`, `memcached_strerror_d`, `PL_RESERVED`, `pool_get_obj()`, `pool_pull()`, `pool_release()`, `st_char_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by `contact_abuse_increment_history()`, `register_abuse_increment_history()`, `serial_get()`, and `serial_increment()`.

int_t cache_set (stringer_t * key, stringer_t * object, time_t expiration)

Set a value in memcached by key.

Parameters:

key a managed string containing a key to be passed to memcached.

object a managed string containing the new data to be associated with the supplied key.

expiration the expiration time of the cached data.

Returns:

0 on failure, or 1 on success.

Definition at line 237 of file cache.c.

References log_info, memcached_set_d, memcached_strerror_d, PL_RESERVED, pool_get_obj(), pool_pull(), pool_release(), st_char_get(), st_length_get(), and st_length_int().

Referenced by register_session_cache(), serial_reset(), smtp_check_greylist(), smtp_fetch_autoreply(), and teacher_data_save().

int_t cache_set_u64 (stringer_t * key, uint64_t value, time_t expiration)

Set a 64 bit value in memcached by key.

Parameters:

key a managed string containing a key to be passed to memcached.

value the new value to be associated with the supplied key.

expiration the expiration time of the cached data.

Returns:

NULL on failure, or the retrieved unsigned 64 bit cached data on success.

Definition at line 261 of file cache.c.

References log_info, memcached_set_d, memcached_strerror_d, PL_RESERVED, pool_get_obj(), pool_pull(), pool_release(), st_char_get(), st_length_get(), and st_length_int().

Referenced by smtp_reply().

int_t cache_silent_add (stringer_t * key, stringer_t * object, time_t expiration)

Add a new key/value pair to the memcached server, but suppress any error messages.

Parameters:

key a managed string with the name of the key to be added to the cache.

object a managed string containing the initial value of the new key.

expiration an expiration time, in seconds, for the newly cached data.

Definition at line 307 of file cache.c.

References memcached_add_d, PL_RESERVED, pool_get_obj(), pool_pull(), pool_release(), st_char_get(), and st_length_get().

Referenced by lock_get().

bool_t cache_start (void)

Definition at line 51 of file cache.c.

References magma_t::cache, magma_t::iface, log_critical, magma, MAGMA_CACHE_INSTANCES, memcached_behavior_set_d, memcached_create_d, memcached_free_d, memcached_server_add_with_weight_d, memcached_strerror_d, pool_alloc(), and pool_set_obj().

Referenced by process_start().

void cache_stop (void)

Destroy the memcached client pool and all active connections.

Returns:

This function returns no value.

Definition at line 110 of file cache.c.

References magma_t::cache, magma_t::iface, magma, memcached_free_d, pool_free(), and pool_get_obj().

Referenced by process_stop().

bool_t lib_load_cache (void)

Initialize the memcached library and bind dynamically to the exported functions that are required.

Returns:

true on success or false on failure.

Definition at line 22 of file cache.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

const char* lib_version_cache (void)

Return the version string of the memcached library.

Returns:

a pointer to a character string containing the memcached library version information.

Definition at line 43 of file cache.c.

References memcached_lib_version_d.

Referenced by lib_load().

Variable Documentation

pool_t* cache_pool = NULL

Definition at line 15 of file cache.c.

magma/engine/config/cache/cache.h File Reference

Types and functions for creating and accessing the cache structures.

Data Structures

- struct **cache_keys_t**
- struct **cache_t**

Functions

- void **cache_free** (void)
 - *Free magma's cache server configuration options.* **bool_t cache_validate** (void)
 - *Validate all of the configured magma cache server instances.* void **cache_output_settings** (void)
 - *Log the full contents of the magma cache configuration settings.* **cache_t * cache_alloc** (uint32_t **number**)
 - *Allocate a new cache server configuration entry and initialize it to its default values.* **bool_t cache_config** (**stringer_t** ***name**, **stringer_t** ***value**)
 - *Set the value of a cache server configuration entry by name.* **bool_t cache_set_value** (**cache_keys_t** ***setting**, **cache_t** ***cache**, **stringer_t** ***value**)
 - *Set the value of a key for a specified cache server configuration entry.* void **cache_output_help** (void)
- Log all cache key information to be returned via "magma -h".*
-

Detailed Description

Types and functions for creating and accessing the cache structures.

Definition in file **cache.h**.

Function Documentation

cache_t* cache_alloc (uint32_t *number*)

Allocate a new cache server configuration entry and initialize it to its default values.

Parameters:

number the index of the cache server instance in the global cache server array.

Returns:

NULL on failure, or a pointer to the requested cache server configuration entry on success.

Definition at line 69 of file cache.c.

References magma_t::cache, cache_keys, cache_set_value(), magma_t::iface, log_critical, magma, and mm_alloc().

Referenced by cache_config().

bool_t cache_config (stringer_t * *name*, stringer_t * *value*)

Set the value of a cache server configuration entry by name.

Note:

This function is designed to operate on lines from a configuration source, like a file or database. When a named server is referenced for the first time, a new cache server configuration instance will be allocated.

Parameters:

name a managed string containing the human-readable cache server configuration parameter to be set.
value a managed string containing the new value of the cache server config key.

Returns:

true if the value was successfully set, or false on failure.

Definition at line 430 of file cache.c.

References bracket_extract_pl(), cache_alloc(), cache_keys, cache_set_value(), CONSTANT, log_critical, MAGMA_CACHE_INSTANCES, NULLER, pl_char_get(), pl_empty(), pl_length_get(), PLACER, placer_t, st_char_get(), st_cmp_ci_eq(), st_cmp_ci_starts(), st_length_get(), st_length_int(), and uint32_conv_bl().

Referenced by config_load_cmdline_settings(), config_load_database_settings(), and config_load_file_settings().

void cache_free (void)

Free magma's cache server configuration options.

Note:

This function assumes all worker threads have finished, and should only be called during the normal shutdown process.

Parameters:

This function returns no value.

Definition at line 22 of file cache.c.

References magma_t::cache, cache_keys, magma_t::iface, log_pedantic, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_ENUM, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_CACHE_INSTANCES, mm_free(), cache_keys_t::norm, ns_empty(), ns_free(), cache_keys_t::offset, st_empty(), st_free(), type(), and multi_t::type.

Referenced by config_free().

void cache_output_help (void)

Log all cache key information to be returned via "magma -h".

Returns:

This function returns no value.

Definition at line 489 of file cache.c.

References multi_t::bl, cache_keys, config_output_value_generic(), log_info, log_options, M_LOG_FILE_DISABLE, M_LOG_FUNCTION_DISABLE, M_LOG_LINE_DISABLE, M_LOG_LINE_FEED_DISABLE, M_LOG_STACK_TRACE_DISABLE, M_LOG_TIME_DISABLE, cache_keys_t::norm, cache_keys_t::required, and multi_t::val.

Referenced by config_output_help().

void cache_output_settings (void)

Log the full contents of the magma cache configuration settings.

Note:

This logs all the details of each memcached instance configured to work with magma.

Returns:

This function returns no value.

Definition at line 212 of file cache.c.

References magma_t::cache, cache_keys, magma_t::iface, log_info, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_CACHE_INSTANCES, cache_keys_t::norm, ns_empty(), cache_keys_t::offset, st_char_get(), st_empty(), st_length_int(), multi_t::type, and type().

Referenced by config_output_settings().

bool_t cache_set_value (cache_keys_t * setting, cache_t * cache, stringer_t * value)

Set the value of a key for a specified cache server configuration entry.

Note:

This function will allocate space for a copy of the key value, and return an error if it is not able to convert it to the proper key data type.

Parameters:

setting a pointer to the cache server key to be set.

cache a pointer to the cache server configuration entry to be modified.

value a managed string containing the new value of the key.

Returns:

true if the value was successfully set, or false on failure.

Definition at line 287 of file cache.c.

References multi_t::binary, CONSTANT, CONTIGUOUS, HEAP, multi_t::i32, multi_t::i64, multi_t::i8, int32_conv_st(), int64_conv_st(), int8_conv_st(), log_critical, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, MANAGED_T, cache_keys_t::name, cache_keys_t::norm, multi_t::ns, ns_dupe(), ns_empty(), ns_free(), ns_import(), cache_keys_t::offset, multi_t::st, st_char_get(), st_cmp_ci_eq(), st_dupe_opts(), st_empty(), st_free(), st_length_get(), type(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, uint16_conv_st(), uint32_conv_st(), uint64_conv_st(), uint8_conv_st(), and multi_t::val.

Referenced by cache_alloc(), and cache_config().

bool_t cache_validate (void)

Validate all of the configured magma cache server instances.

Note:

This makes set that all required config keys have been set, and that at least one host has been configured.

Returns:

true if all cache servers have been validated, or false on failure.

Definition at line 98 of file cache.c.

References magma_t::cache, cache_keys, multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, magma_t::iface, log_critical, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_CACHE_INSTANCES, cache_keys_t::norm, ns_empty(), cache_keys_t::offset, st_empty(), multi_t::type, type(), multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by config_validate_settings().

magma/engine/config/cache/keys.h File Reference

A collection of keys that define the configuration of cache instances.

Variables

- `cache_keys_t cache_keys []`

Detailed Description

A collection of keys that define the configuration of cache instances.

Definition in file **keys.h**.

Variable Documentation

`cache_keys_t cache_keys[]`

```
Initial value: {
    {
        .offset = offsetof (cache_t, name),
        .norm.type = M_TYPE_NULLER,
        .norm.val.st = NULL,
        .name = ".name",
        .description = "The host name or address of the cache instance.",
        .required = true
    },
    {
        .offset = offsetof (cache_t, port),
        .norm.type = M_TYPE_UINT32,
        .norm.val.u32 = 11211,
        .name = ".port",
        .description = "The port used by the cache instance.",
        .required = false
    },
    {
        .offset = offsetof (cache_t, weight),
        .norm.type = M_TYPE_UINT32,

        .norm.val.u32 = 1024,
        .name = ".weight",
        .description = "The relative weight of the cache instance.",
        .required = false
    }
}
```

Definition at line 16 of file `keys.h`.

Referenced by `cache_alloc()`, `cache_config()`, `cache_free()`, `cache_output_help()`, `cache_output_settings()`, and `cache_validate()`.

magma/engine/config/global/keys.h File Reference

A collection of keys that define access rules, default values and other parameters needed when loading the global configuration.

Variables

- `magma_keys_t magma_keys []`
-

Detailed Description

A collection of keys that define access rules, default values and other parameters needed when loading the global configuration.

Definition in file **keys.h**.

Variable Documentation

magma_keys_t magma_keys[]

Definition at line 44 of file keys.h.

Referenced by `config_free()`, `config_key_lookup()`, `config_load_defaults()`, `config_output_help()`, `config_output_settings()`, and `config_validate_settings()`.

magma/engine/config/relay/keys.h File Reference

A collection of keys that define access rules, default values and other parameters needed to configure relay instances.

Variables

- `relay_keys_t relay_keys []`

Detailed Description

A collection of keys that define access rules, default values and other parameters needed to configure relay instances.

Definition in file `keys.h`.

Variable Documentation

`relay_keys_t relay_keys[]`

```
Initial value: {
    {
        .offset = offsetof (relay_t, name),
        .norm.type = M_TYPE_NULLER,
        .norm.val.st = NULL,
        .name = ".name",
        .description = "The host name or address of the mail relay server.",
        .required = true
    },
    {
        .offset = offsetof (relay_t, port),
        .norm.type = M_TYPE_UINT32,
        .norm.val.u32 = 25,
        .name = ".port",
        .description = "The port used by the mail relay server.",
        .required = false
    },
    {
        .offset = offsetof (relay_t, secure),
        .norm.type = M_TYPE_BOOLEAN,
        .norm.val.binary = false,
        .name = ".secure",
        .description = "Determines whether connections should be made using SSL.",
        .required = false
    }
}
```

Definition at line 16 of file `keys.h`.

Referenced by `relay_alloc()`, `relay_config()`, `relay_free()`, `relay_output_help()`, `relay_output_settings()`, and `relay_validate()`.

magma/engine/config/servers/keys.h File Reference

A collection of keys that define access rules, default values and other parameters needed when configuring server instances.

Variables

- `server_keys_t server_keys []`
-

Detailed Description

A collection of keys that define access rules, default values and other parameters needed when configuring server instances.

Definition in file **keys.h**.

Variable Documentation

server_keys_t server_keys[]

Definition at line 16 of file keys.h.

Referenced by `servers_alloc()`, `servers_config()`, `servers_free()`, `servers_output_help()`, `servers_output_settings()`, and `servers_validate()`.

magma/engine/config/config.h File Reference

The entry point for all configuration modules used by magma.

```
#include "cache/cache.h"
#include "relay/relay.h"
#include "servers/servers.h"
#include "global/global.h"
```

Defines

- #define **MAGMA_HOSTNAME_MAX** _POSIX_HOST_NAME_MAX
 - #define **MAGMA_FILEPATH_MAX** PATH_MAX
 - #define **MAGMA_FILENAME_MAX** NAME_MAX
 - #define **MAGMA_THREAD_STACK_SIZE** 1048576
 - #define **MAGMA_THREAD_BUFFER_SIZE** 1024
 - #define **MAGMA_WORKER_THREAD_LIMIT** 16384
 - #define **MAGMA_CRYPTOGRAPHY_SEED_SIZE** 64
 - #define **MAGMA_LOGS** "logs/"
 - #define **MAGMA_RESOURCE_FONTS** "resources/**fonts**/"
 - #define **MAGMA_RESOURCE_PAGES** "resources/**pages**/"
 - #define **MAGMA_RESOURCE_VIRUS** "resources/virus/"
 - #define **MAGMA_RESOURCE_LOCATION** "resources/location/"
 - #define **MAGMA_RESOURCE_TEMPLATES** "resources/**templates**/"
 - #define **MAGMA_LOCATION_CACHE** CONSTANT("disable")
 - #define **MAGMA_CACHE_INSTANCES** 8
 - #define **MAGMA_CACHE_SERVER_RETRY** 600
 - #define **MAGMA_CACHE_SOCKET_TIMEOUT** 10
 - #define **MAGMA_BLACKLIST_INSTANCES** 6
 - #define **MAGMA_RELAY_INSTANCES** 8
 - #define **MAGMA_SERVER_INSTANCES** 32
 - #define **MAGMA_CONNECTION_BUFFER_SIZE** 8192
 - #define **MAGMA_SMTP_MAX_HELO_SIZE** MAGMA_HOSTNAME_MAX
 - #define **MAGMA_SMTP_MAX_ADDRESS_SIZE** 256
 - #define **MAGMA_SMTP_RECIPIENT_LIMIT** 256
 - #define **MAGMA_SMTP_RELAY_LIMIT** 256
 - #define **MAGMA_SMTP_LINE_WRAP_LENGTH** 80
 - #define **MAGMA_SMTP_MAX_MESSAGE_SIZE** 1073741824
 - #define **CONFIG_CHECK_EXISTS**(option, ptype)
 - #define **CONFIG_CHECK_FILE_READABLE**(x) CONFIG_CHECK_EXISTS(x,"Filename")
 - #define **CONFIG_CHECK_DIR_READABLE**(x) CONFIG_CHECK_EXISTS(x,"Directory")
 - #define **CONFIG_CHECK_READWRITE**(option, ptype)
 - #define **CONFIG_CHECK_FILE_READWRITE**(x) CONFIG_CHECK_READWRITE(x,"Filename")
 - #define **CONFIG_CHECK_DIR_READWRITE**(x) CONFIG_CHECK_READWRITE(x,"Directory")
-

Detailed Description

The entry point for all configuration modules used by magma.

Definition in file **config.h**.

Define Documentation

#define CONFIG_CHECK_DIR_READABLE(x) CONFIG_CHECK_EXISTS(x,"Directory")

Definition at line 96 of file config.h.

Referenced by config_validate_settings().

#define CONFIG_CHECK_DIR_READWRITE(x) CONFIG_CHECK_READWRITE(x,"Directory")

Definition at line 102 of file config.h.

Referenced by config_validate_settings().

#define CONFIG_CHECK_EXISTS(option, ptype)

```
Value:if (option && !file_accessible(option)) { \
    log_critical(#ptype " specified in " #option " is not accessible: { path = %s,\
error = %s }", option, strerror_r(errno, bufptr, buflen)); \
    result = false; \
}
```

Definition at line 91 of file config.h.

#define CONFIG_CHECK_FILE_READABLE(x) CONFIG_CHECK_EXISTS(x,"Filename")

Definition at line 95 of file config.h.

Referenced by config_validate_settings().

#define CONFIG_CHECK_FILE_READWRITE(x) CONFIG_CHECK_READWRITE(x,"Filename")

Definition at line 101 of file config.h.

#define CONFIG_CHECK_READWRITE(option, ptype)

```
Value:if (option && !file_readwritable(option)) { \
    log_critical(#ptype " specified in " #option " is not accessible for reading and\
writing: { path = %s, error = %s }", option, strerror_r(errno, bufptr, buflen)); \
    result = false; \
}
```

Definition at line 97 of file config.h.

#define MAGMA_BLACKLIST_INSTANCES 6

Definition at line 59 of file config.h.

Referenced by config_value_set().

#define MAGMA_CACHE_INSTANCES 8

Definition at line 50 of file config.h.

Referenced by cache_config(), cache_free(), cache_output_settings(), cache_start(), and cache_validate().

#define MAGMA_CACHE_SERVER_RETRY 600

Definition at line 53 of file config.h.

#define MAGMA_CACHE_SOCKET_TIMEOUT 10

Definition at line 56 of file config.h.

#define MAGMA_CONNECTION_BUFFER_SIZE 8192

Definition at line 68 of file config.h.

#define MAGMA_CRYPTOGRAPHY_SEED_SIZE 64

Definition at line 36 of file config.h.

#define MAGMA_FILENAME_MAX NAME_MAX

Definition at line 24 of file config.h.

#define MAGMA_FILEPATH_MAX PATH_MAX

Definition at line 21 of file config.h.

Referenced by args_parse(), process_kill(), and tank_start().

#define MAGMA_HOSTNAME_MAX _POSIX_HOST_NAME_MAX

Definition at line 18 of file config.h.

Referenced by process_start(), and warehouse_fetch_domains().

#define MAGMA_LOCATION_CACHE CONSTANT("disable")

Definition at line 47 of file config.h.

#define MAGMA_LOGS "logs/"

Definition at line 39 of file config.h.

#define MAGMA_RELAY_INSTANCES 8

Definition at line 62 of file config.h.

Referenced by relay_config(), relay_counter(), relay_free(), relay_output_settings(), relay_validate(), and smtp_client_connect().

#define MAGMA_RESOURCE_FONTS "resources/fonts/"

Definition at line 40 of file config.h.

#define MAGMA_RESOURCE_LOCATION "resources/location/"

Definition at line 43 of file config.h.

#define MAGMA_RESOURCE_PAGES "resources/pages/"

Definition at line 41 of file config.h.

#define MAGMA_RESOURCE_TEMPLATES "resources/templates/"

Definition at line 44 of file config.h.

#define MAGMA_RESOURCE_VIRUS "resources/virus/"

Definition at line 42 of file config.h.

#define MAGMA_SERVER_INSTANCES 32

Definition at line 65 of file config.h.

Referenced by servers_config(), servers_encryption_start(), servers_encryption_stop(), servers_free(), servers_get_by_socket(), servers_get_count_using_port(), servers_network_start(), servers_network_stop(), servers_output_settings(), and servers_validate().

#define MAGMA_SMTP_LINE_WRAP_LENGTH 80

Definition at line 85 of file config.h.

#define MAGMA_SMTP_MAX_ADDRESS_SIZE 256

Definition at line 75 of file config.h.

#define MAGMA_SMTP_MAX_HELO_SIZE MAGMA_HOSTNAME_MAX

Definition at line 72 of file config.h.

#define MAGMA_SMTP_MAX_MESSAGE_SIZE 1073741824

Definition at line 88 of file config.h.

#define MAGMA_SMTP_RECIPIENT_LIMIT 256

Definition at line 78 of file config.h.

#define MAGMA_SMTP_RELAY_LIMIT 256

Definition at line 80 of file config.h.

#define MAGMA_THREAD_BUFFER_SIZE 1024

Definition at line 30 of file config.h.

#define MAGMA_THREAD_STACK_SIZE 1048576

Definition at line 27 of file config.h.

#define MAGMA_WORKER_THREAD_LIMIT 16384

Definition at line 33 of file config.h.

magma/objects/config/config.h File Reference

The user configuration interface.

Enumerations

- enum { **USER_CONF_STATUS_NONE** = 0, **USER_CONF_STATUS_CRITICAL** = 1 }

Functions

- struct **__attribute__((__packed__))**
- user_config_t** * **user_config_alloc** (uint64_t usernum)
- config.c* **user_config_t** * **user_config_create** (uint64_t usernum)
- Create a new user config collection object for the specified user and populate it with config entries from the database. **int_t** **user_config_edit** (**user_config_t** *collection, **stringer_t** *key, **stringer_t** *value)
- Change the value of, or delete a key from a user's config collection. **user_config_entry_t** * **user_config_entry_alloc** (**stringer_t** *key, **stringer_t** *value, uint64_t flags)
- Create and initialize new user config entry object. void **user_config_entry_free** (**user_config_entry_t** *entry)
- Destroy a user config entry. void **user_config_free** (**user_config_t** *collection)
- Free a user config collection object. **int_t** **user_config_update** (**user_config_t** *collection)
- Update a collection of user config options from the database, if necessary. **int_t** **user_config_delete** (uint64_t usernum, **stringer_t** *key)
- datatier.c* **bool_t** **user_config_fetch** (**user_config_t** *collection)
- Fetch a user's config collection from the database. **int_t** **user_config_upsert** (uint64_t usernum, **stringer_t** *key, **stringer_t** *value, uint64_t flags)

Update a user config entry in the database, or insert it if it does not already exist.
Variables

- user_config_entry_t**
- user_config_t**

Detailed Description

The user configuration interface.

Definition in file **config.h**.

Enumeration Type Documentation

anonymous enum

Enumerator:

USER_CONF_STATUS_NONE
USER_CONF_STATUS_CRITICAL

Definition at line 22 of file config.h.

Function Documentation

struct __attribute__((__packed__)) [read]

Definition at line 27 of file config.h.

References `__attribute__::serial`.

user_config_t* user_config_alloc (uint64_t *usernum*)

config.c

config.c

Parameters:

usernum the numerical user id for the requested user.

Returns:

NULL on failure or a pointer to the newly allocated user config collection object on success.

Definition at line 49 of file config.c.

References `align()`, `inx_alloc()`, `log_pedantic`, `M_INX_TREE`, `mm_alloc()`, `mm_free()`, `user_config_entry_free()`, and `user_config_t`.

Referenced by `user_config_create()`.

user_config_t* user_config_create (uint64_t *usernum*)

Create a new user config collection object for the specified user and populate it with config entries from the database.

Parameters:

usernum the numerical user id for the user to be queried.

Returns:

NULL on failure, or a pointer to the newly created user config collection object on success.

Definition at line 111 of file config.c.

References `log_pedantic`, `OBJECT_CONFIG`, `serial_get()`, `user_config_alloc()`, `user_config_fetch()`, `user_config_free()`, and `user_config_t`.

Referenced by `portal_endpoint_config_edit()`, and `portal_endpoint_config_load()`.

int_t user_config_delete (uint64_t *usernum*, stringer_t * *key*)

datatier.c

datatier.c

Parameters:

usernum the numerical userid of the user to whom the requested config key belongs.

key a managed string containing the name of the config key to be deleted.

Returns:

1 if the config option was successfully deleted, 0 if the config key couldn't be found in the database, or -1 on general failure.

Definition at line 21 of file `datatier.c`.

References `log_check`, `log_pedantic`, `mm_wipe()`, `st_char_get()`, `st_length_get()`, `st_length_int()`, and `stmt_exec_affected()`.

Referenced by `user_config_edit()`.

`int_t user_config_edit (user_config_t * collection, stringer_t * key, stringer_t * value)`

Change the value of, or delete a key from a user's config collection.

Parameters:

collection a pointer to a collection of user's config entries.

key a pointer to a managed string containing the name of the user config key to be modified.

value a pointer to a managed string containing the name of the new value of the key, or NULL if it is to be deleted.

Returns:

-1 on error or 1 on success.

Definition at line 165 of file `config.c`.

References `inx_delete()`, `inx_find()`, `inx_replace()`, `log_pedantic`, `M_TYPE_STRINGER`, `OBJECT_CONFIG`, `serial_increment()`, `multi_t::st`, `st_cmp_cs_eq()`, `st_empty()`, `user_config_delete()`, `user_config_entry_alloc()`, `user_config_entry_free()`, `user_config_entry_t`, `user_config_upsert()`, and `multi_t::val`.

Referenced by `portal_endpoint_config_edit()`.

`user_config_entry_t* user_config_entry_alloc (stringer_t * key, stringer_t * value, uint64_t flags)`

Create and initialize new user config entry object.

Parameters:

key a managed string containing the key of the config entry object.

value a managed string containing the value of the config entry object.

flags a bitmask reflecting the flags of the config entry object.

Returns:

NULL on failure, or a pointer to the newly allocated and initialized user config entry object on success.

Definition at line 75 of file `config.c`.

References `align()`, `FOREIGNDATA`, `JOINTED`, `log_pedantic`, `mm_alloc()`, `mm_copy()`, `PLACER_T`, `placer_t`, `st_data_get()`, `st_length_get()`, `STACK`, and `user_config_entry_t`.

Referenced by `user_config_edit()`, and `user_config_fetch()`.

`void user_config_entry_free (user_config_entry_t * entry)`

Destroy a user config entry.

Parameters:

entry a pointer to the user config entry object to be freed.

Returns:

This function returns no value.

Definition at line 20 of file config.c.

References mm_free().

Referenced by user_config_alloc(), user_config_edit(), and user_config_fetch().

bool_t user_config_fetch (user_config_t * collection)

Fetch a user's config collection from the database.

Parameters:

collection a pointer to the specified user config collection object (with usernum field set by the caller) to be populated from the database.

Returns:

true on success or false on failure.

Definition at line 101 of file datatier.c.

References inx_insert(), log_info, log_pedantic, M_TYPE_STRINGER, mm_wipe(), PLACER, res_field_block(), res_field_length(), res_field_uint64(), res_row_next(), res_table_free(), multi_t::st, stmt_get_result(), user_config_entry_alloc(), user_config_entry_free(), user_config_entry_t, and multi_t::val.

Referenced by user_config_create(), and user_config_update().

void user_config_free (user_config_t * collection)

Free a user config collection object.

Parameters:

collection a pointer to the user config collection object to be freed.

Returns:

This function returns no value.

Definition at line 34 of file config.c.

References inx_cleanup(), and mm_free().

Referenced by portal_endpoint_config_edit(), portal_endpoint_config_load(), and user_config_create().

int_t user_config_update (user_config_t * collection)

Update a collection of user config options from the database, if necessary.

Note:

This function is not currently being used anywhere. If this function returns -1, the config entries are left in an undefined state and should not be relied upon.

Parameters:

collection a pointer to a collection of user config entries to be checked for updates.

Returns:

1 if the collection is up-to-date, 0 if it was refreshed to update changes, or -1 if the refresh operation failed.

Definition at line 137 of file config.c.

References `inx_truncate()`, `OBJECT_CONFIG`, `serial_get()`, and `user_config_fetch()`.

int_t user_config_upsert (uint64_t *usernum*, stringer_t * *key*, stringer_t * *value*, uint64_t *flags*)

Update a user config entry in the database, or insert it if it does not already exist.

Parameters:

usernum the numerical id of the user to whom the specified config entry belongs.

key a pointer to a managed string containing the name of the config entry to be updated or inserted.

value a pointer to a managed string containing the value of the specified config entry key.

flags a bitmask of flags for the config entry (`USER_CONF_STATUS_CRITICAL` is supported).

Returns:

2 if the config entry was updated, 1 if a new config key was inserted into the database, 0 if no update was necessary, or -1 on general failure.

Definition at line 57 of file `datatier.c`.

References `log_check`, `log_pedantic`, `mm_wipe()`, `st_char_get()`, `st_length_get()`, `st_length_int()`, and `stmt_exec_affected()`.

Referenced by `user_config_edit()`.

Variable Documentation

user_config_entry_t

Definition at line 19 of file `config.h`.

Referenced by `portal_config_collection()`, `user_config_edit()`, `user_config_entry_alloc()`, and `user_config_fetch()`.

user_config_t

Definition at line 30 of file `config.h`.

Referenced by `portal_endpoint_config_edit()`, `portal_endpoint_config_load()`, `user_config_alloc()`, and `user_config_create()`.

magma/engine/config/global/datatier.c File Reference

Database interface for the engine module.

```
#include "magma.h"
```

Functions

- `uint64_t config_fetch_host_number` (void)
- *Retrieve this computer's host number from the database.* `table_t * config_fetch_settings` (void)

Retrieve the entire collection of configuration key/value pairs from the database.

Detailed Description

Database interface for the engine module.

Definition in file `datatier.c`.

Function Documentation

`uint64_t config_fetch_host_number` (void)

Retrieve this computer's host number from the database.

`datatier.c`

Returns:

this host's numerical identifier, or 0 on failure.

Definition at line 19 of file `datatier.c`.

References `magma_t::host`, `log_error`, `magma`, `mm_wipe()`, `magma_t::name`, `ns_length_get()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, and `stmt_get_result()`.

Referenced by `config_load_database_settings()`.

`table_t* config_fetch_settings` (void)

Retrieve the entire collection of configuration key/value pairs from the database.

Parameters:

none This function accepts no parameters.

Returns:

NULL on failure, or a database table of configuration key/value pairs on success.

Definition at line 45 of file `datatier.c`.

References `magma_t::host`, `magma`, `mm_wipe()`, `magma_t::name`, `ns_length_get()`, and `stmt_get_result()`.

Referenced by `config_load_database_settings()`.

magma/objects/config/datatier.c File Reference

The database routines for the user configuration interface.

```
#include "magma.h"
```

Functions

- **int_t user_config_delete** (uint64_t usernum, **stringer_t** *key)
- *Delete a user config entry from the database by key.* **int_t user_config_upsert** (uint64_t usernum, **stringer_t** *key, **stringer_t** *value, uint64_t flags)
- *Update a user config entry in the database, or insert it if it does not already exist.* **bool_t user_config_fetch** (**user_config_t** *collection)

Fetch a user's config collection from the database.

Detailed Description

The database routines for the user configuration interface.

Definition in file **datatier.c**.

Function Documentation

int_t user_config_delete (uint64_t *usernum*, **stringer_t** * *key*)

Delete a user config entry from the database by key.

datatier.c

Parameters:

usernum the numerical userid of the user to whom the requested config key belongs.

key a managed string containing the name of the config key to be deleted.

Returns:

1 if the config option was successfully deleted, 0 if the config key couldn't be found in the database, or -1 on general failure.

Definition at line 21 of file datatier.c.

References `log_check`, `log_pedantic`, `mm_wipe()`, `st_char_get()`, `st_length_get()`, `st_length_int()`, and `stmt_exec_affected()`.

Referenced by `user_config_edit()`.

bool_t user_config_fetch (**user_config_t** * *collection*)

Fetch a user's config collection from the database.

Parameters:

collection a pointer to the specified user config collection object (with `usernum` field set by the caller) to be populated from the database.

Returns:

true on success or false on failure.

Definition at line 101 of file `datatier.c`.

References `inx_insert()`, `log_info`, `log_pedantic`, `M_TYPE_STRINGER`, `mm_wipe()`, `PLACER`, `res_field_block()`, `res_field_length()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `multi_t::st`, `stmt_get_result()`, `user_config_entry_alloc()`, `user_config_entry_free()`, `user_config_entry_t`, and `multi_t::val`.

Referenced by `user_config_create()`, and `user_config_update()`.

`int_t user_config_upsert(uint64_t usernum, stringer_t * key, stringer_t * value, uint64_t flags)`

Update a user config entry in the database, or insert it if it does not already exist.

Parameters:

usernum the numerical id of the user to whom the specified config entry belongs.

key a pointer to a managed string containing the name of the config entry to be updated or inserted.

value a pointer to a managed string containing the value of the specified config entry key.

flags a bitmask of flags for the config entry (`USER_CONF_STATUS_CRITICAL` is supported).

Returns:

2 if the config entry was updated, 1 if a new config key was inserted into the database, 0 if no update was necessary, or -1 on general failure.

Definition at line 57 of file `datatier.c`.

References `log_check`, `log_pedantic`, `mm_wipe()`, `st_char_get()`, `st_length_get()`, `st_length_int()`, and `stmt_exec_affected()`.

Referenced by `user_config_edit()`.

magma/objects/contacts/datatier.c File Reference

Functions for handling user contacts in the database.

```
#include "magma.h"
```

Functions

- **int_t contact_update_stamp** (uint64_t contactnum, uint64_t usernum, uint64_t foldernum)
- *Touch the last updated time stamp of a contact entry in the database.* **int_t contact_delete** (uint64_t contactnum, uint64_t usernum, uint64_t foldernum)
- *Delete a user's contact entry and its associated details from the database.* **int_t contact_update** (uint64_t contactnum, uint64_t usernum, uint64_t cur_folder, uint64_t target_folder, **stringer_t** *name)
- *Update a user contact entry in the database.* **uint64_t contact_insert** (uint64_t usernum, uint64_t foldernum, **stringer_t** *name)
- *Insert a new contact entry into the database.* **int_t contact_detail_delete** (uint64_t contactnum, **stringer_t** *key)
- *Delete the specified contact detail of a contact entry from the database.* **int_t contact_detail_upsert** (uint64_t contactnum, **stringer_t** *key, **stringer_t** *value, uint64_t flags)
- *Update a specified contact detail in the database, or insert it if it does not exist.* **int_t contact_details_fetch** (**contact_t** *contact)
- *Populate a contact entry with its details from the database.* **int_t contacts_fetch** (uint64_t usernum, **contact_folder_t** *folder)

Retrieve all of a user's contact entries in a specified contacts folder from the database.

Detailed Description

Functions for handling user contacts in the database.

Definition in file **datatier.c**.

Function Documentation

int_t contact_delete (uint64_t *contactnum*, uint64_t *usernum*, uint64_t *foldernum*)

Delete a user's contact entry and its associated details from the database.

datatier.c

Parameters:

contactnum the numerical id of the contact entry to be deleted.

usernum the numerical id of the user to whom the specified contact entry belongs.

foldernum the numerical id of the parent contact folder containing the contact entry.

Returns:

1 if the specified contact was deleted successfully, 0 if no matching contact was found in the database, or -1 on general failure.

Definition at line 66 of file datatier.c.

References `log_check`, `log_pedantic`, `mm_wipe()`, and `stmt_exec_affected()`.

Referenced by `contact_remove()`, and `portal_endpoint_contacts_remove()`.

int_t contact_detail_delete (uint64_t *contactnum*, stringer_t * *key*)

Delete the specified contact detail of a contact entry from the database.

Parameters:

contactnum the numerical id of the contact entry to have the specified detail removed.

key a managed string containing the name of the contact detail to be removed from the entry.

Returns:

-1 on error, 0 if no matching detail was found in the database, or 1 if the delete operation was successful.

Definition at line 209 of file *datatier.c*.

References *log_check*, *log_pedantic*, *mm_wipe()*, *st_char_get()*, *st_length_get()*, *st_length_int()*, and *stmt_exec_affected()*.

Referenced by *contact_edit()*.

int_t contact_detail_upsert (uint64_t *contactnum*, stringer_t * *key*, stringer_t * *value*, uint64_t *flags*)

Update a specified contact detail in the database, or insert it if it does not exist.

Parameters:

contactnum the numerical id of the contact entry to be modified.

key a managed string containing the name of the contact detail to be updated.

value a managed string containing the new value of the specified contact detail.

flags a bitmask of flags to be associated with the specified contact entry detail.

Returns:

-1 on error, 0 if no update was necessary, 1 if a new contact detail was inserted into the database, or 2 if the specified contact detail was updated successfully.

Definition at line 246 of file *datatier.c*.

References *log_check*, *log_pedantic*, *mm_wipe()*, *st_char_get()*, *st_length_get()*, *st_length_int()*, and *stmt_exec_affected()*.

Referenced by *contact_edit()*.

int_t contact_details_fetch (contact_t * *contact*)

Populate a contact entry with its details from the database.

Parameters:

contact a pointer to the contact entry object that will be updated.

Returns:

-1 on failure or 1 on success.

Definition at line 290 of file *datatier.c*.

References *contact_detail_alloc()*, *contact_detail_free()*, *contact_detail_t*, *inx_insert()*, *log_info*, *log_pedantic*, *M_TYPE_STRINGER*, *mm_wipe()*, *PLACER*, *res_field_block()*, *res_field_length()*, *res_field_uint64()*, *res_row_next()*, *res_table_free()*, *multi_t::st*, *stmt_get_result()*, and *multi_t::val*.

Referenced by *contacts_fetch()*.

uint64_t contact_insert (uint64_t *usernum*, uint64_t *foldernum*, stringer_t * *name*)

Insert a new contact entry into the database.

Parameters:

usernum the numerical id of the user to whom the contact entry belongs.
foldernum the numerical id of the parent folder to contain the contact entry.
name a pointer to a managed string containing the name of the new contact entry.

Returns:

-1 on failure, 0 if no item was inserted into the database, or the id of the newly inserted contact entry in the database on success.

Definition at line 177 of file *datatier.c*.

References *mm_wipe()*, *st_char_get()*, *st_length_get()*, and *stmt_insert()*.

Referenced by *contact_create()*.

int_t contact_update (uint64_t *contactnum*, uint64_t *usernum*, uint64_t *cur_folder*, uint64_t *target_folder*, stringer_t * *name*)

Update a user contact entry in the database.

Parameters:

contactnum the numerical id of the contact entry to be modified.
usernum the numerical id of the user to whom the specified contact entry belongs.
cur_folder the numerical id of the parent contact containing the specified contact entry.
target_folder if not 0, sets the new parent contact folder to which the specified contact entry will belong.
name if not NULL, sets the new name of the specified contact entry.

Returns:

-1 on error, 0 if the specified contact entry was not found in the database, or 1 if the contact entry was successfully updated.

Definition at line 111 of file *datatier.c*.

References *ISNULL*, *log_check*, *log_pedantic*, *mm_wipe()*, *st_char_get()*, *st_empty()*, *st_length_get()*, and *stmt_exec_affected()*.

Referenced by *contact_edit()*, and *contact_move()*.

int_t contact_update_stamp (uint64_t *contactnum*, uint64_t *usernum*, uint64_t *foldernum*)

Touch the last updated time stamp of a contact entry in the database.

Parameters:

contactnum the numerical id of the contact entry to have its time stamp updated.
usernum the numerical id of the user to whom the contact entry belongs.
foldernum the numerical id of the parent folder containing the contact entry.

Returns:

1 if the contact time stamp was updated successfully, 0 if the specified contact was not found, or -1 on general failure.

Definition at line 22 of file *datatier.c*.

References log_check, log_pedantic, mm_wipe(), and stmt_exec_affected().

Referenced by contact_edit().

int_t contacts_fetch (uint64_t *usernum*, contact_folder_t * *folder*)

Retrieve all of a user's contact entries in a specified contacts folder from the database.

Parameters:

usernum the numerical id of the user whose contacts will be retrieved.

folder a pointer to the contact folder which will have its contents listed.

Returns:

-1 on failure or 1 on success.

LOW: Should we bother with error checking?

Definition at line 344 of file datatier.c.

References contact_alloc(), contact_details_fetch(), contact_free(), contact_t, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_insert(), log_info, log_pedantic, M_TYPE_UINT64, mm_wipe(), PLACER, res_field_block(), res_field_length(), res_field_uint64(), res_row_next(), res_table_free(), stmt_get_result(), multi_t::u64, and multi_t::val.

Referenced by contacts_update().

magma/objects/folders/datatier.c File Reference

Folder data functions.

```
#include "magma.h"
```

Functions

- **inx_t * magma_folder_fetch** (uint64_t usernum, uint_t type)
- *Fetch all of a user's folders of a specified type from the database.* uint64_t **magma_folder_insert** (uint64_t usernum, **stringer_t** *name, uint64_t parent, uint32_t order, uint_t type)
- *Insert a new folder into the database.* **bool_t magma_folder_delete** (uint64_t usernum, uint64_t foldernum, uint_t type)
- *Delete a folder from the database.* **bool_t magma_folder_rename** (uint64_t usernum, uint64_t foldernum, uint_t type, **stringer_t** *rename)

Rename a folder in the database.

Detailed Description

Folder data functions.

Definition in file **datatier.c**.

Function Documentation

bool_t magma_folder_delete (uint64_t *usernum*, uint64_t *foldernum*, uint_t *type*)

Delete a folder from the database.

datatier.c

Parameters:

usernum the numerical id of the user requesting the folder removal.

foldernum the numerical id of the folder to be removed.

type the type of the folder to be deleted (can be M_FOLDER_MESSAGES or M_FOLDER_CONTACTS).

Returns:

true on success or false on failure.

Definition at line 141 of file datatier.c.

References mm_wipe(), and stmt_exec_affected().

Referenced by contact_folder_remove(), and message_folder_remove().

inx_t* magma_folder_fetch (uint64_t *usernum*, uint_t *type*)

Fetch all of a user's folders of a specified type from the database.

Parameters:

usernum the numerical id of the requested user.

type the folder class of the folders to be retrieved (can be M_FOLDER_MESSAGES or M_FOLDER_CONTACTS).

Returns:

NULL on failure, or an *inx* object holding all of the requested folders on success.

Definition at line 21 of file *datatier.c*.

References *inx_alloc()*, *inx_free()*, *inx_insert()*, *log_info*, *log_pedantic*, *M_INX_TREE*, *M_TYPE_UINT64*, *magma_folder_funcs()*, *magma_folder_t*, *mm_wipe()*, *PLACER*, *res_field_block()*, *res_field_length()*, *res_field_uint32()*, *res_field_uint64()*, *res_row_next()*, *res_table_free()*, *stmt_get_result()*, *multi_t::u64*, and *multi_t::val*.

Referenced by *contacts_update()*, and *messages_update()*.

uint64_t magma_folder_insert (uint64_t *usernum*, stringer_t * *name*, uint64_t *parent*, uint32_t *order*, uint_t *type*)

Insert a new folder into the database.

Parameters:

usernum the numerical id of the user to whom the new folder belongs.

name a managed string containing the name of the new folder.

parent the numerical id of the mail folder to be the parent of the new mail folder.

order the order number of this folder in its parent folder.

type the type of the new folder (can be M_FOLDER_MESSAGES or M_FOLDER_CONTACTS).

Returns:

0 on failure, or the numerical id of the newly inserted folder in the database on success.

Definition at line 96 of file *datatier.c*.

References *mm_wipe()*, *st_char_get()*, *st_length_get()*, and *stmt_insert()*.

Referenced by *contact_folder_create()*, and *message_folder_create()*.

bool_t magma_folder_rename (uint64_t *usernum*, uint64_t *foldernum*, uint_t *type*, stringer_t * *rename*)

Rename a folder in the database.

Parameters:

usernum the numerical id of the user requesting the folder renaming.

foldernum the numerical id of the folder to be renamed.

type the type of the folder to be renamed (can be M_FOLDER_MESSAGES or M_FOLDER_CONTACTS).

rename a managed string containing the new name of the specified folder.

Returns:

true on success or false on failure.

Definition at line 181 of file *datatier.c*.

References *mm_wipe()*, *st_char_get()*, *st_length_get()*, and *stmt_exec_affected()*.

Referenced by *contact_folder_rename()*.

magma/objects/mail/datatier.c File Reference

Functions used to interface with and manage message data.

```
#include "magma.h"
```

Functions

- void **mail_db_hide_message** (uint64_t messagenum)
- *Set a message invisible in the database.* **bool_t mail_db_delete_message** (uint64_t usernum, uint64_t messagenum, uint32_t size, **int_t** transaction)
- *Delete a mail message from the mysql database and adjust the owner's quota.* **int_t mail_db_update_message_folder** (uint64_t usernum, uint64_t messagenum, uint64_t source, uint64_t target, int64_t transaction)
- *Update a mail message's parent folder in the database.* uint64_t **mail_db_insert_message** (uint64_t usernum, uint64_t foldernum, uint32_t status, uint32_t size, uint64_t signum, uint64_t sigkey, **int_t** transaction)
- *Insert a mail message into the database.* uint64_t **mail_db_insert_duplicate_message** (uint64_t usernum, uint64_t foldernum, uint32_t status, uint32_t size, uint64_t signum, uint64_t sigkey, uint64_t created, **int_t** transaction)

Insert a duplicate entry for a message in the database.

Detailed Description

Functions used to interface with and manage message data.

Definition in file **datatier.c**.

Function Documentation

bool_t mail_db_delete_message (uint64_t *usernum*, uint64_t *messagenum*, uint32_t *size*, **int_t** *transaction*)

Delete a mail message from the mysql database and adjust the owner's quota.

datatier.c

Parameters:

usernum the user id to whom the target mail message belongs.

messagenum the message id of the mail message to be deleted.

size the storage size, in bytes, of the message to be deleted.

transaction the mysql connection id on which to execute the statements.

Returns:

0 on failure or 1 on success.

Definition at line 46 of file datatier.c.

References log_error, mm_wipe(), and stmt_exec_affected_conn().

Referenced by mail_remove_message().

void mail_db_hide_message (uint64_t *messagenum*)

Set a message invisible in the database.

Parameters:

messageum the message id of the mail message to be hidden.

Returns:

This function returns no value.

Definition at line 20 of file *datatier.c*.

References *mm_wipe()*, and *stmt_exec()*.

Referenced by *mail_load_header()*, and *mail_load_message()*.

uint64_t mail_db_insert_duplicate_message (uint64_t *usernum*, uint64_t *foldernum*, uint32_t *status*, uint32_t *size*, uint64_t *signum*, uint64_t *sigkey*, uint64_t *created*, int_t *transaction*)

Insert a duplicate entry for a message in the database.

Note:

This function will also update the user's storage quota information in the database.

Parameters:

usernum the numerical id of the user that owns the message.

foldernum the numerical id of the parent folder containing the message.

status the status flags value for the message.

size the size, in bytes, of the mail message on disk.

signum the spam signature for the message.

sigkey the spam key for the message.

created the UNIX timestamp of when the message was created.

transaction the transaction id for the database operation, in case the caller wants to roll back the transaction.

Returns:

NULL on failure, or the ID of the newly inserted message on success.

Definition at line 309 of file *datatier.c*.

References *magma_t::active*, *ISNULL*, *log_pedantic*, *magma*, *MAIL_MARK_JUNK*, *mm_wipe()*, *st_char_get()*, *st_length_get()*, *stmt_exec_conn()*, *stmt_insert_conn()*, and *magma_t::storage*.

Referenced by *mail_copy_message()*.

uint64_t mail_db_insert_message (uint64_t *usernum*, uint64_t *foldernum*, uint32_t *status*, uint32_t *size*, uint64_t *signum*, uint64_t *sigkey*, int_t *transaction*)

Insert a mail message into the database.

Note:

This function will also update the user's storage quota information in the database.

Parameters:

usernum the numerical id of the user to whom the mail message will belong.

foldernum the numerical id of the folder that will be the parent folder of the message.

status the status flags of the mail message.

size the size, in bytes, of the mail message on disk.

signum the spam signature for the message.

sigkey the spam key for the message.

transaction the transaction id for the database operation, in case the caller wants to roll back the transaction.

Returns:

0 on failure or the id of the newly inserted mail message on success.

Definition at line 177 of file `datatier.c`.

References `magma_t::active`, `ISNULL`, `log_pedantic`, `magma`, `MAIL_MARK_JUNK`, `mm_wipe()`, `st_char_get()`, `st_length_get()`, `stmt_exec_conn()`, `stmt_insert_conn()`, and `magma_t::storage`.

Referenced by `mail_store_message()`.

`int_t mail_db_update_message_folder (uint64_t usernum, uint64_t messagenum, uint64_t source, uint64_t target, int64_t transaction)`

Update a mail message's parent folder in the database.

usernum the numerical id of the user to whom the mail message belongs. *messagenum* the numerical id of the target mail message of the operation. *source* the numerical id of the parent folder in which the mail message currently resides. *target* the numerical id of the destination folder which is to be the new parent of the mail message. *transaction* a transaction id for the database operation, in case the caller needs to roll back changes on failure.

Returns:

-1 on failure, 0 if the target message could not be located in the database, or 1 on success.

Definition at line 105 of file `datatier.c`.

References `log_pedantic`, `mm_wipe()`, `mysql_stmt_errno_d`, `mysql_stmt_error_d`, `pool_get_obj()`, `sql_pool`, and `stmt_exec_affected_conn()`.

Referenced by `mail_move_message()`.

magma/objects/messages/datatier.c File Reference

Message data functions.

```
#include "magma.h"
```

Functions

- **bool_t messages_fetch** (uint64_t usernum, message_folder_t *folder)

Populate a message folder with all of its child messages from the database.

Detailed Description

Message data functions.

Definition in file **datatier.c**.

Function Documentation

bool_t messages_fetch (uint64_t *usernum*, message_folder_t * *folder*)

Populate a message folder with all of its child messages from the database.

datatier.c

Parameters:

usernum the numerical id of the user that owns the folder.

folder a pointer to the message folder object to be populated.

Returns:

true on success or false on failure.

Definition at line 21 of file datatier.c.

References `inx_insert()`, `log_info`, `log_pedantic`, `M_TYPE_UINT64`, `message_alloc()`, `message_free()`, `message_t`, `mm_wipe()`, `PLACER`, `res_field_block()`, `res_field_length()`, `res_field_uint32()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `stmt_get_result()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `messages_update()`.

magma/objects/users/datatier.c File Reference

The database interface for the user objects.

```
#include "magma.h"
```

Functions

- **bool_t meta_data_flags_replace** (**inx_t** *messages, uint64_t usernum, uint64_t foldernum, uint32_t flags)
- *Remove all user (non-system) flags from a collection of mail messages, and set the specified flags mask for them.* **bool_t meta_data_flags_remove** (**inx_t** *messages, uint64_t usernum, uint64_t foldernum, uint32_t flags)
- *Remove the specified flags mask from a collection of mail messages.* **bool_t meta_data_flags_add** (**inx_t** *messages, uint64_t usernum, uint64_t foldernum, uint32_t flags)
- *Add the specified flags mask to a collection of mail messages.* **uint64_t meta_data_delete_folder** (uint64_t usernum, uint64_t foldernum)
- *Delete a message folder from the database.* **uint64_t meta_data_update_folder_name** (uint64_t usernum, uint64_t foldernum, **stringer_t** *name, uint64_t parent, uint32_t order)
- *Update the record for a message folder in the database.* **uint64_t meta_data_insert_folder** (uint64_t usernum, **stringer_t** *name, uint64_t parent, uint32_t order)
- *Insert a new mail folder into the database.* **void meta_data_update_log** (**meta_user_t** *user, **META_PROT** prot)
- *Update the per-user entry in the Log table for the specified protocol.* **void meta_data_update_lock** (uint64_t usernum, uint8_t lock)
- *Update a user's lock in the database.* **int_t meta_data_insert_tag** (**meta_message_t** *message, **stringer_t** *tag)
- *Insert a tag for a message into the database.* **int_t meta_data_truncate_tags** (**meta_message_t** *message)
- *Remove all tags associated with a message in the database.* **int_t meta_data_delete_tag** (**meta_message_t** *message, **stringer_t** *tag)
- *Remove a tag from a message in the database..* **inx_t * meta_data_fetch_all_tags** (uint64_t usernum)
- *Fetch all the tags attached to messages of a specified user from the database.* **void meta_data_fetch_message_tags** (**meta_message_t** *message)
- *Fetch the tags for a specified message from the database.* **bool_t meta_data_fetch_messages** (**meta_user_t** *user)
- *Fetch all of a user's stored messages from the database and attach them to the meta user object.* **bool_t adjust_message_encryption** (**meta_user_t** *user, **meta_message_t** *message, **stringer_t** *oprivkey)
- *Adjust the encrypted status of a message, in accordance with the user's secure flag.* **void encrypt_user_messages** (**meta_user_t** *user)
- *Encrypt all of a user's messages that aren't tagged as encrypted already.* **void decrypt_user_messages** (**meta_user_t** *user)
- *Decrypt all of a user's messages that aren't tagged as unencrypted already.* **int_t meta_check_message_encryption** (**meta_user_t** *user)
- *Make sure that a user's messages' on-disk encryption statuses match the user's security settings.* **bool_t meta_data_fetch_folders** (**meta_user_t** *user)
- *Retrieve all of a user's message folders from the database.* **bool_t meta_data_fetch_user** (**meta_user_t** *user)
- *Populate a meta user object with user information stored in the database.* **int_t meta_data_user_build** (**meta_user_t** *user, **stringer_t** *passhash, **stringer_t** *passkey)
- *Build a meta user object by username, hashed password, and hashed key storage password.* **int_t meta_data_user_build_storage_keys** (uint64_t usernum, **stringer_t** *passkey, **stringer_t** **priv_out, **stringer_t** **pub_out, bool dont_create, **bool_t** do_trans, uint32_t tid)
- *Retrieve the on-disk mail storage key pair associated with the user, or create it if it doesn't exist.* **int_t meta_data_user_save_storage_keys** (uint64_t usernum, **stringer_t** *passkey, **stringer_t** *pubkey, **stringer_t** *privkey, **bool_t** do_trans, uint32_t tid)

- *Persist the user's storage keys into the mysql database.* **bool_t meta_data_acknowledge_alert** (uint64_t alertnum, uint64_t usernum, uint32_t transaction)
 - *Mark a user alert message as acknowledged in the database.* **inx_t * meta_data_fetch_alerts** (uint64_t usernum)
 - *Get all unacknowledged alert messages for a user.* **bool_t meta_data_fetch_mailbox_aliases** (meta_user_t *user)
 - *Get all the mailbox aliases for a specified user.* **bool_t meta_data_check_mailbox** (stringer_t *address)
- Check to see if a specified mailbox exists.*
-

Detailed Description

The database interface for the user objects.

Definition in file **datatier.c**.

Function Documentation

bool_t adjust_message_encryption (meta_user_t * *user*, meta_message_t * *message*, stringer_t * *oprivkey*)

Adjust the encrypted status of a message, in accordance with the user's secure flag.

Parameters:

user a pointer to the meta user object owning the specified message.

message a pointer to the meta message that should have its on-disk data updated accordingly.

oprivkey if re-encryption is specified (user's secure flag is set and the message is already encrypted), this is a pointer to a managed string that contains the user's original private key in binary form.

Returns:

true on success or false on failure.

Definition at line 746 of file datatier.c.

References cryptex_free(), cryptex_total_length(), ecies_decrypt(), ecies_encrypt(), ECIES_PRIVATE_BINARY, ECIES_PUBLIC_BINARY, file_load(), meta_user_t::flags, FMESSAGE_MAGIC_1, FMESSAGE_MAGIC_2, FMESSAGE_OPT_ENCRYPTED, meta_message_t::foldernum, get_temp_file_handle(), inx_alloc(), inx_free(), inx_insert(), log_pedantic, M_INX_LINKED, M_TYPE_UINT64, mail_message_path(), MAIL_STATUS_ENCRYPTED, message_fheader_t, meta_message_t::messagenum, meta_data_flags_add(), meta_data_flags_remove(), META_USER_ENCRYPT_DATA, ns_free(), ns_import(), meta_message_t::server, st_char_get(), st_data_get(), st_length_get(), meta_message_t::status, meta_user_t::storage_privkey, meta_user_t::storage_pubkey, tran_commit(), tran_rollback(), tran_start(), multi_t::type, multi_t::u64, meta_user_t::usernum, and multi_t::val.

Referenced by decrypt_user_messages(), and encrypt_user_messages().

void decrypt_user_messages (meta_user_t * *user*)

Decrypt all of a user's messages that aren't tagged as unencrypted already.

Parameters:

user the meta user object to have its messages processed.

Returns:

This function returns no value.

Definition at line 997 of file `datatier.c`.

References `adjust_message_encryption()`, `encrypt_user_messages()`, `enqueue()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `log_info`, `log_pedantic`, `MAIL_STATUS_ENCRYPTED`, `meta_user_t::messages`, `META_PROT_GENERIC`, `meta_user_ref_dec()`, `st_char_get()`, `meta_message_t::status`, and `meta_user_t::username`.

Referenced by `meta_check_message_encryption()`.

void encrypt_user_messages (meta_user_t * user)

Encrypt all of a user's messages that aren't tagged as encrypted already.

Parameters:

user the meta user object to have its messages processed.

Returns:

This function returns no value.

Definition at line 953 of file `datatier.c`.

References `adjust_message_encryption()`, `enqueue()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `log_info`, `log_pedantic`, `MAIL_STATUS_ENCRYPTED`, `meta_user_t::messages`, `META_PROT_GENERIC`, `meta_user_ref_dec()`, `st_char_get()`, `meta_message_t::status`, and `meta_user_t::username`.

Referenced by `decrypt_user_messages()`, and `meta_check_message_encryption()`.

int_t meta_check_message_encryption (meta_user_t * user)

Make sure that a user's messages' on-disk encryption statuses match the user's security settings.

Note:

If the user's secure flag is on, then all messages should be encrypted. Any unencrypted messages need to be encrypted in place. If the secure flag has been disabled and there are still encrypted messages on disk, the messages need to be reverted to plaintext form.

Parameters:

user the meta user object that owns the specified messages.

Returns:

-1 on failure, 0 if there were no messages to be sync'ed, or 1 if additional encryption processing is required.

Definition at line 1044 of file `datatier.c`.

References `decrypt_user_messages()`, `encrypt_user_messages()`, `enqueue()`, `meta_user_t::flags`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_ENCRYPTED`, `meta_user_t::messages`, `META_PROT_GENERIC`, `META_USER_ENCRYPT_DATA`, `meta_user_ref_add()`, `meta_message_t::status`, `meta_user_t::storage_privkey`, and `meta_user_t::storage_pubkey`.

Referenced by `meta_data_fetch_messages()`.

bool_t meta_data_acknowledge_alert (uint64_t *alertnum*, uint64_t *usernum*, uint32_t *transaction*)

Mark a user alert message as acknowledged in the database.

datatier.c

Note:

If the table is not updated immediately, another check is made to see if the alert is still pending. If so, false is returned.

Parameters:

alertnum the numerical id of the alert message to be acknowledged.

usernum the numerical id of the user to whom the alert message belongs.

transaction the mysql transaction id of the acknowledgment operation, in cases batch changes need to be rolled back.

Returns:

true if the alert was acknowledged successfully, or false on failure.

Definition at line 1655 of file datatier.c.

References log_pedantic, mm_wipe(), res_field_uint64(), res_row_next(), res_table_free(), stmt_exec_affected_conn(), and stmt_get_result_conn().

Referenced by portal_endpoint_alert_acknowledge().

bool_t meta_data_check_mailbox (stringer_t * *address*)

Check to see if a specified mailbox exists.

Parameters:

address a managed string containing the email address to be matched against any of the rows in the Mailboxes table.

Returns:

true if the specified email address exists as a configured mailbox in the database or false otherwise.

Definition at line 1837 of file datatier.c.

References log_pedantic, mm_wipe(), res_row_next(), res_table_free(), st_char_get(), st_empty(), st_length_get(), and stmt_get_result().

Referenced by meta_get().

uint64_t meta_data_delete_folder (uint64_t *usernum*, uint64_t *foldernum*)

Delete a message folder from the database.

Parameters:

usernum the numerical id of the user that owns the folder to be deleted.

foldernum the folder id of the message folder to be deleted.

Returns:

1 if the message folder was successfully, or <= 0 if there was an error.

Definition at line 237 of file datatier.c.

References M_FOLDER_MESSAGES, mm_wipe(), stmt_exec_affected(), and type().

Referenced by `imap_folder_remove()`.

`int_t meta_data_delete_tag (meta_message_t * message, stringer_t * tag)`

Remove a tag from a message in the database..

Parameters:

message a pointer to the meta message object of the message to have the tag stripped.

tag a managed string containing the name of the tag to be deleted.

Returns:

0 on success or -1 on failure.

Definition at line 503 of file `datatier.c`.

References `meta_message_t::messagenum`, `mm_wipe()`, `st_char_get()`, `st_length_get()`, and `stmt_exec_affected()`.

Referenced by `portal_endpoint_messages_tag()`.

`inx_t* meta_data_fetch_alerts (uint64_t usernum)`

Get all unacknowledged alert messages for a user.

Parameters:

usernum the numerical id of the user for whom the alert messages will be fetched.

Returns:

NULL on failure, or a pointer to an `inx` holder containing all of the user's alert messages on success.

Definition at line 1718 of file `datatier.c`.

References `alert_alloc()`, `inx_alloc()`, `inx_free()`, `inx_insert()`, `log_error`, `log_info`, `log_pedantic`, `M_INX_LINKED`, `M_TYPE_UINT64`, `mm_free()`, `mm_wipe()`, `PLACER`, `res_field_block()`, `res_field_length()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `stmt_get_result()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `portal_endpoint_alert_list()`.

`inx_t* meta_data_fetch_all_tags (uint64_t usernum)`

Fetch all the tags attached to messages of a specified user from the database.

Parameters:

usernum the numerical id of the user for whom the message tags will be fetched.

Returns:

NULL on failure, or an `inx` holder containing a list of all the user's messages' tags as managed strings on success.

Definition at line 534 of file `datatier.c`.

References `inx_alloc()`, `inx_free()`, `inx_insert()`, `log_error`, `log_pedantic`, `M_INX_LINKED`, `M_TYPE_UINT64`, `res_field_string()`, `res_row_next()`, `res_table_free()`, `st_free()`, `stmt_get_result()`, `multi_t::type`, `multi_t::u64`, and `multi_t::val`.

Referenced by portal_endpoint_messages_tags().

bool_t meta_data_fetch_folders (meta_user_t * user)

Retrieve all of a user's message folders from the database.

Note:

If the user already had a working set of message folders they will be deleted first.

Parameters:

user a pointer to the meta user object of the user making the request, which will be updated on success.

Returns:

-1 on failure or 1 on success.

Definition at line 1093 of file datatier.c.

References meta_folder_t::foldernum, meta_user_t::folders, inx_alloc(), inx_cleanup(), inx_insert(), log_error, log_pedantic, M_FOLDER_MESSAGES, M_INX_LINKED, M_TYPE_UINT64, mm_alloc(), mm_copy(), mm_free(), mm_wipe(), meta_folder_t::name, meta_folder_t::order, meta_folder_t::parent, res_field_block(), res_field_length(), res_field_uint32(), res_field_uint64(), res_row_next(), res_table_free(), stmt_get_result(), multi_t::type, type(), multi_t::u64, meta_user_t::usernum, and multi_t::val.

Referenced by meta_folders_update().

bool_t meta_data_fetch_mailbox_aliases (meta_user_t * user)

Get all the mailbox aliases for a specified user.

Parameters:

user a pointer to the partially populated meta user object to be queried, with its usernum field set.

Returns:

true on success or false on failure.

Definition at line 1775 of file datatier.c.

References alias_alloc(), meta_user_t::aliases, inx_alloc(), inx_cleanup(), inx_insert(), log_error, log_info, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, mm_free(), mm_wipe(), PLACER, res_field_block(), res_field_length(), res_field_uint64(), res_field_uint8(), res_row_next(), res_table_free(), st_char_get(), st_length_int(), stmt_get_result(), multi_t::u64, meta_user_t::usernum, and multi_t::val.

Referenced by meta_user_build(), and meta_user_update().

void meta_data_fetch_message_tags (meta_message_t * message)

Fetch the tags for a specified message from the database.

Note:

The results of the operation will be stored in the specified meta message object's "tags" member.

Parameters:

message the meta message object for which the tags will be looked up.

Returns:

This function returns no value.

Definition at line 590 of file `datatier.c`.

References `ar_alloc()`, `ar_append()`, `ARRAY_TYPE_STRINGER`, `meta_message_t::messagenum`, `mm_wipe()`, `res_field_string()`, `res_row_count()`, `res_row_next()`, `res_table_free()`, `stmt_get_result()`, and `meta_message_t::tags`.

Referenced by `meta_data_fetch_messages()`, and `portal_endpoint_messages_tag()`.

bool_t meta_data_fetch_messages (meta_user_t * user)

Fetch all of a user's stored messages from the database and attach them to the meta user object.

Note:

Any of the user's existing messages will be destroyed first to allow for updates.

Parameters:

user the meta user object whose mail messages will be retrieved.

Returns:

true on success or false on failure.

Definition at line 629 of file `datatier.c`.

References `meta_message_t::created`, `meta_message_t::foldernum`, `inx_alloc()`, `inx_cleanup()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_insert()`, `log_error`, `log_info`, `log_pedantic`, `M_INX_LINKED`, `M_TYPE_UINT64`, `MAIL_STATUS_TAGGED`, `meta_message_t::messagenum`, `meta_user_t::messages`, `meta_check_message_encryption()`, `meta_data_fetch_message_tags()`, `meta_message_free()`, `mm_alloc()`, `mm_copy()`, `mm_free()`, `mm_wipe()`, `res_field_block()`, `res_field_length()`, `res_field_uint32()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `meta_message_t::server`, `meta_message_t::sigkey`, `meta_message_t::signum`, `meta_message_t::size`, `st_char_get()`, `meta_message_t::status`, `stmt_get_result()`, `multi_t::type`, `multi_t::u64`, `meta_user_t::username`, `meta_user_t::usernum`, and `multi_t::val`.

Referenced by `meta_messages_login_update()`, and `meta_messages_update()`.

bool_t meta_data_fetch_user (meta_user_t * user)

Populate a meta user object with user information stored in the database.

Note:

Fields fetched from the database include: flags, hashed password, lock & ssl status, and quota information.

Parameters:

user a pointer to the partially populated meta user object to be updated, with the `usernum` field already supplied.

Returns:

true on success or false on failure.

LOW: This user fetch function was intended only to refresh existing user configurations/preferences. In theory an inactivity lock would have been cleared when the session was created so we shouldn't need to even check that value at this stage.

Definition at line 1193 of file `datatier.c`.

References meta_user_t::flags, meta_user_t::lock_status, meta_data_update_lock(), META_USER_ENCRYPT_DATA, META_USER_OVERQUOTA, META_USER_SSL, mm_wipe(), meta_user_t::passhash, res_field_int8(), res_field_string(), res_row_next(), res_table_free(), st_cleanup(), stmt_get_result(), and meta_user_t::usernum.

Referenced by meta_user_build(), and meta_user_update().

bool_t meta_data_flags_add (inx_t * *messages*, uint64_t *usernum*, uint64_t *foldernum*, uint32_t *flags*)

Add the specified flags mask to a collection of mail messages.

Parameters:

messages an inx holder containing the collection of messages to have their flags updated.
usernum the numerical id of the user to whom the target messages belong, for validation purposes.
foldernum the numerical id of the parent folder containing the messages to be updated, for validation purposes.
flags a mask of all flags that are to be added to any matching messages in the collection.

Returns:

true on success or false on failure.

Definition at line 176 of file datatier.c.

References inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), log_pedantic, meta_message_t::messagenum, mm_wipe(), and stmt_exec().

Referenced by adjust_message_encryption(), imap_fetch(), imap_update_flags(), portal_endpoint_messages_flag(), and portal_endpoint_messages_tag().

bool_t meta_data_flags_remove (inx_t * *messages*, uint64_t *usernum*, uint64_t *foldernum*, uint32_t *flags*)

Remove the specified flags mask from a collection of mail messages.

Parameters:

messages an inx holder containing the collection of messages to have their flags removed.
usernum the numerical id of the user to whom the target messages belong, for validation purposes.
foldernum the numerical id of the parent folder containing the messages to be updated, for validation purposes.
flags a mask of all flags that are to be stripped from any matching messages in the collection.

Returns:

true on success or false on failure.

Definition at line 104 of file datatier.c.

References meta_message_t::foldernum, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), log_pedantic, meta_message_t::messagenum, mm_wipe(), and stmt_exec().

Referenced by adjust_message_encryption(), imap_select(), imap_update_flags(), portal_endpoint_messages_flag(), and portal_endpoint_messages_tag().

bool_t meta_data_flags_replace (inx_t * *messages*, uint64_t *usernum*, uint64_t *foldernum*, uint32_t *flags*)

Remove all user (non-system) flags from a collection of mail messages, and set the specified flags mask for them.

Note:

The new mask can contain both user and system flags, but only user flags will be stripped from each message initially.

Parameters:

messages an `inx` holder containing the collection of messages to have their flags updated.

usernum the numerical of the user to whom the target messages belong, for validation purposes.

foldernum the numerical id of the parent folder containing the messages to be updated, for validation purposes.

flags a mask of all flags that are to be added to any matching messages in the collection.

Returns:

true on success or false on failure.

Definition at line 24 of file `datatier.c`.

References `meta_message_t::foldernum`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `log_pedantic`, `MAIL_STATUS_USER_FLAGS`, `meta_message_t::messagenum`, `mm_wipe()`, and `stmt_exec()`.

Referenced by `imap_update_flags()`, and `portal_endpoint_messages_flag()`.

`uint64_t meta_data_insert_folder (uint64_t usernum, stringer_t * name, uint64_t parent, uint32_t order)`

Insert a new mail folder into the database.

Parameters:

usernum the numerical id of the user to whom the new folder belongs.

name a managed string containing the name of the new mail folder.

parent the numerical id of the mail folder to be the parent of the new mail folder.

order the order number of this folder in its parent folder.

Returns:

0 on failure, or the numerical id of the newly inserted mail folder in the database on success.

Definition at line 329 of file `datatier.c`.

References `M_FOLDER_MESSAGES`, `mm_wipe()`, `st_char_get()`, `st_length_get()`, `stmt_insert()`, and `type()`.

Referenced by `imap_folder_create()`, and `imap_folder_rename()`.

`int_t meta_data_insert_tag (meta_message_t * message, stringer_t * tag)`

Insert a tag for a message into the database.

Parameters:

message the meta message object of the message to be tagged.

tag a managed string containing the name of the tag.

Returns:

0 on success or -1 on failure.

Definition at line 445 of file `datatier.c`.

References meta_message_t::messagenum, mm_wipe(), st_char_get(), st_length_get(), and stmt_exec_affected().

Referenced by portal_endpoint_messages_tag().

int_t meta_data_truncate_tags (meta_message_t * message)

Remove all tags associated with a message in the database.

Parameters:

message a pointer to the meta message object of the message to have all of its tags stripped.

Returns:

0 on success or -1 on failure.

Definition at line 476 of file datatier.c.

References meta_message_t::messagenum, mm_wipe(), and stmt_exec_affected().

Referenced by portal_endpoint_messages_tag().

uint64_t meta_data_update_folder_name (uint64_t usernum, uint64_t foldernum, stringer_t * name, uint64_t parent, uint32_t order)

Update the record for a message folder in the database.

Parameters:

usernum the numerical id of the user that owns the specified folder.

foldernum the id of the folder to have its properties adjusted.

name a managed string containing the new name of the specified folder.

parent the id of the new parent folder to be set for the specified message folder.

order the value of the order for the specified folder.

Returns:

1 on success, or <= 0 on failure.

Definition at line 275 of file datatier.c.

References M_FOLDER_MESSAGES, mm_wipe(), st_char_get(), st_length_get(), stmt_exec_affected(), and type().

Referenced by imap_folder_rename().

void meta_data_update_lock (uint64_t usernum, uint8_t lock)

Update a user's lock in the database.

Parameters:

usernum the numerical id of the user for whom the lock will be set.

lock the new value to which the specified user's lock will be set.

Returns:

This function returns no value.

Definition at line 408 of file datatier.c.

References `log_pedantic`, `mm_wipe()`, and `stmt_exec_affected()`.

Referenced by `meta_data_fetch_user()`, `meta_data_user_build()`, and `smtp_fetch_authorization()`.

`void meta_data_update_log (meta_user_t * user, META_PROT prot)`

Update the per-user entry in the Log table for the specified protocol.

Note:

This function will set the last session timestamp for the user, and increment the sessions counter in the database.

Parameters:

user the meta user object of the user making the logging request.

prot the protocol associated with the log request: `META_PROT_POP`, `META_PROT_IMAP`, or `META_PROT_WEB`.

Returns:

This function returns no value.

Definition at line 375 of file `datatier.c`.

References `log_pedantic`, `META_PROT_IMAP`, `META_PROT_POP`, `META_PROT_WEB`, `mm_wipe()`, `stmt_exec_affected()`, and `meta_user_t::usernum`.

Referenced by `imap_login()`, and `pop_pass()`.

`int_t meta_data_user_build (meta_user_t * user, stringer_t * passhash, stringer_t * passkey)`

Build a meta user object by username, hashed password, and hashed key storage password.

Parameters:

user a user meta object with the username field populated.

passhash a managed string with a multi-round hashed of the user's password for mysql database lookup.

passkeys a managed string with a single-pass hash of the user's password.

Returns:

-1 for unexpected program/system error, 0 for password auth failure, or 1 on success.

Definition at line 1264 of file `datatier.c`.

References `meta_user_t::flags`, `meta_user_t::lock_status`, `log_pedantic`, `meta_data_update_lock()`, `meta_data_user_build_storage_keys()`, `META_USER_ENCRYPT_DATA`, `META_USER_OVERQUOTA`, `META_USER_SSL`, `mm_wipe()`, `meta_user_t::passhash`, `res_field_int8()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `st_char_get()`, `st_dup()`, `st_empty()`, `st_free()`, `st_length_get()`, `st_length_int()`, `stmt_get_result()`, `meta_user_t::storage_privkey`, `meta_user_t::storage_pubkey`, `meta_user_t::username`, and `meta_user_t::usernum`.

Referenced by `meta_user_build()`.

`int_t meta_data_user_build_storage_keys (uint64_t usernum, stringer_t * passkey, stringer_t ** priv_out, stringer_t ** pub_out, bool dont_create, bool_t do_trans, uint32_t tid)`

Retrieve the on-disk mail storage key pair associated with the user, or create it if it doesn't exist.

Parameters:

usernum the user id of the target account.

passkey the single-round hashed password for storage key management, which can be NULL if *dont_create* is specified.

priv_out a pointer to a managed string to receive a copy of the ECIES private key if not NULL.

pub_out a pointer to a managed string to receive a copy of the ECIES public key if not NULL.

dont_create if true, ensures that storage keys won't be created if they don't already exist.

do_trans specifies whether or not a transaction id will be supplied for database operations.

tid if *do_trans* is set, the mysql transaction id to be used for all database operations.

Returns:

-1 on general failure, 0 if keys were successfully retrieved, or 1 if keys were generated.

Definition at line 1381 of file `datatier.c`.

References `base64_decode_mod()`, `base64_decode_opts()`, `CONTIGUOUS`, `ecies_key_create()`, `ecies_key_free()`, `ecies_key_private_bin()`, `ecies_key_public_bin()`, `log_info`, `log_pedantic`, `MANAGED_T`, `meta_data_user_save_storage_keys()`, `mm_free()`, `mm_sec_free()`, `mm_wipe()`, `res_field_block()`, `res_field_length()`, `res_row_next()`, `res_table_free()`, `scramble_decrypt()`, `SECURE`, `st_alloc_opts()`, `st_cleanup()`, `st_copy_in()`, `st_data_get()`, `st_empty()`, `st_free()`, `st_import()`, `stmt_get_result()`, and `stmt_get_result_conn()`.

Referenced by `meta_data_user_build()`, `register_data_insert_user()`, and `smtp_store_message()`.

```
int_t meta_data_user_save_storage_keys (uint64_t usernum,  stringer_t * passkey,  stringer_t  
* pubkey,  stringer_t * privkey,  bool_t do_trans,  uint32_t tid)
```

Persist the user's storage keys into the mysql database.

Parameters:

usernum the user id of the target account.

passkey the passkey that will be used to encrypt the user's private key symmetrically in the database.

pubkey the user's storage public key as a base64 encoded string.

privkey the user's encrypted storage private key as a base64 encoded string.

do_trans specifies whether or not a transaction id will be supplied for database operations.

tid if *do_trans* is set, the mysql transaction id to be used for all database operations.

Returns:

-1 on failure or 0 on success.

Definition at line 1576 of file `datatier.c`.

References `base64_encode_mod()`, `log_info`, `mm_copy()`, `mm_wipe()`, `scramble_encrypt()`, `scramble_free()`, `scramble_total_length()`, `st_char_get()`, `st_free()`, `st_import()`, `st_length_get()`, `stmt_exec_affected()`, and `stmt_exec_affected_conn()`.

Referenced by `meta_data_user_build_storage_keys()`.

magma/objects/warehouse/datatier.c File Reference

Functions used by the warehouse objects to access the database.

```
#include "magma.h"
```

Functions

- **inx_t * warehouse_fetch_domains** (void)
 - *Fetch the list of configured domains from the database.* **inx_t * warehouse_fetch_patterns** (void)
Fetch the pattern list from the database.
-

Detailed Description

Functions used by the warehouse objects to access the database.

Definition in file **datatier.c**.

Function Documentation

inx_t* warehouse_fetch_domains (void)

Fetch the list of configured domains from the database.

datatier.c

Returns:

NULL on failure, or an inx object holding all the configured domains on success.

Definition at line 19 of file datatier.c.

References domain_alloc(), inx_alloc(), inx_free(), inx_insert(), log_check, log_info, log_pedantic, M_INX_TREE, M_TYPE_STRINGER, MAGMA_HOSTNAME_MAX, mm_free(), PLACER, res_field_block(), res_field_int8(), res_field_length(), res_row_next(), res_table_free(), multi_t::st, stmt_get_result(), and multi_t::val.

Referenced by domain_start().

inx_t* warehouse_fetch_patterns (void)

Fetch the pattern list from the database.

Returns:

NULL on failure or a pointer to an inx holder containing a collection of managed strings with the patterns on success.

Definition at line 69 of file datatier.c.

References inx_alloc(), inx_free(), inx_insert(), log_pedantic, M_INX_LINKED, M_TYPE_UINT64, res_field_string(), res_row_next(), res_table_free(), st_free(), stmt_get_result(), multi_t::u64, and multi_t::val.

Referenced by pattern_update().

magma/servers/smtp/datatier.c File Reference

Functions used to interface with and manage data needed by the SMTP protocol.

```
#include "magma.h"
```

Functions

- **int_t smtp_get_action** (**chr_t** *string, **size_t** length)
- *Parse an smtp event action string from the Dispatch table. **stringer_t** * smtp_fetch_autoreply* (**uint64_t** autoreply, **uint64_t** usernum)
- *Fetch a specified auto-reply message for a user. **int_t** smtp_fetch_inbound* (**credential_t** *cred, **stringer_t** *address, **smtp_inbound_prefs_t** **output)
- *Fetch a user's smtp inbound preferences from the database. **table_t** * smtp_fetch_rollmessages* (**uint64_t** usernum)
- *Retrieve a list, at a maximum of 20 entries, of the oldest messages owned by a user. void* **smtp_update_receive_stats** (**connection_t** *con, **smtp_inbound_prefs_t** *prefs)
- *Update the receiving statistics and per-user log tables in the database for a successfully received smtp message. **uint64_t** smtp_insert_spamsig* (**smtp_inbound_prefs_t** *prefs, **uint64_t** key, **int_t** code)
- *Store a spam signature in the database. void* **smtp_update_transmission_stats** (**connection_t** *con)
- *Update the transmission and per-user log tables in the database for a successfully sent smtp message. **int_t** smtp_check_transmit_quota* (**uint64_t** usernum, **size_t** num_recipients, **smtp_outbound_prefs_t** *prefs)
- *Check to see if a user's current mail send request would push them over their daily transmission quota. **int_t** smtp_fetch_authorization* (**credential_t** *cred, **smtp_outbound_prefs_t** **output)
- *Check if a user is authorized to send messages, and retrieve the user's outbound smtp preferences. **int_t** smtp_check_receive_quota* (**connection_t** *con, **smtp_inbound_prefs_t** *prefs)
- *Check the user's received statistics from the database to see if receiving a message would result in a quota overage. **int_t** smtp_check_authorized_from* (**uint64_t** usernum, **stringer_t** *address)

Check to see if a user is permitted to send email originating from a specified email address.

Detailed Description

Functions used to interface with and manage data needed by the SMTP protocol.

Definition in file **datatier.c**.

Function Documentation

int_t smtp_check_authorized_from (**uint64_t** *usernum*, **stringer_t** * *address*)

Check to see if a user is permitted to send email originating from a specified email address.

datatier.c

Note:

The authorization attempt first checks against the specified email address, and if unsuccessful, upon the domain component of the email address if wildcards are enabled for that domain.

Parameters:

usernum the numerical id of the user attempting to send the mail message.

address a pointer to a managed string containing the From address value of the mail message to be sent.

Returns:

1 if the user is authorized to send email from the specified address, or 0 otherwise.

Definition at line 760 of file `datatier.c`.

References `domain_wildcard()`, `mail_domain_get()`, `mm_wipe()`, `placer_t`, `res_row_count()`, `res_table_free()`, `st_char_get()`, `st_length_get()`, and `stmt_get_result()`.

Referenced by `portal_outbound_checks()`, and `smtp_data_outbound()`.

`int_t smtp_check_receive_quota (connection_t * con, smtp_inbound_prefs_t * prefs)`

Check the user's received statistics from the database to see if receiving a message would result in a quota overage.

Note:

The sum total of all emails received by the user over the past 24 hours is calculated from the database, and these checks are made: 1. The amount of mail messages received in the past 24 hours by the user does not exceed their daily mail received quota. 2. The messages received in the past 24 hours by the user from this subnet does not exceed the user's daily per-subnet received quota.

Parameters:

con a pointer to the connection object over which the smtp message was received.

prefs a pointer to the user's smtp inbound mail preferences.

Returns:

0 if message receipt is permitted, 1 if the general daily receiving limit was exceeded, 2 if the sending subnet's transmission limit was exceeded, or -1 if there was a general error.

Definition at line 695 of file `datatier.c`.

References `con_addr_subnet()`, `smtp_inbound_prefs_t::daily_rcv_limit`, `smtp_inbound_prefs_t::daily_rcv_limit_ip`, `log_pedantic`, `mm_wipe()`, `number`, `res_field_block()`, `res_field_length()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `st_char_get()`, `st_free()`, `st_length_get()`, `stmt_get_result()`, `uint64_conv_bl()`, and `smtp_inbound_prefs_t::usernum`.

Referenced by `smtp_rcpt_to()`.

`int_t smtp_check_transmit_quota (uint64_t usernum, size_t num_recipients, smtp_outbound_prefs_t * prefs)`

Check to see if a user's current mail send request would push them over their daily transmission quota.

Note:

This check is performed by querying the database to see how many messages a user has sent in the past 24 hour period, and by adding the current number of recipients of the pending email request to that number to see if their quota would be exceeded.

Parameters:

con a pointer to the connection object of the user attempting to send mail.

Returns:

-1 on error, 0 if the send operation is permitted, or 1 if the send operation would result in a daily send quota overage.

Definition at line 525 of file `datatier.c`.

References `smtp_outbound_prefs_t::daily_send_limit`, `log_pedantic`, `mm_wipe()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `smtp_outbound_prefs_t::sent_today`, and `stmt_get_result()`.

Referenced by portal_outbound_checks(), and smtp_data_outbound().

int_t smtp_fetch_authorization (credential_t * cred, smtp_outbound_prefs_t ** output)

Check if a user is authorized to send messages, and retrieve the user's outbound smtp preferences.

Note:

This function first checks if the account is locked in the Users table; next it populates the user's outbound mail preferences with a combination of data from the Users and Dispatch tables. The number of messages the user has sent in the past 24 hours is also computed from the Transmitting table.

Parameters:

cred a pointer to a credential object for the user, which must be of type CREDENTIAL_AUTH.

output a pointer to the address of an outbound smtp preferences object to receive the value of the lookup.

Returns:

1 on success or <= 0 on failure. 0: authentication failure (invalid username/password combination). -1: general or database failure. -2: the account is subject to an administrative lock. -3: the account is locked due to suspicion of abuse violations. -4: the account has been locked at the request of the user.

Definition at line 577 of file datatier.c.

References credential_t::auth, CREDENTIAL_AUTH, smtp_outbound_prefs_t::daily_send_limit, smtp_outbound_prefs_t::domain, smtp_outbound_prefs_t::importance, log_error, log_pedantic, meta_data_update_lock(), mm_alloc(), mm_free(), mm_wipe(), res_field_int8(), res_field_string(), res_field_uint32(), res_field_uint64(), res_row_next(), res_table_free(), smtp_outbound_prefs_t::send_size_limit, smtp_outbound_prefs_t::sent_today, smtp_outbound_prefs_t::ssl, st_char_get(), st_length_get(), st_length_int(), stmt_get_result(), credential_t::type, and smtp_outbound_prefs_t::usernum.

Referenced by portal_outbound_checks(), smtp_auth_login(), and smtp_auth_plain().

stringer_t* smtp_fetch_autoreply (uint64_t autoreply, uint64_t usernum)

Fetch a specified auto-reply message for a user.

Note:

This function first checks the cache, and falls back to the database.

Parameters:

autoreply the numerical id of the auto-reply message in the database.

usernum the numerical id of the user to whom the auto-reply message belongs.

Returns:

NULL on failure, or a pointer to a managed string containing the user's auto-reply on success.

Definition at line 56 of file datatier.c.

References cache_get(), cache_set(), log_pedantic, mm_wipe(), PLACER, res_field_string(), res_row_next(), res_table_free(), and stmt_get_result().

Referenced by smtp_reply().

int_t smtp_fetch_inbound (credential_t * cred, stringer_t * address, smtp_inbound_prefs_t ** output)

Fetch a user's smtp inbound preferences from the database.

Parameters:

cred a pointer to the credential object of a user with
address Returns -1 for errors, -2 for an admin lock, -3 for an inactivity lock, -4 for a user lock, -5 for an abuse lock, -6 if the domain isn't local and 0 if the domain is local but the address wasn't found. If everything works, return 1 to indicate success.

Definition at line 116 of file *datatier.c*.

References *smtp_inbound_filter_t::action*, *smtp_inbound_prefs_t::address*, *smtp_inbound_prefs_t::autoreply*, *smtp_inbound_prefs_t::bounces*, *CONTIGUOUS*, *CREDENTIAL_MAIL*, *smtp_inbound_prefs_t::daily_rcv_limit*, *smtp_inbound_prefs_t::daily_rcv_limit_ip*, *smtp_inbound_prefs_t::dkim*, *smtp_inbound_prefs_t::dkimaction*, *smtp_inbound_prefs_t::domain*, *domain_wildcard()*, *smtp_inbound_filter_t::expression*, *smtp_inbound_filter_t::field*, *smtp_inbound_prefs_t::filters*, *smtp_inbound_filter_t::foldernum*, *smtp_inbound_prefs_t::forwarded*, *smtp_inbound_prefs_t::greylist*, *smtp_inbound_prefs_t::greytime*, *HEAP*, *smtp_inbound_prefs_t::inbox*, *inx_alloc()*, *inx_insert()*, *smtp_inbound_filter_t::label*, *smtp_inbound_filter_t::location*, *log_error*, *log_pedantic*, *M_INX_LINKED*, *M_TYPE_UINT64*, *credential_t::mail*, *MANAGED_T*, *mm_alloc()*, *mm_free()*, *mm_wipe()*, *smtp_inbound_prefs_t::overquota*, *smtp_inbound_prefs_t::phish*, *smtp_inbound_prefs_t::phishaction*, *smtp_inbound_prefs_t::quota*, *smtp_inbound_prefs_t::rbl*, *smtp_inbound_prefs_t::rblaction*, *smtp_inbound_prefs_t::rcptto*, *smtp_inbound_prefs_t::rcv_size_limit*, *res_field_block()*, *res_field_int8()*, *res_field_length()*, *res_field_string()*, *res_field_uint32()*, *res_field_uint64()*, *res_row_count()*, *res_row_next()*, *res_table_free()*, *smtp_inbound_prefs_t::rollout*, *smtp_inbound_filter_t::rulenum*, *smtp_inbound_prefs_t::secure*, *SMTP_FILTER_ACTION_LABEL*, *SMTP_FILTER_ACTION_MOVE*, *SMTP_FILTER_LOCATION_FIELD*, *smtp_free_inbound()*, *smtp_get_action()*, *smtp_list_free_filter()*, *smtp_inbound_prefs_t::spam*, *smtp_inbound_prefs_t::spamaction*, *smtp_inbound_prefs_t::spf*, *smtp_inbound_prefs_t::spfaction*, *st_char_get()*, *st_dupe()*, *st_dupe_opts()*, *st_length_get()*, *st_length_int()*, *stmt_get_result()*, *smtp_inbound_prefs_t::stor_size*, *smtp_inbound_filter_t::type*, *credential_t::type*, *multi_t::u64*, *smtp_inbound_prefs_t::usernum*, *multi_t::val*, *smtp_inbound_prefs_t::virus*, and *smtp_inbound_prefs_t::virusaction*.

Referenced by *smtp_rcpt_to()*.

table_t* smtp_fetch_rollmessages (uint64_t usernum)

Retrieve a list, at a maximum of 20 entries, of the oldest messages owned by a user.

Parameters:

usernum the numerical id of the user whose messages are to be queried.

Returns:

NULL on failure, or a pointer to a sql results set containing the user's oldest messages on success.

Definition at line 327 of file *datatier.c*.

References *log_pedantic*, *mm_wipe()*, and *stmt_get_result()*.

Referenced by *smtp_rollout()*.

int_t smtp_get_action (chr_t * string, size_t length)

Parse an smtp event action string from the Dispatch table.

Note:

These values describe user-specified actions for events related to the spam filter, virus scanner, phishing detection, SPF, DKIM, and RBL checks.

Parameters:

string a pointer to a null-terminated string containing the name of the action.
length the length, in bytes, of the action string.

Returns:

the code of the corresponding smtp event action, SMTP_ACTION_UNDEFINED if the action is not known, or SMTP_ACTION_ERROR on failure.

Definition at line 24 of file datatier.c.

References log_error, PLACER, SMTP_ACTION_BOUNCE, SMTP_ACTION_DELETE, SMTP_ACTION_ERROR, SMTP_ACTION_MARK, SMTP_ACTION_MARK_READ, SMTP_ACTION_REJECT, SMTP_ACTION_UNDEFINED, and st_cmp_cs_eq().

Referenced by smtp_fetch_inbound().

uint64_t smtp_insert_spamsig (smtp_inbound_prefs_t * *prefs*, uint64_t *key*, int_t *code*)

Store a spam signature in the database.

See also:

dspam_process()

Parameters:

prefs a pointer to the specified user's inbound mail preferences data.
key a randomly chosen authentication key for the signature.
code the dspam return code for the message from dspam_process()

Returns:

0 on failure, or the number of the newly inserted spam signature on success.

Definition at line 428 of file datatier.c.

References log_pedantic, mm_wipe(), smtp_inbound_prefs_t::spamsig, st_char_get(), st_length_get(), stmt_insert(), and smtp_inbound_prefs_t::usernum.

Referenced by smtp_store_spamsig().

void smtp_update_receive_stats (connection_t * *con*, smtp_inbound_prefs_t * *prefs*)

Update the receiving statistics and per-user log tables in the database for a successfully received smtp message.

Note:

The Receiving table is updated with the subnet address from which the message was received; the Log table for the user is updated to reflect the newly calculated totals of bounces or messages received.

Parameters:

con the connection across which the smtp message was received.
prefs a pointer to the user's smtp inbound mail preferences.

Returns:

This function returns no value.

Definition at line 356 of file datatier.c.

References `smtp_inbound_prefs_t::bounces`, `con_addr_subnet()`, `log_pedantic`, `smtp_session_t::mailfrom`, `mm_wipe()`, `PLACER`, `connection_t::smtp`, `st_char_get()`, `st_cmp_cs_eq()`, `st_free()`, `st_length_get()`, `stmt_exec()`, and `smtp_inbound_prefs_t::usernum`.

Referenced by `smtp_accept_message()`.

void smtp_update_transmission_stats (connection_t * con)

Update the transmission and per-user log tables in the database for a successfully sent smtp message.

Note:

The Transmitting table is updated with the timestamp of this transaction; the Log table for the user is updated to reflect the newly calculated total for messages sent.

Parameters:

con a pointer to the connection object across which the smtp message was sent.

prefs a pointer to the user's smtp inbound mail preferences.

Returns:

This function returns no value.

Definition at line 480 of file `datatier.c`.

References `log_pedantic`, `mm_wipe()`, `smtp_session_t::num_recipients`, `smtp_session_t::out_prefs`, `connection_t::smtp`, `stmt_exec()`, and `smtp_outbound_prefs_t::usernum`.

Referenced by `smtp_data_outbound()`.

magma/web/register/datatier.c File Reference

Functions for handling the new user registration process.

```
#include "magma.h"
```

Functions

- **inx_t * register_data_fetch_blocklist** (void)
 - *Fetch the blocklist for new user registration from the database.* **bool_t register_data_check_username** (**stringer_t** *username)
 - *Check to see if a username requested by a registration attempt has already been taken.* **bool_t register_data_insert_user** (**connection_t** *con, **register_session_t** *reg, **int_t** transaction, **uint64_t** *outuser)
- Insert a newly registered user into the database using information gathered by registration step #2.*
-

Detailed Description

Functions for handling the new user registration process.

Definition in file **datatier.c**.

Function Documentation

bool_t register_data_check_username (**stringer_t** * username)

Check to see if a username requested by a registration attempt has already been taken.

datatier.c

Parameters:

username the username to be checked against the database.

Returns:

true if the username is taken or false if it is not.

Definition at line 68 of file datatier.c.

References `mm_wipe()`, `res_row_next()`, `res_table_free()`, `st_char_get()`, `st_length_get()`, and `stmt_get_result()`.

Referenced by `register_business_step1()`.

inx_t* register_data_fetch_blocklist (void)

Fetch the blocklist for new user registration from the database.

HIGH: The prepared statements being used aren't valid. The queries need to be copied over and created. `register_fetch_blocklist` still needs to be defined.

Returns:

an inx holder containing the registration blocklist as a collection of managed strings.

Definition at line 21 of file datatier.c.

References `inx_alloc()`, `inx_free()`, `inx_insert()`, `log_pedantic`, `M_INX_LINKED`, `M_TYPE_UINT64`, `res_field_string()`, `res_row_next()`, `res_table_free()`, `st_free()`, `stmt_get_result()`, `multi_t::u64`, and `multi_t::val`.

Referenced by register_blocklist_update().

bool_t register_data_insert_user (connection_t * con, register_session_t * reg, int_t transaction, uint64_t * outuser)

Insert a newly registered user into the database using information gathered by registration step #2.

Note:

The following steps occur: 1. Insert a new user into the Users table, supplying username, hashed password, plan info, and quota. 2. Insert a blank entry into the Profile table for the user. 3. Insert an entry into the Folders table for the user's "Inbox" folder. 4. Insert a new entry into the Log table containing the usernum and IP address of the client request. 5. Insert a new entry into the Dispatch table for the user, configuring the spam folder, inbox, send/receive/daily send/daily receive limits, etc. 6. Insert a new entry into the Mailboxes table for the user.

Parameters:

con a pointer to the connection object of the client making the registration request.

reg the current registration session of the user to be added.

transaction a mysql transaction id for all database operations, since they all need to be committed atomically or rolled back.

outuser a pointer to a numerical id to receive the newly generated and inserted user id.

Returns:

true if the new user account was successfully created, or false on failure.

Definition at line 111 of file datatier.c.

References credential_t::auth, con_addr_presentation(), CONTIGUOUS, credential_alloc_auth(), credential_free(), magma_t::domain, log_pedantic, magma, MANAGED_T, MANAGEDBUF, meta_data_user_build_storage_keys(), mm_wipe(), ns_length_get(), register_session_t::password, register_session_t::plan, SECURE, st_alloc_opts(), st_char_get(), st_cleanup(), st_data_get(), st_free(), st_length_get(), st_merge, stmt_exec_conn(), stmt_insert_conn(), magma_t::system, and register_session_t::username.

Referenced by register_business_step2().

magma/web/statistics/datatier.c File Reference

Interface for harvesting statistics from the database for the portal statistics app.
`#include "magma.h"`

Defines

- `#define PORTAL_STATISTICS_TIMEOUT 300`

Functions

- `void statistics_init (void)`
- *Initialize the prepared sql statements used by the portal statistics page.* `bool_t statistics_refresh (void)`

Refresh all portal statistics from the database, if they haven't been updated recently. Variables

- `pthread_mutex_t portal_statistics_mutex = PTHREAD_MUTEX_INITIALIZER`
- `time_t statistics_last_updated = 0`
- `statistics_vp_t portal_stats [portal_stat_users_num_statements]`

Detailed Description

Interface for harvesting statistics from the database for the portal statistics app.

Definition in file **datatier.c**.

Define Documentation

`#define PORTAL_STATISTICS_TIMEOUT 300`

Definition at line 16 of file datatier.c.

Referenced by `statistics_refresh()`.

Function Documentation

`void statistics_init (void)`

Initialize the prepared sql statements used by the portal statistics page.
datatier.c

Returns:

This function returns no value.

Definition at line 29 of file datatier.c.

References `mm_wipe()`, `portal_stat_emails_received_today`, `portal_stat_emails_received_week`,
`portal_stat_emails_sent_today`, `portal_stat_emails_sent_week`, `portal_stat_total_users`,

portal_stat_users_checked_email_today, portal_stat_users_checked_email_week,
portal_stat_users_num_statements, portal_stat_users_registered_today, portal_stat_users_registered_week,
portal_stat_users_sent_email_today, portal_stat_users_sent_email_week, and statistics_vp_t::stmt.

Referenced by statistics_refresh().

bool_t statistics_refresh (void)

Refresh all portal statistics from the database, if they haven't been updated recently.

Returns:

true if all statistics were refreshed successfully, or false if they were not.

Definition at line 51 of file datatier.c.

References log_pedantic, mutex_lock(), mutex_unlock(), portal_statistics_mutex,
PORTAL_STATISTICS_TIMEOUT, res_field_uint64(), res_row_next(), res_table_free(), statistics_init(),
statistics_last_updated, statistics_vp_t::stmt, stmt_get_result(), and statistics_vp_t::val.

Referenced by statistics_process().

Variable Documentation

pthread_mutex_t portal_statistics_mutex = PTHREAD_MUTEX_INITIALIZER

Definition at line 18 of file datatier.c.

Referenced by statistics_refresh().

statistics_vp_t portal_stats[portal_stat_users_num_statements]

Definition at line 22 of file datatier.c.

time_t statistics_last_updated = 0

Definition at line 19 of file datatier.c.

Referenced by statistics_refresh().

magma/web/teacher/datatier.c File Reference

Allow users to train their statistical mail filter.

```
#include "magma.h"
```

Functions

- void **teacher_data_free** (**teacher_data_t** **teach*)
- *Free a spam signature object.* void **teacher_data_delete** (**teacher_data_t** **teach*)
- *Delete spam signature information from the database.* **teacher_data_t** * **teacher_data_fetch** (uint64_t *signum*)
- *Fetch information about a spam signature from the database.* void **teacher_data_save** (**teacher_data_t** **teach*)
- *Save spam signature information to the cache.* **teacher_data_t** * **teacher_data_get** (uint64_t *signum*)

Get information about a spam signature from the cache, or fall back to the database.

Detailed Description

Allow users to train their statistical mail filter.

Definition in file **datatier.c**.

Function Documentation

void teacher_data_delete (**teacher_data_t** * *teach*)

Delete spam signature information from the database.

datatier.c

HIGH: After training a signature, we should search the messages table for references to the signature being trained and update the message status flags. The UPDATE_SIGNATURE_FLAGS_ADD/UPDATE_SIGNATURE_FLAGS_REMOVE queries were created for that purpose but aren't being used right now. Note to self: retroactively brand messages as junk accordingly.

Note:

If the signature matched junk, all matching messages in the database belonging to the user will have their junk flag cleared. But if the signature didn't match junk, all matching messages in the database belonging to the user will have the junk flag added.

Parameters:

teach the spam signature to be removed from the database.

Returns:

This function returns no value.

Definition at line 43 of file datatier.c.

References `teacher_data_t::completed`, `teacher_data_t::disposition`, `log_pedantic`, `MAIL_MARK_JUNK`, `mm_wipe()`, `teacher_data_t::password`, `teacher_data_t::signature`, `teacher_data_t::signum`, `st_cleanup()`, `stmt_exec()`, `teacher_data_t::username`, and `teacher_data_t::usernum`.

Referenced by `teacher_process()`.

teacher_data_t* teacher_data_fetch (uint64_t *signum*)

Fetch information about a spam signature from the database.

Parameters:

signum the numerical id of the spam signature to be retrieved.

NULL on failure, or a pointer to a newly allocated signature teacher object on success.

Definition at line 142 of file datatier.c.

References `teacher_data_t::disposition`, `teacher_data_t::keynum`, `mm_alloc()`, `mm_wipe()`, `teacher_data_t::password`, `res_field_int8()`, `res_field_string()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `teacher_data_t::signature`, `teacher_data_t::signum`, `stmt_get_result()`, `teacher_data_free()`, `teacher_data_t::username`, and `teacher_data_t::usernum`.

Referenced by `teacher_data_get()`.

void teacher_data_free (teacher_data_t * *teach*)

Free a spam signature object.

Parameters:

teach the spam signature object to be freed.

Returns:

This function returns no value.

Definition at line 20 of file datatier.c.

References `mm_free()`, `teacher_data_t::password`, `teacher_data_t::signature`, `st_cleanup()`, and `teacher_data_t::username`.

Referenced by `teacher_data_fetch()`, `teacher_data_get()`, and `teacher_process()`.

teacher_data_t* teacher_data_get (uint64_t *signum*)

Get information about a spam signature from the cache, or fall back to the database.

Parameters:

signum the numerical id of the spam signature to be retrieved.

NULL on failure, or a pointer to a newly allocated signature teacher object on success.

Definition at line 234 of file datatier.c.

References `cache_get()`, `teacher_data_t::completed`, `serialization_t::data`, `data`, `deserialize_int32()`, `deserialize_st()`, `deserialize_uint64()`, `teacher_data_t::disposition`, `teacher_data_t::keynum`, `log_pedantic`, `mm_alloc()`, `mm_wipe()`, `teacher_data_t::password`, `PLACER`, `teacher_data_t::signature`, `teacher_data_t::signum`, `st_free()`, `teacher_data_fetch()`, `teacher_data_free()`, `teacher_data_t::username`, and `teacher_data_t::usernum`.

Referenced by `teacher_process()`.

void teacher_data_save (teacher_data_t * *teach*)

Save spam signature information to the cache.

Note:

The information will be cached for 2 hours.

Parameters:

teach the spam signature to be cached.

Returns:

This function returns no value.

Definition at line 194 of file `datatier.c`.

References `cache_set()`, `teacher_data_t::completed`, `data`, `teacher_data_t::disposition`, `teacher_data_t::keynum`, `log_pedantic`, `teacher_data_t::password`, `PLACER`, `serialize_int32()`, `serialize_st()`, `serialize_uint64()`, `teacher_data_t::signature`, `teacher_data_t::signum`, `st_free()`, `teacher_data_t::username`, and `teacher_data_t::usernum`.

Referenced by `teacher_process()`.

magma/engine/config/global/global.c File Reference

Functions for handling the global configuration.

```
#include "magma.h"
```

```
#include "keys.h"
```

Functions

- void **config_free** (void)
- *LOW: We should use use basename() and dirname() to cleanup path strings.* void **config_output_value_generic** (chr_t *prefix, chr_t *name, M_TYPE type, void *val, bool_t required)
- *Output a key name and value in a generic way.* void **config_output_value** (magma_keys_t *key)
- *Log the contents of a magma configuration option.* void **config_output_settings** (void)
- *Log a display of all active magma configuration settings.* void **config_output_help** (void)
- *Log all magma config key settings, as well as server, relay server, and cache server settings.* bool_t **config_validate_settings** (void)
- *Validate all the user configuration settings.* bool_t **config_value_set** (magma_keys_t *setting, stringer_t *value)
- *Set the value of a global config key.* bool_t **config_load_defaults** (void)
- *Load all the default values for non-required configuration options.* magma_keys_t * **config_key_lookup** (stringer_t *name)
- *Get a magma config key by name.* bool_t **config_load_file_settings** (void)
- *Load the magma configuration file specified in magma.config.file, or from the command line.* bool_t **config_load_database_settings** (void)
- *Load all magma configuration options present in the database.* bool_t **config_load_cmdline_settings** (void)

Load all magma configuration options specified by the user on the command line.
Variables

- __thread char **threadBuffer** [1024]
- magma_t **magma** = { .config.file = "magma.config" }
- bool_t **exit_and_dump** = false
- stringer_t * **cmdline_config_data** = NULL

Detailed Description

Functions for handling the global configuration.

Definition in file **global.c**.

Function Documentation

void config_free (void)

LOW: We should use use basename() and dirname() to cleanup path strings.

global.c

Free all loaded magma configuration options.

Note:

First all magma config keys will be freed, then the cache servers, relay servers, and magma servers.

Returns:

This function returns no value.

Definition at line 28 of file global.c.

References magma_t::blacklists, cache_free(), log_pedantic, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_ENUM, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma_keys, mm_free(), magma_keys_t::norm, ns_free(), NULLER, PLACER, relay_free(), servers_free(), magma_t::smtp, st_cmp_cs_eq(), st_free(), store, type(), and multi_t::type.

Referenced by config_load_defaults(), and process_stop().

magma_keys_t* config_key_lookup (stringer_t * *name*)

Get a magma config key by name.

Parameters:

name a managed string with the name of the magma config option to be looked up.

Returns:

NULL on failure or a pointer to the found magma key object on success.

Definition at line 705 of file global.c.

References magma_keys, NULLER, and st_cmp_ci_eq().

Referenced by config_load_cmdline_settings(), config_load_database_settings(), config_load_file_settings(), and config_validate_settings().

bool_t config_load_cmdline_settings (void)

Load all magma configuration options specified by the user on the command line.

Note:

Each key/value pair extracted from the database is submitted to the following logic: If a config option was loaded from the database, the key must allow it to be configurable via the database. Check to see that any key that has previously been set is allowed to be overwritten. If the key is required, it may not contain an empty value. Finally, this function sets the appropriate magma key corresponding to the config key. All leftover keys not matched to global magma keys will be configured via servers, relay, and cache server options.

Returns:

true if all database config options were parsed and evaluated successfully, or false on failure.

Definition at line 917 of file global.c.

References cache_config(), cmdline_config_data, config_key_lookup(), config_value_set(), CONSTANT, magma_keys_t::file, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_key_next(), inx_cursor_value_active(), log_critical, mt_is_empty(), magma_keys_t::name, nvp_alloc(), nvp_free(), nvp_parse(), nvp_t::pairs, relay_config(), magma_keys_t::required, servers_config(), magma_keys_t::set, multi_t::st, st_char_get(), st_cmp_ci_starts(), st_empty(), st_free(), st_length_int(), and multi_t::val.

Referenced by process_start().

bool_t config_load_database_settings (void)

Load all magma configuration options present in the database.

Note:

Each key/value pair extracted from the database is submitted to the following logic: If a config option was loaded from the database, the key must allow it to be configurable via the database. Check to see that any key that has previously been set is allowed to be overwritten. If the key is required, it may not contain an empty value. Finally, this function sets the appropriate magma key corresponding to the config key. All leftover keys not matched to global magma keys will be configured via servers, relay, and cache server options.

Returns:

true if all database config options were parsed and evaluated successfully, or false on failure.

Definition at line 829 of file global.c.

References `cache_config()`, `config_fetch_host_number()`, `config_fetch_settings()`, `config_key_lookup()`, `config_value_set()`, `CONSTANT`, `magma_keys_t::database`, `magma_t::host`, `log_critical`, `magma_keys_t::name`, `magma_t::number`, `magma_keys_t::overwrite`, `PLACER`, `relay_config()`, `magma_keys_t::required`, `res_field_block()`, `res_field_length()`, `res_row_count()`, `res_row_get()`, `res_table_free()`, `servers_config()`, `magma_keys_t::set`, `st_char_get()`, `st_cmp_ci_starts()`, `st_empty()`, and `st_length_int()`.

Referenced by `process_start()`.

bool_t config_load_defaults (void)

Load all the default values for non-required configuration options.

Returns:

true if all default magma config values were successfully loaded, or false on failure.

Definition at line 686 of file global.c.

References `config_free()`, `config_value_set()`, `log_info`, and `magma_keys`.

Referenced by `args_parse()`, and `process_start()`.

bool_t config_load_file_settings (void)

Load the magma configuration file specified in `magma.config.file`, or from the command line.

Note:

Parses the config file data into a series of name/value pairs, and make sure that for each key: If a config option was loaded from the file, the key must allow it to be configurable via the file, and If the key is required, it may not contain an empty value. Finally, this function sets the appropriate magma key corresponding to the config key. All leftover keys not matched to global magma keys will be configured via servers, relay, and cache server options.

Returns:

true if all config file options were parsed and evaluated successfully, or false on failure.

Definition at line 727 of file global.c.

References `cache_config()`, `magma_t::config`, `config_key_lookup()`, `config_value_set()`, `CONSTANT`, `magma_keys_t::file`, `magma_t::file`, `file_load()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_key_next()`,

inx_cursor_value_active(), log_critical, mt_is_empty(), magma_keys_t::name, nvp_alloc(), nvp_free(), nvp_parse(), nvp_t::pairs, relay_config(), magma_keys_t::required, servers_config(), magma_keys_t::set, multi_t::st, st_char_get(), st_cmp_ci_starts(), st_empty(), st_free(), st_length_int(), and multi_t::val.

Referenced by process_start().

void config_output_help (void)

Log all magma config key settings, as well as server, relay server, and cache server settings.

Returns:

This function returns no value.

Definition at line 294 of file global.c.

References multi_t::bl, cache_output_help(), config_output_value_generic(), log_info, log_options, M_LOG_FILE_DISABLE, M_LOG_FUNCTION_DISABLE, M_LOG_LINE_DISABLE, M_LOG_LINE_FEED_DISABLE, M_LOG_STACK_TRACE_DISABLE, M_LOG_TIME_DISABLE, magma_keys, magma_keys_t::norm, relay_output_help(), magma_keys_t::required, servers_output_help(), and multi_t::val.

Referenced by args_parse().

void config_output_settings (void)

Log a display of all active magma configuration settings.

Returns:

This function returns no value.

Definition at line 267 of file global.c.

References cache_output_settings(), magma_t::config, config_output_value(), magma_t::file, magma_t::host, log_info, magma_keys, magma_t::name, magma_t::number, relay_output_settings(), and servers_output_settings().

Referenced by config_validate_settings().

void config_output_value (magma_keys_t * key)

Log the contents of a magma configuration option.

Parameters:

key a pointer to the magma configuration key to be dumped.

Returns:

This function returns no value.

Definition at line 195 of file global.c.

References magma_t::blacklists, log_info, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma_keys_t::name, magma_keys_t::norm, ns_empty(), NULLER, PLACER, magma_t::smtp, st_char_get(), st_cmp_cs_eq(), st_empty(), st_length_int(), magma_keys_t::store, and multi_t::type.

Referenced by config_output_settings().

void config_output_value_generic (chr_t * *prefix*, chr_t * *name*, M_TYPE *type*, void * *val*, bool_t *required*)

Output a key name and value in a generic way.

Parameters:

prefix a pointer to a null-terminated string containing an optional prefix to be printed before the supplied key name.

name a pointer to a null-terminated string containing the name of the key being output.

type an M_TYPE value specifying the multi-type of the key value.

val a void pointer to the value of the specified key, which will be printed in accordance with the supplied multi-type.

required a boolean value specifying whether the specified key is a required configuration option.

Returns:

This function returns no value.

Definition at line 83 of file global.c.

References magma_t::blacklists, CONSTANT, EMPTY, HTTP, IMAP, log_info, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_ENUM, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, MOLTEN, ns_empty(), NULLER, PLACER, POP, SMTP, magma_t::smtp, SSL_PORT, st_char_get(), st_cmp_cs_eq(), st_empty(), st_length_int(), SUBMISSION, and TCP_PORT.

Referenced by cache_output_help(), config_output_help(), relay_output_help(), and servers_output_help().

bool_t config_validate_settings (void)

Validate all the user configuration settings.

Note:

The steps are as follows: 1. Check to see that all required keys have been set. 2. If magma.iface.virus.available is set, magma.iface.virus.signatures must be set. 3. Make sure 10 <= magma.iface.cache.retry <= 86400 4. Make sure 1 <= magma.iface.cache.timeout <= 3600 5. Make sure magma.iface.cache.retry <= magma.iface.cache.timeout 6. Make sure 40 <= magma.smtp.wrap_line_length <= 65535 7. Make sure 8 <= magma.smtp.recipient_limit <= 32768 8. Make sure 16 <= magma.smtp.relay_limit 9. Make sure 16384 <= system_limit_max(RLIMIT_STACK) 10. If magma.system.daemonize is set, make sure magma.output.file is not false 11. If magma.output.file is enabled, magma.output.path must be set. 12. If **magma.dkim.enabled** is set, then magma.dkim.domain, magma.dkim.selector, and magma.dkim.privkey must all be set. 13. Validate all the configured magma servers, relay servers, and cache servers. 14. Check all config key filenames and directories to ensure that they exist and are accessible. 15. Make sure magma.admin.contact and point to valid email addresses, if they are specified. 16. If magma.config.output_config is set, dump the current configuration.

Definition at line 332 of file global.c.

References magma_t::abuse, magma_t::admin, magma_t::cache, cache_validate(), magma_t::config, CONFIG_CHECK_DIR_READABLE, CONFIG_CHECK_DIR_READWRITE, CONFIG_CHECK_FILE_READABLE, config_key_lookup(), config_output_settings(), magma_t::contact, contact_business_valid_email(), magma_t::daemonize, magma_t::dkim, magma_t::domain, magma_t::enabled, exit_and_dump, magma_t::file, magma_t::http, magma_t::iface, magma_t::library, log_critical, magma_keys,

magma_t::output, magma_t::output_config, magma_t::path, PLACER, magma_t::privkey, magma_t::recipient_limit, magma_t::relay_limit, relay_validate(), magma_t::root_directory, magma_t::selector, servers_validate(), magma_keys_t::set, magma_t::smtp, magma_t::spool, magma_t::system, system_limit_max(), magma_t::thread_stack_size, magma_t::virus, and magma_t::wrap_line_length.

Referenced by process_start().

bool_t config_value_set (magma_keys_t * setting, stringer_t * value)

Set the value of a global config key.

Note:

This function will also free the value of the global config key if it has already previously been set.

Parameters:

setting a pointer to the global key to have its value adjusted.

value a managed string containing the new key value, or if NULL, the key's default value will be used.

Returns:

true if the specified key's value was set successfully, or false on failure.

LOW: Realtime blacklist domains are handled using custom code because we don't yet have a generic type to store lists.

Definition at line 500 of file global.c.

References multi_t::binary, magma_t::blacklists, CONSTANT, CONTIGUOUS, HEAP, multi_t::i32, multi_t::i64, multi_t::i8, int32_conv_st(), int64_conv_st(), int8_conv_st(), log_critical, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, MAGMA_BLACKLIST_INSTANCES, MANAGED_T, magma_keys_t::name, magma_keys_t::norm, multi_t::ns, ns_dupe(), ns_empty(), ns_free(), ns_import(), NULLER, PLACER, magma_t::smtp, smtp_add_bypass_entry(), multi_t::st, st_char_get(), st_cmp_ci_eq(), st_cmp_cs_eq(), st_dupe_opts(), st_empty(), st_free(), st_length_get(), magma_keys_t::store, multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, uint16_conv_st(), uint32_conv_st(), uint64_conv_st(), uint8_conv_st(), and multi_t::val.

Referenced by config_load_cmdline_settings(), config_load_database_settings(), config_load_defaults(), and config_load_file_settings().

Variable Documentation

stringer_t* cmdline_config_data = NULL

Definition at line 19 of file global.c.

Referenced by args_parse(), and config_load_cmdline_settings().

bool_t exit_and_dump = false

Definition at line 18 of file global.c.

Referenced by args_parse(), config_validate_settings(), and process_start().

magma_t magma = { .config.file = "magma.config" }

Definition at line 17 of file global.c.

Referenced by args_parse(), cache_alloc(), cache_free(), cache_output_settings(), cache_start(), cache_stop(), cache_validate(), con_init_network_buffer(), config_fetch_host_number(), config_fetch_settings(), contact_business(), credential_alloc_auth(), credential_username(), dkim_create(), http_body(), http_content_load_fonts(), http_content_refresh(), http_content_start(), http_data_value_parse(), http_load_file(), http_parse_header(), http_parse_method(), http_print_301(), http_response(), http_response_connection(), http_response_header(), http_response_options(), imap_append_message(), imap_command_parser(), lib_load(), lib_unload(), log_internal(), log_rotate(), log_start(), mail_add_forward_headers(), mail_add_outbound_headers(), mail_create_directory(), mail_db_insert_duplicate_message(), mail_db_insert_message(), mail_message_cleanup(), mail_message_path(), mm_sec_start(), net_init(), net_listen(), portal_endpoint(), portal_endpoint_error(), portal_endpoint_response(), portal_process(), portal_upload(), process_start(), process_stop(), queue_init(), queue_shutdown(), queue_signal(), rand_start(), rand_thread_start(), register_business_step2(), register_captcha_random_font(), register_data_insert_user(), relay_alloc(), relay_counter(), relay_free(), relay_output_settings(), relay_validate(), servers_alloc(), servers_encryption_start(), servers_encryption_stop(), servers_free(), servers_get_by_socket(), servers_get_count_using_port(), servers_network_start(), servers_network_stop(), servers_output_settings(), servers_validate(), sess_create(), sess_get(), sess_token(), smtp_accept_message(), smtp_add_bypass_entry(), smtp_bounce(), smtp_bypass_check(), smtp_check_rbl(), smtp_client_connect(), smtp_client_send_helo(), smtp_data(), smtp_ehlo(), smtp_mail_from(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(), smtp_parse_rcpt_to(), smtp_rcpt_to(), spf_start(), spf_stop(), spool_path(), sql_open(), sql_start(), sql_stop(), ssl_start(), st_alloc_opts(), st_realloc(), stmt_start(), stmt_stop(), system_change_root_directory(), system_fork_daemon(), system_init_core_dumps(), system_init_impersonation(), system_init_resource_limits(), tank_start(), thread_launch(), virus_check(), virus_engine_create(), virus_engine_refresh(), virus_sigs_total(), virus_start(), and virus_stop().

__thread char threadBuffer[1024]

Definition at line 16 of file global.c.

magma/engine/config/global/global.h File Reference

The global configuration structure used for overall system settings, and functions to initialize it at startup and free it at shutdown.

Data Structures

- struct **magma_keys_t**
- struct **magma_t**

Functions

- **uint64_t config_fetch_host_number** (void)
- *datatier.c* **table_t * config_fetch_settings** (void)
- *Retrieve the entire collection of configuration key/value pairs from the database.* void **config_free** (void)
- *global.c* **magma_keys_t * config_key_lookup** (stringer_t *name)
- *Get a magma config key by name.* **bool_t config_load_database_settings** (void)
- *Load all magma configuration options present in the database.* **bool_t config_load_cmdline_settings** (void)
- *Load all magma configuration options specified by the user on the command line.* **bool_t config_load_defaults** (void)
- *Load all the default values for non-required configuration options.* **bool_t config_load_file_settings** (void)
- *Load the magma configuration file specified in magma.config.file, or from the command line.* void **config_output_help** (void)
- *Log all magma config key settings, as well as server, relay server, and cache server settings.* void **config_output_settings** (void)
- *Log a display of all active magma configuration settings.* void **config_output_value_generic** (chr_t *prefix, chr_t *name, M_TYPE type, void *val, **bool_t** required)
- *Output a key name and value in a generic way.* void **config_output_value** (magma_keys_t *key)
- *Log the contents of a magma configuration option.* **bool_t config_validate_settings** (void)
- *Validate all the user configuration settings.* **bool_t config_value_set** (magma_keys_t *setting, stringer_t *value)

Set the value of a global config key.

Detailed Description

The global configuration structure used for overall system settings, and functions to initialize it at startup and free it at shutdown.

Definition in file **global.h**.

Function Documentation

uint64_t config_fetch_host_number (void)

datatier.c
datatier.c

Returns:

this host's numerical identifier, or 0 on failure.

Definition at line 19 of file *datatier.c*.

References magma_t::host, log_error, magma, mm_wipe(), magma_t::name, ns_length_get(), res_field_uint64(), res_row_next(), res_table_free(), and stmt_get_result().

Referenced by config_load_database_settings().

table_t* config_fetch_settings (void)

Retrieve the entire collection of configuration key/value pairs from the database.

Parameters:

none This function accepts no parameters.

Returns:

NULL on failure, or a database table of configuration key/value pairs on success.

Definition at line 45 of file datatier.c.

References magma_t::host, magma, mm_wipe(), magma_t::name, ns_length_get(), and stmt_get_result().

Referenced by config_load_database_settings().

void config_free (void)

global.c

global.c

Free all loaded magma configuration options.

Note:

First all magma config keys will be freed, then the cache servers, relay servers, and magma servers.

Returns:

This function returns no value.

Definition at line 28 of file global.c.

References magma_t::blacklists, cache_free(), log_pedantic, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_ENUM, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma_keys, mm_free(), magma_keys_t::norm, ns_free(), NULLER, PLACER, relay_free(), servers_free(), magma_t::smtp, st_cmp_cs_eq(), st_free(), store, type(), and multi_t::type.

Referenced by config_load_defaults(), and process_stop().

magma_keys_t* config_key_lookup (stringer_t * *name*)

Get a magma config key by name.

Parameters:

name a managed string with the name of the magma config option to be looked up.

Returns:

NULL on failure or a pointer to the found magma key object on success.

Definition at line 705 of file global.c.

References magma_keys, NULLER, and st_cmp_ci_eq().

Referenced by `config_load_cmdline_settings()`, `config_load_database_settings()`, `config_load_file_settings()`, and `config_validate_settings()`.

bool_t config_load_cmdline_settings (void)

Load all magma configuration options specified by the user on the command line.

Note:

Each key/value pair extracted from the database is submitted to the following logic: If a config option was loaded from the database, the key must allow it to be configurable via the database. Check to see that any key that has previously been set is allowed to be overwritten. If the key is required, it may not contain an empty value. Finally, this function sets the appropriate magma key corresponding to the config key. All leftover keys not matched to global magma keys will be configured via servers, relay, and cache server options.

Returns:

true if all database config options were parsed and evaluated successfully, or false on failure.

Definition at line 917 of file `global.c`.

References `cache_config()`, `cmdline_config_data`, `config_key_lookup()`, `config_value_set()`, `CONSTANT`, `magma_keys_t::file`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_key_next()`, `inx_cursor_value_active()`, `log_critical`, `mt_is_empty()`, `magma_keys_t::name`, `nvp_alloc()`, `nvp_free()`, `nvp_parse()`, `nvp_t::pairs`, `relay_config()`, `magma_keys_t::required`, `servers_config()`, `magma_keys_t::set`, `multi_t::st`, `st_char_get()`, `st_cmp_ci_starts()`, `st_empty()`, `st_free()`, `st_length_int()`, and `multi_t::val`.

Referenced by `process_start()`.

bool_t config_load_database_settings (void)

Load all magma configuration options present in the database.

Note:

Each key/value pair extracted from the database is submitted to the following logic: If a config option was loaded from the database, the key must allow it to be configurable via the database. Check to see that any key that has previously been set is allowed to be overwritten. If the key is required, it may not contain an empty value. Finally, this function sets the appropriate magma key corresponding to the config key. All leftover keys not matched to global magma keys will be configured via servers, relay, and cache server options.

Returns:

true if all database config options were parsed and evaluated successfully, or false on failure.

Definition at line 829 of file `global.c`.

References `cache_config()`, `config_fetch_host_number()`, `config_fetch_settings()`, `config_key_lookup()`, `config_value_set()`, `CONSTANT`, `magma_keys_t::database`, `magma_t::host`, `log_critical`, `magma_keys_t::name`, `magma_t::number`, `magma_keys_t::overwrite`, `PLACER`, `relay_config()`, `magma_keys_t::required`, `res_field_block()`, `res_field_length()`, `res_row_count()`, `res_row_get()`, `res_table_free()`, `servers_config()`, `magma_keys_t::set`, `st_char_get()`, `st_cmp_ci_starts()`, `st_empty()`, and `st_length_int()`.

Referenced by `process_start()`.

bool_t config_load_defaults (void)

Load all the default values for non-required configuration options.

Returns:

true if all default magma config values were successfully loaded, or false on failure.

Definition at line 686 of file global.c.

References config_free(), config_value_set(), log_info, and magma_keys.

Referenced by args_parse(), and process_start().

bool_t config_load_file_settings (void)

Load the magma configuration file specified in magma.config.file, or from the command line.

Note:

Parses the config file data into a series of name/value pairs, and make sure that for each key: If a config option was loaded from the file, the key must allow it to be configurable via the file, and If the key is required, it may not contain an empty value. Finally, this function sets the appropriate magma key corresponding to the config key. All leftover keys not matched to global magma keys will be configured via servers, relay, and cache server options.

Returns:

true if all config file options were parsed and evaluated successfully, or false on failure.

Definition at line 727 of file global.c.

References cache_config(), magma_t::config, config_key_lookup(), config_value_set(), CONSTANT, magma_keys_t::file, magma_t::file, file_load(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_key_next(), inx_cursor_value_active(), log_critical, mt_is_empty(), magma_keys_t::name, nvp_alloc(), nvp_free(), nvp_parse(), nvp_t::pairs, relay_config(), magma_keys_t::required, servers_config(), magma_keys_t::set, multi_t::st, st_char_get(), st_cmp_ci_starts(), st_empty(), st_free(), st_length_int(), and multi_t::val.

Referenced by process_start().

void config_output_help (void)

Log all magma config key settings, as well as server, relay server, and cache server settings.

Returns:

This function returns no value.

Definition at line 294 of file global.c.

References multi_t::bl, cache_output_help(), config_output_value_generic(), log_info, log_options, M_LOG_FILE_DISABLE, M_LOG_FUNCTION_DISABLE, M_LOG_LINE_DISABLE, M_LOG_LINE_FEED_DISABLE, M_LOG_STACK_TRACE_DISABLE, M_LOG_TIME_DISABLE, magma_keys, magma_keys_t::norm, relay_output_help(), magma_keys_t::required, servers_output_help(), and multi_t::val.

Referenced by args_parse().

void config_output_settings (void)

Log a display of all active magma configuration settings.

Returns:

This function returns no value.

Definition at line 267 of file global.c.

References `cache_output_settings()`, `magma_t::config`, `config_output_value()`, `magma_t::file`, `magma_t::host`, `log_info`, `magma_keys`, `magma_t::name`, `magma_t::number`, `relay_output_settings()`, and `servers_output_settings()`.

Referenced by `config_validate_settings()`.

void config_output_value (magma_keys_t * key)

Log the contents of a magma configuration option.

Parameters:

key a pointer to the magma configuration key to be dumped.

Returns:

This function returns no value.

Definition at line 195 of file global.c.

References `magma_t::blacklists`, `log_info`, `log_pedantic`, `M_TYPE_BOOLEAN`, `M_TYPE_INT16`, `M_TYPE_INT32`, `M_TYPE_INT64`, `M_TYPE_INT8`, `M_TYPE_NULLER`, `M_TYPE_STRINGER`, `M_TYPE_UINT16`, `M_TYPE_UINT32`, `M_TYPE_UINT64`, `M_TYPE_UINT8`, `magma_keys_t::name`, `magma_keys_t::norm`, `ns_empty()`, `NULLER`, `PLACER`, `magma_t::smtp`, `st_char_get()`, `st_cmp_cs_eq()`, `st_empty()`, `st_length_int()`, `magma_keys_t::store`, and `multi_t::type`.

Referenced by `config_output_settings()`.

void config_output_value_generic (chr_t * prefix, chr_t * name, M_TYPE type, void * val, bool_t required)

Output a key name and value in a generic way.

Parameters:

prefix a pointer to a null-terminated string containing an optional prefix to be printed before the supplied key name.

name a pointer to a null-terminated string containing the name of the key being output.

type an `M_TYPE` value specifying the multi-type of the key value.

val a void pointer to the value of the specified key, which will be printed in accordance with the supplied multi-type.

required a boolean value specifying whether the specified key is a required configuration option.

Returns:

This function returns no value.

Definition at line 83 of file global.c.

References `magma_t::blacklists`, `CONSTANT`, `EMPTY`, `HTTP`, `IMAP`, `log_info`, `log_pedantic`, `M_TYPE_BOOLEAN`, `M_TYPE_ENUM`, `M_TYPE_INT16`, `M_TYPE_INT32`, `M_TYPE_INT64`, `M_TYPE_INT8`, `M_TYPE_NULLER`, `M_TYPE_STRINGER`, `M_TYPE_UINT16`, `M_TYPE_UINT32`, `M_TYPE_UINT64`, `M_TYPE_UINT8`, `MOLTEN`, `ns_empty()`, `NULLER`, `PLACER`, `POP`, `SMTP`, `magma_t::smtp`, `SSL_PORT`, `st_char_get()`, `st_cmp_cs_eq()`, `st_empty()`, `st_length_int()`, `SUBMISSION`, and `TCP_PORT`.

Referenced by `cache_output_help()`, `config_output_help()`, `relay_output_help()`, and `servers_output_help()`.

bool_t config_validate_settings (void)

Validate all the user configuration settings.

Note:

The steps are as follows: 1. Check to see that all required keys have been set. 2. If `magma.iface.virus.available` is set, `magma.iface.virus.signatures` must be set. 3. Make sure `10 <= magma.iface.cache.retry <= 86400` 4. Make sure `1 <= magma.iface.cache.timeout <= 3600` 5. Make sure `magma.iface.cache.retry <= magma.iface.cache.timeout` 6. Make sure `40 <= magma.smtp.wrap_line_length <= 65535` 7. Make sure `8 <= magma.smtp.recipient_limit <= 32768` 8. Make sure `16 <= magma.smtp.relay_limit` 9. Make sure `16384 <= system_limit_max(RLIMIT_STACK)` 10. If `magma.system.daemonize` is set, make sure `magma.output.file` is not false 11. If `magma.output.file` is enabled, `magma.output.path` must be set. 12. If **`magma.dkim.enabled`** is set, then `magma.dkim.domain`, `magma.dkim.selector`, and `magma.dkim.privkey` must all be set. 13. Validate all the configured magma servers, relay servers, and cache servers. 14. Check all config key filenames and directories to ensure that they exist and are accessible. 15. Make sure `magma.admin.contact` and point to valid email addresses, if they are specified. 16. If `magma.config.output_config` is set, dump the current configuration.

Definition at line 332 of file `global.c`.

References `magma_t::abuse`, `magma_t::admin`, `magma_t::cache`, `cache_validate()`, `magma_t::config`, `CONFIG_CHECK_DIR_READABLE`, `CONFIG_CHECK_DIR_READWRITE`, `CONFIG_CHECK_FILE_READABLE`, `config_key_lookup()`, `config_output_settings()`, `magma_t::contact`, `contact_business_valid_email()`, `magma_t::daemonize`, `magma_t::dkim`, `magma_t::domain`, `magma_t::enabled`, `exit_and_dump`, `magma_t::file`, `magma_t::http`, `magma_t::iface`, `magma_t::library`, `log_critical`, `magma_keys`, `magma_t::output`, `magma_t::output_config`, `magma_t::path`, `PLACER`, `magma_t::privkey`, `magma_t::recipient_limit`, `magma_t::relay_limit`, `relay_validate()`, `magma_t::root_directory`, `magma_t::selector`, `servers_validate()`, `magma_keys_t::set`, `magma_t::smtp`, `magma_t::spool`, `magma_t::system`, `system_limit_max()`, `magma_t::thread_stack_size`, `magma_t::virus`, and `magma_t::wrap_line_length`.

Referenced by `process_start()`.

bool_t config_value_set (magma_keys_t * setting, stringer_t * value)

Set the value of a global config key.

Note:

This function will also free the value of the global config key if it has already previously been set.

Parameters:

setting a pointer to the global key to have its value adjusted.

value a managed string containing the new key value, or if NULL, the key's default value will be used.

Returns:

true if the specified key's value was set successfully, or false on failure.

LOW: Realtime blacklist domains are handled using custom code because we don't yet have a generic type to store lists.

Definition at line 500 of file `global.c`.

References `multi_t::binary`, `magma_t::blacklists`, `CONSTANT`, `CONTIGUOUS`, `HEAP`, `multi_t::i32`, `multi_t::i64`, `multi_t::i8`, `int32_conv_st()`, `int64_conv_st()`, `int8_conv_st()`, `log_critical`, `M_TYPE_BOOLEAN`,

M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, MAGMA_BLACKLIST_INSTANCES, MANAGED_T, magma_keys_t::name, magma_keys_t::norm, multi_t::ns, ns_dupe(), ns_empty(), ns_free(), ns_import(), NULLER, PLACER, magma_t::smtp, smtp_add_bypass_entry(), multi_t::st, st_char_get(), st_cmp_ci_eq(), st_cmp_cs_eq(), st_dupe_opts(), st_empty(), st_free(), st_length_get(), magma_keys_t::store, multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, uint16_conv_st(), uint32_conv_st(), uint64_conv_st(), uint8_conv_st(), and multi_t::val.

Referenced by config_load_cmdline_settings(), config_load_database_settings(), config_load_defaults(), and config_load_file_settings().

magma/engine/config/relay/relay.c File Reference

Functions for handling relay server instance configurations.

```
#include "magma.h"
```

```
#include "keys.h"
```

Functions

- void **relay_counter** (void)
- *Update the relay server counters to reflect the number of configured standard and premium relay servers.* void **relay_free** (void)
- *Free magma's relay server configuration options.* **relay_t** * **relay_alloc** (uint32_t **number**)
- *Allocate and initialize a magma server relay object, or return it if it already exists.* **bool_t** **relay_validate** (void)
- *Validate all of the configured magma relay server instances.* void **relay_output_settings** (void)
- **bool_t** **relay_set_value** (**relay_keys_t** *setting, **relay_t** *relay, **stringer_t** *value)
- *Set a specified key for a relay server configuration entry.* **bool_t** **relay_config** (**stringer_t** *name, **stringer_t** *value)
- *Set the value of a relay server configuration entry by name.* void **relay_output_help** (void)

Log all relay key information to be returned via "magma -h".

Detailed Description

Functions for handling relay server instance configurations.

\$Author\$ \$Author\$ \$Revision\$

Definition in file **relay.c**.

Function Documentation

relay_t* **relay_alloc** (uint32_t *number*)

Allocate and initialize a magma server relay object, or return it if it already exists.

Parameters:

number a zero-based index into the global relay server table where the entry should be created and/or queried.

Returns:

NULL on failure, or a pointer to the requested relay server object on success.

Definition at line 91 of file relay.c.

References magma_t::host, log_critical, magma, mm_alloc(), magma_t::relay, relay_keys, and relay_set_value().

Referenced by relay_config().

bool_t **relay_config** (**stringer_t** * *name*, **stringer_t** * *value*)

Set the value of a relay server configuration entry by name.

Note:

This function is designed to operate on lines from a configuration source, like a file or database. When a named server is referenced for the first time, a new relay server configuration instance will be allocated.

Parameters:

name a managed string containing the human-readable relay server configuration parameter to be set.
value a managed string containing the new value of the relay server config key.

Returns:

true if the value was successfully set, or false on failure.

Definition at line 453 of file relay.c.

References bracket_extract_pl(), CONSTANT, log_critical, MAGMA_RELAY_INSTANCES, NULLER, pl_char_get(), pl_empty(), pl_length_get(), PLACER, placer_t, relay_alloc(), relay_keys, relay_set_value(), st_char_get(), st_cmp_ci_eq(), st_cmp_ci_starts(), st_length_get(), st_length_int(), and uint32_conv_bl().

Referenced by config_load_cmdline_settings(), config_load_database_settings(), and config_load_file_settings().

void relay_counter (void)

Update the relay server counters to reflect the number of configured standard and premium relay servers.

Returns:

This function returns no value.

Definition at line 21 of file relay.c.

References magma_t::count, magma_t::host, magma, MAGMA_RELAY_INSTANCES, magma_t::premium, and magma_t::relay.

Referenced by relay_set_value().

void relay_free (void)

Free magma's relay server configuration options.

Note:

This function assumes all worker threads have finished, and should only be called during the normal shutdown process.

Parameters:

This function returns no value.

Definition at line 44 of file relay.c.

References magma_t::host, log_pedantic, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_ENUM, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_RELAY_INSTANCES, mm_free(), relay_keys_t::norm, ns_empty(), ns_free(), relay_keys_t::offset, magma_t::relay, relay_keys, st_empty(), st_free(), type(), and multi_t::type.

Referenced by config_free().

void relay_output_help (void)

Log all relay key information to be returned via "magma -h".

Returns:

This function returns no value.

Definition at line 513 of file relay.c.

References multi_t::bl, config_output_value_generic(), log_info, log_options, M_LOG_FILE_DISABLE, M_LOG_FUNCTION_DISABLE, M_LOG_LINE_DISABLE, M_LOG_LINE_FEED_DISABLE, M_LOG_STACK_TRACE_DISABLE, M_LOG_TIME_DISABLE, relay_keys_t::norm, relay_keys, relay_keys_t::required, and multi_t::val.

Referenced by config_output_help().

void relay_output_settings (void)

Definition at line 233 of file relay.c.

References magma_t::host, log_info, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_RELAY_INSTANCES, relay_keys_t::norm, ns_empty(), relay_keys_t::offset, magma_t::relay, relay_keys, st_char_get(), st_empty(), st_length_int(), multi_t::type, and type().

Referenced by config_output_settings().

bool_t relay_set_value (relay_keys_t * setting, relay_t * relay, stringer_t * value)

Set a specified key for a relay server configuration entry.

Note:

This function will allocate space for a copy of the key value, and return an error if it is not able to convert it to the proper key data type.

Parameters:

setting a pointer to the relay server configuration key to be set.

relay a pointer to the relay server configuration entry to be adjusted.

value a managed string containing the new value of the config key.

Returns:

true if the value was successfully set, or false on error.

Definition at line 308 of file relay.c.

References multi_t::binary, CONSTANT, CONTIGUOUS, HEAP, multi_t::i32, multi_t::i64, multi_t::i8, int32_conv_st(), int64_conv_st(), int8_conv_st(), log_critical, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, MANAGED_T, relay_keys_t::name, relay_keys_t::norm, multi_t::ns, ns_dupe(), ns_empty(), ns_free(), ns_import(), relay_keys_t::offset, relay_counter(), multi_t::st, st_char_get(), st_cmp_ci_eq(), st_dupe_opts(), st_empty(), st_free(), st_length_get(), type(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, uint16_conv_st(), uint32_conv_st(), uint64_conv_st(), uint8_conv_st(), and multi_t::val.

Referenced by relay_alloc(), and relay_config().

bool_t relay_validate (void)

Validate all of the configured magma relay server instances.

Iterates through all of the server structures and verifies that each was supplied with valid values for all of its required keys. This function also applies the application specific validation rules. Specifically it verifies that only one server structure has been configured to use a given port number.

Note:

This makes set that all required config keys have been set, and that at least one host has been configured.

Returns:

true if all relay servers have been validated, or false on failure.

Definition at line 120 of file relay.c.

References magma_t::host, multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, log_critical, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_RELAY_INSTANCES, relay_keys_t::norm, ns_empty(), relay_keys_t::offset, magma_t::relay, relay_keys, st_empty(), multi_t::type, type(), multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by config_validate_settings().

magma/servers/smtp/relay.c File Reference

Functions to relay messages via SMTP to the outbound mail server.

```
#include "magma.h"
```

Functions

- void **smtp_client_close** (**client_t** *client)
- *Issue an smtp client QUIT command.* **client_t** * **smtp_client_connect** (**int_t** premium)
- *Connect to a randomly selected mail relay server, and wait for a successful banner message.* **int_t** **smtp_client_send_helo** (**client_t** *client)
- *Issue a EHLO command to an smtp server, or fall back to HELO, and wait for a successful response.* **int_t** **smtp_client_send_mailfrom** (**client_t** *client, **stringer_t** *mailfrom, **size_t** send_size)
- *Issue a MAIL FROM command to an smtp server, and wait for a successful response.* **int_t** **smtp_client_send_nullfrom** (**client_t** *client)
- *Issue a MAIL FROM command to an smtp server with a null sender, and wait for a successful response.* **int_t** **smtp_client_send_rcptto** (**client_t** *client, **stringer_t** *rcptto)
- *Issue a RCPT TO command to an smtp server, and wait for a successful response.* **int_t** **smtp_client_send_data** (**client_t** *client, **stringer_t** *message)

Issue a DATA command to an smtp server, and wait for a successful response.

Detailed Description

Functions to relay messages via SMTP to the outbound mail server.

Definition in file **relay.c**.

Function Documentation

void smtp_client_close (**client_t** * *client*)

Issue an smtp client QUIT command.

relay.c

Parameters:

client a pointer to the smtp client session to be closed.

Returns:

This function returns no value.

Definition at line 20 of file relay.c.

References `client_close()`, `client_read_line()`, `client_write()`, and `PLACER`.

Referenced by `portal_smtp_relay_message()`, `smtp_bounce()`, `smtp_forward_message()`, `smtp_relay_message()`, `smtp_reply()`, and `smtp_send_message()`.

client_t * **smtp_client_connect** (**int_t** *premium*)

Connect to a randomly selected mail relay server, and wait for a successful banner message.

Parameters:

premium if set, a premium relay will be selected instead of a standard one.

Returns:

NULL on failure or a pointer to the newly established network client object connected to a mail relay on success.

Definition at line 36 of file relay.c.

References client_t::buffer, client_close(), client_connect(), client_read_line(), client_secure(), magma_t::count, magma_t::host, client_t::line, log_pedantic, magma, MAGMA_RELAY_INSTANCES, relay_t::name, net_set_timeout(), relay_t::port, magma_t::premium, rand_get_uint32(), magma_t::relay, relay_t::secure, client_t::sockd, st_char_get(), st_length_get(), st_length_int(), and magma_t::timeout.

Referenced by portal_smtp_relay_message(), smtp_bounce(), smtp_forward_message(), smtp_relay_message(), smtp_reply(), and smtp_send_message().

int_t smtp_client_send_data (client_t * *client*, stringer_t * *message*)

Issue a DATA command to an smtp server, and wait for a successful response.

Parameters:

client a pointer to the network client to issue the DATA command.

message a pointer to a managed string containing the body of the message to be sent.

Returns:

-2 if the remote server rejected the command, -1 on general network failure, or 1 on success.

Definition at line 238 of file relay.c.

References client_t::buffer, client_read_line(), client_write(), length, client_t::line, log_pedantic, PLACER, st_char_get(), st_length_get(), and st_length_int().

Referenced by portal_smtp_relay_message(), smtp_bounce(), smtp_forward_message(), smtp_relay_message(), smtp_reply(), and smtp_send_message().

int_t smtp_client_send_helo (client_t * *client*)

Issue a EHLO command to an smtp server, or fall back to HELO, and wait for a successful response.

Parameters:

client a pointer to the network client to issue the remote command.

Returns:

-1 on failure or 1 on success.

Definition at line 115 of file relay.c.

References client_t::buffer, client_print(), client_read_line(), magma_t::host, client_t::line, log_pedantic, magma, magma_t::name, st_char_get(), and st_length_get().

Referenced by portal_smtp_relay_message(), smtp_bounce(), smtp_forward_message(), smtp_relay_message(), smtp_reply(), and smtp_send_message().

int_t smtp_client_send_mailfrom (client_t * *client*, stringer_t * *mailfrom*, size_t *send_size*)

Issue a MAIL FROM command to an smtp server, and wait for a successful response.

Parameters:

client a pointer to the network client to issue the MAIL FROM command.

mailfrom a pointer to a managed string containing the address parameter for the MAIL FROM command.

send_size if greater than 0, specify the optional SIZE parameter to the MAIL FROM command.

Returns:

-2 if the remote server rejected the command, -1 on general network failure, or 1 on success.

Definition at line 165 of file relay.c.

References `client_t::buffer`, `client_print()`, `client_read_line()`, `client_t::line`, `log_pedantic`, `st_char_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by `portal_smtp_relay_message()`, `smtp_relay_message()`, and `smtp_send_message()`.

int_t smtp_client_send_nullfrom (client_t * *client*)

Issue a MAIL FROM command to an smtp server with a null sender, and wait for a successful response.

Note:

Null senders are used when the sender is not concerned about being notified about bounced messages.

Parameters:

client a pointer to the network client to issue the MAIL FROM command.

Returns:

-2 if the remote server rejected the command, -1 on general network failure, or 1 on success.

Definition at line 192 of file relay.c.

References `client_t::buffer`, `client_print()`, `client_read_line()`, `client_t::line`, `log_pedantic`, `st_char_get()`, and `st_length_int()`.

Referenced by `smtp_bounce()`, `smtp_forward_message()`, and `smtp_reply()`.

int_t smtp_client_send_rcptto (client_t * *client*, stringer_t * *rcptto*)

Issue a RCPT TO command to an smtp server, and wait for a successful response.

Parameters:

client a pointer to the network client to issue the RCPT TO command.

rcptto a pointer to a managed string containing the recipient address parameter for the RCPT TO command.

Returns:

-2 if the remote server rejected the command, -1 on general network failure, or 1 on success.

Definition at line 215 of file relay.c.

References `client_t::buffer`, `client_print()`, `client_read_line()`, `client_t::line`, `log_pedantic`, `st_char_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by `portal_smtp_relay_message()`, `smtp_bounce()`, `smtp_forward_message()`, `smtp_relay_message()`, `smtp_reply()`, and `smtp_send_message()`.

magma/engine/config/relay/relay.h File Reference

The types and functions involved with creating and accessing the relay structures.

Data Structures

- struct **relay_keys_t**
- struct **relay_t**

Functions

- void **relay_free** (void)
 - *Free magma's relay server configuration options.* **bool_t relay_validate** (void)
 - *Validate all of the configured magma relay server instances.* void **relay_output_settings** (void)
 - **relay_t * relay_alloc** (uint32_t **number**)
 - *Allocate and initialize a magma server relay object, or return it if it already exists.* **bool_t relay_config** (**stringer_t** *name, **stringer_t** *value)
 - *Set the value of a relay server configuration entry by name.* **bool_t relay_set_value** (**relay_keys_t** *setting, **relay_t** *relay, **stringer_t** *value)
 - *Set a specified key for a relay server configuration entry.* void **relay_output_help** (void)
- Log all relay key information to be returned via "magma -h".*
-

Detailed Description

The types and functions involved with creating and accessing the relay structures.

Definition in file **relay.h**.

Function Documentation

relay_t* relay_alloc (uint32_t *number*)

Allocate and initialize a magma server relay object, or return it if it already exists.

Parameters:

number a zero-based index into the global relay server table where the entry should be created and/or queried.

Returns:

NULL on failure, or a pointer to the requested relay server object on success.

Definition at line 91 of file relay.c.

References magma_t::host, log_critical, magma, mm_alloc(), magma_t::relay, relay_keys, and relay_set_value().

Referenced by relay_config().

bool_t relay_config (**stringer_t** * *name*, **stringer_t** * *value*)

Set the value of a relay server configuration entry by name.

Note:

This function is designed to operate on lines from a configuration source, like a file or database. When a named server is referenced for the first time, a new relay server configuration instance will be allocated.

Parameters:

name a managed string containing the human-readable relay server configuration parameter to be set.
value a managed string containing the new value of the relay server config key.

Returns:

true if the value was successfully set, or false on failure.

Definition at line 453 of file relay.c.

References bracket_extract_pl(), CONSTANT, log_critical, MAGMA_RELAY_INSTANCES, NULLER, pl_char_get(), pl_empty(), pl_length_get(), PLACER, placer_t, relay_alloc(), relay_keys, relay_set_value(), st_char_get(), st_cmp_ci_eq(), st_cmp_ci_starts(), st_length_get(), st_length_int(), and uint32_conv_bl().

Referenced by config_load_cmdline_settings(), config_load_database_settings(), and config_load_file_settings().

void relay_free (void)

Free magma's relay server configuration options.

Note:

This function assumes all worker threads have finished, and should only be called during the normal shutdown process.

Parameters:

This function returns no value.

Definition at line 44 of file relay.c.

References magma_t::host, log_pedantic, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_ENUM, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_RELAY_INSTANCES, mm_free(), relay_keys_t::norm, ns_empty(), ns_free(), relay_keys_t::offset, magma_t::relay, relay_keys, st_empty(), st_free(), type(), and multi_t::type.

Referenced by config_free().

void relay_output_help (void)

Log all relay key information to be returned via "magma -h".

Returns:

This function returns no value.

Definition at line 513 of file relay.c.

References multi_t::bl, config_output_value_generic(), log_info, log_options, M_LOG_FILE_DISABLE, M_LOG_FUNCTION_DISABLE, M_LOG_LINE_DISABLE, M_LOG_LINE_FEED_DISABLE, M_LOG_STACK_TRACE_DISABLE, M_LOG_TIME_DISABLE, relay_keys_t::norm, relay_keys, relay_keys_t::required, and multi_t::val.

Referenced by config_output_help().

void relay_output_settings (void)

Definition at line 233 of file relay.c.

References magma_t::host, log_info, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_RELAY_INSTANCES, relay_keys_t::norm, ns_empty(), relay_keys_t::offset, magma_t::relay, relay_keys, st_char_get(), st_empty(), st_length_int(), multi_t::type, and type().

Referenced by config_output_settings().

bool_t relay_set_value (relay_keys_t * setting, relay_t * relay, stringer_t * value)

Set a specified key for a relay server configuration entry.

Note:

This function will allocate space for a copy of the key value, and return an error if it is not able to convert it to the proper key data type.

Parameters:

setting a pointer to the relay server configuration key to be set.

relay a pointer to the relay server configuration entry to be adjusted.

value a managed string containing the new value of the config key.

Returns:

true if the value was successfully set, or false on error.

Definition at line 308 of file relay.c.

References multi_t::binary, CONSTANT, CONTIGUOUS, HEAP, multi_t::i32, multi_t::i64, multi_t::i8, int32_conv_st(), int64_conv_st(), int8_conv_st(), log_critical, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, MANAGED_T, relay_keys_t::name, relay_keys_t::norm, multi_t::ns, ns_dupe(), ns_empty(), ns_free(), ns_import(), relay_keys_t::offset, relay_counter(), multi_t::st, st_char_get(), st_cmp_ci_eq(), st_dupe_opts(), st_empty(), st_free(), st_length_get(), type(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, uint16_conv_st(), uint32_conv_st(), uint64_conv_st(), uint8_conv_st(), and multi_t::val.

Referenced by relay_alloc(), and relay_config().

bool_t relay_validate (void)

Validate all of the configured magma relay server instances.

Iterates through all of the server structures and verifies that each was supplied with valid values for all of its required keys. This function also applies the application specific validation rules. Specifically it verifies that only one server structure has been configured to use a given port number.

Note:

This makes set that all required config keys have been set, and that at least one host has been configured.

Returns:

true if all relay servers have been validated, or false on failure.

Definition at line 120 of file relay.c.

References magma_t::host, multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, log_critical, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_RELAY_INSTANCES, relay_keys_t::norm, ns_empty(), relay_keys_t::offset, magma_t::relay, relay_keys, st_empty(), multi_t::type, type(), multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by config_validate_settings().

magma/engine/config/servers/servers.c File Reference

Functions for handling the server instance configurations.

```
#include "magma.h"
```

```
#include "keys.h"
```

Functions

- `uint64_t servers_get_count_using_port (uint32_t port)`
- *Get the number of servers that are configured to use a specified port.* `server_t * servers_get_by_socket (int sockd)`
- *Lookup the server instance associated with a socket descriptor.* `bool_t servers_network_start (void)`
- *Setup the listening sockets for all configured servers, or shut them all down if any one of them fails.* `bool_t servers_encryption_start (void)`
- *Create an SSL context for each SSL-configured server instance.* `void servers_network_stop (void)`
- *Close the listening sockets of all running servers.* `void servers_encryption_stop (void)`
- *Destroy the contexts of all SSL-enabled servers.* `void servers_free (void)`
- *Free magma's server configuration options.* `server_t * servers_alloc (uint32_t number)`
- *Allocate a new magma server configuration entry and initialize it to its default values.* `bool_t servers_validate (void)`
- *Validate all of the configured magma server instances.* `void servers_output_settings (void)`
- *Log the full contents of the magma server instance settings.* `bool_t servers_set_value (server_keys_t *setting, server_t *server, stringer_t *value)`
- *Set the value of a key for a specified magma server configuration entry.* `bool_t servers_config (stringer_t *name, stringer_t *value)`
- *Set the value of a magma server configuration entry by name.* `void servers_output_help (void)`

Log all server key information to be returned via "magma -h".

Detailed Description

Functions for handling the server instance configurations.

Definition in file `servers.c`.

Function Documentation

`server_t* servers_alloc (uint32_t number)`

Allocate a new magma server configuration entry and initialize it to its default values.

servers.c

Parameters:

number the index of the magma server instance in the global magma server array.

Returns:

NULL on failure, or a pointer to the requested magma server configuration entry on success.

Definition at line 162 of file `servers.c`.

References `log_critical`, `magma`, `mm_alloc()`, `server_t::network`, `server_keys`, `magma_t::servers`, `servers_set_value()`, and `server_t::sockd`.

Referenced by servers_config().

bool_t servers_config (stringer_t * name, stringer_t * value)

Set the value of a magma server configuration entry by name.

Note:

This function is designed to operate on lines from a configuration source, like a file or database. When a named server is referenced for the first time, a new magma server configuration instance will be allocated.

Parameters:

name a managed string containing the human-readable magma server configuration parameter to be set.

value a managed string containing the new value of the magma server config key.

Returns:

true if the value was successfully set, or false on failure.

Definition at line 605 of file servers.c.

References bracket_extract_pl(), CONSTANT, log_critical, MAGMA_SERVER_INSTANCES, NULLER, pl_char_get(), pl_empty(), pl_length_get(), PLACER, placer_t, server_keys, servers_alloc(), servers_set_value(), st_char_get(), st_cmp_ci_eq(), st_cmp_ci_starts(), st_length_get(), st_length_int(), and uint32_conv_bl().

Referenced by config_load_cmdline_settings(), config_load_database_settings(), and config_load_file_settings().

bool_t servers_encryption_start (void)

Create an SSL context for each SSL-configured server instance.

Returns:

true on success or false on failure.

Definition at line 74 of file servers.c.

References server_t::certificate, magma, MAGMA_SERVER_INSTANCES, magma_t::servers, server_t::ssl, and ssl_server_create().

Referenced by process_start().

void servers_encryption_stop (void)

Destroy the contexts of all SSL-enabled servers.

Returns:

This function returns no value.

Definition at line 101 of file servers.c.

References server_t::context, magma, MAGMA_SERVER_INSTANCES, magma_t::servers, server_t::ssl, and ssl_server_destroy().

Referenced by process_stop().

void servers_free (void)

Free magma's server configuration options.

Note:

This function assumes all worker threads have finished, and should only be called during the normal shutdown process.

Parameters:

This function returns no value.

Definition at line 115 of file servers.c.

References log_pedantic, M_TYPE_BLOCK, M_TYPE_BOOLEAN, M_TYPE_DOUBLE, M_TYPE_ENUM, M_TYPE_FLOAT, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_SERVER_INSTANCES, mm_free(), server_keys_t::norm, ns_empty(), ns_free(), server_keys_t::offset, server_keys, magma_t::servers, st_empty(), st_free(), type(), and multi_t::type.

Referenced by config_free().

server_t* servers_get_by_socket (int sockd) [inline]

Lookup the server instance associated with a socket descriptor.

Parameters:

sockd the socket file descriptor to be looked up.

Returns:

NULL on failure, or a pointer to the server structure for the specified file descriptor on success.

Definition at line 39 of file servers.c.

References magma, MAGMA_SERVER_INSTANCES, server_t::network, magma_t::servers, and server_t::sockd.

Referenced by net_listen().

uint64_t servers_get_count_using_port (uint32_t port)

Get the number of servers that are configured to use a specified port.

Parameters:

port the port number to be matched against the server list.

Returns:

the number of servers configured to use the port.

Definition at line 22 of file servers.c.

References count, magma, MAGMA_SERVER_INSTANCES, server_t::network, server_t::port, and magma_t::servers.

Referenced by servers_validate(), and signal_shutdown().

bool_t servers_network_start (void)

Setup the listening sockets for all configured servers, or shut them all down if any one of them fails.

Note:

If any of the servers have an empty name or domain, the value of magma.host.name will be used instead.

Returns:

true if all server listening sockets were initialized successfully, or false on failure.

Definition at line 55 of file servers.c.

References server_t::domain, magma_t::host, magma, MAGMA_SERVER_INSTANCES, magma_t::name, server_t::name, net_init(), ns_length_get(), magma_t::servers, servers_network_stop(), st_empty(), and st_import().

Referenced by process_start().

void servers_network_stop (void)

Close the listening sockets of all running servers.

Returns:

This function returns no value.

Definition at line 88 of file servers.c.

References magma, MAGMA_SERVER_INSTANCES, net_shutdown(), server_t::network, magma_t::servers, and server_t::sockd.

Referenced by process_stop(), and servers_network_start().

void servers_output_help (void)

Log all server key information to be returned via "magma -h".

Returns:

This function returns no value.

Definition at line 662 of file servers.c.

References multi_t::bl, config_output_value_generic(), log_info, log_options, M_LOG_FILE_DISABLE, M_LOG_FUNCTION_DISABLE, M_LOG_LINE_DISABLE, M_LOG_LINE_FEED_DISABLE, M_LOG_STACK_TRACE_DISABLE, M_LOG_TIME_DISABLE, server_keys_t::norm, server_keys_t::required, server_keys, and multi_t::val.

Referenced by config_output_help().

void servers_output_settings (void)

Log the full contents of the magma server instance settings.

Returns:

This function returns no value.

Definition at line 324 of file servers.c.

References `CONSTANT`, `EMPTY`, `HTTP`, `IMAP`, `log_info`, `log_pedantic`, `M_TYPE_BOOLEAN`, `M_TYPE_ENUM`, `M_TYPE_INT16`, `M_TYPE_INT32`, `M_TYPE_INT64`, `M_TYPE_INT8`, `M_TYPE_NULLER`, `M_TYPE_STRINGER`, `M_TYPE_UINT16`, `M_TYPE_UINT32`, `M_TYPE_UINT64`, `M_TYPE_UINT8`, `magma`, `MAGMA_SERVER_INSTANCES`, `MOLTEN`, `server_keys_t::norm`, `ns_empty()`, `NULLER`, `server_keys_t::offset`, `POP`, `server_keys`, `magma_t::servers`, `SMTP`, `SSL_PORT`, `st_char_get()`, `st_cmp_cs_eq()`, `st_empty()`, `st_length_int()`, `SUBMISSION`, `TCP_PORT`, `multi_t::type`, and `type()`.

Referenced by `config_output_settings()`.

bool_t servers_set_value (server_keys_t * setting, server_t * server, stringer_t * value)

Set the value of a key for a specified magma server configuration entry.

Note:

This function will allocate space for a copy of the key value, and return an error if it is not able to convert it to the proper key data type.

Parameters:

setting a pointer to the magma server key to be set.

server a pointer to the magma server configuration entry to be modified.

value a managed string containing the new value of the key.

Returns:

true if the value was successfully set, or false on failure.

Definition at line 427 of file servers.c.

References `multi_t::binary`, `CONSTANT`, `CONTIGUOUS`, `HEAP`, `HTTP`, `multi_t::i32`, `multi_t::i64`, `multi_t::i8`, `IMAP`, `int32_conv_st()`, `int64_conv_st()`, `int8_conv_st()`, `log_critical`, `M_TYPE_BOOLEAN`, `M_TYPE_ENUM`, `M_TYPE_INT16`, `M_TYPE_INT32`, `M_TYPE_INT64`, `M_TYPE_INT8`, `M_TYPE_NULLER`, `M_TYPE_STRINGER`, `M_TYPE_UINT16`, `M_TYPE_UINT32`, `M_TYPE_UINT64`, `M_TYPE_UINT8`, `MANAGED_T`, `MOLTEN`, `server_keys_t::name`, `server_keys_t::norm`, `multi_t::ns`, `ns_dupe()`, `ns_empty()`, `ns_free()`, `ns_import()`, `NULLER`, `server_keys_t::offset`, `POP`, `SMTP`, `SSL_PORT`, `multi_t::st`, `st_char_get()`, `st_cmp_ci_eq()`, `st_dupe_opts()`, `st_empty()`, `st_free()`, `st_length_get()`, `st_length_int()`, `SUBMISSION`, `TCP_PORT`, `type()`, `multi_t::type`, `multi_t::u16`, `multi_t::u32`, `multi_t::u64`, `multi_t::u8`, `uint16_conv_st()`, `uint32_conv_st()`, `uint64_conv_st()`, `uint8_conv_st()`, and `multi_t::val`.

Referenced by `servers_alloc()`, and `servers_config()`.

bool_t servers_validate (void)

Validate all of the configured magma server instances.

Note:

This function also checks to see that no servers are binding to the same port.

Returns:

true if all servers have been validated, or false on failure.

Definition at line 189 of file servers.c.

References `buflen`, `bufptr`, `server_t::certificate`, `CONSTANT`, `EMPTY`, `file_readwritable()`, `file_world_accessible()`, `multi_t::i16`, `multi_t::i32`, `multi_t::i64`, `multi_t::i8`, `log_critical`, `log_pedantic`,

M_TYPE_BOOLEAN, M_TYPE_ENUM, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_SERVER_INSTANCES, server_t::network, server_keys_t::norm, ns_empty(), NULLER, server_keys_t::offset, server_t::port, server_keys, magma_t::servers, servers_get_count_using_port(), server_t::ssl, SSL_PORT, st_cmp_ci_eq(), st_empty(), multi_t::type, type(), multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by config_validate_settings().

magma/engine/config/servers/servers.h File Reference

The types and functions involved in creating and accessing the server structure.

Data Structures

- struct **server_keys_t**
- struct **server_t**
- struct **server_config_t**

Enumerations

- enum **M_PORT** { **TCP_PORT** = 1, **SSL_PORT** }
- enum **M_PROTOCOL** { **MOLTEN** = 1, **HTTP**, **POP**, **IMAP**, **SMTP**, **SUBMISSION** }

Functions

- **server_t** * **servers_alloc** (uint32_t number)
- *servers.c* **bool_t** **servers_config** (**stringer_t** *name, **stringer_t** *value)
- *Set the value of a magma server configuration entry by name.* **bool_t** **servers_encryption_start** (void)
- *Create an SSL context for each SSL-configured server instance.* void **servers_encryption_stop** (void)
- *Destroy the contexts of all SSL-enabled servers.* void **servers_free** (void)
- *Free magma's server configuration options.* **server_t** * **servers_get_by_socket** (int sockd)
- *Lookup the server instance associated with a socket descriptor.* uint64_t **servers_get_count_using_port** (uint32_t port)
- *Get the number of servers that are configured to use a specified port.* **bool_t** **servers_network_start** (void)
- *Setup the listening sockets for all configured servers, or shut them all down if any one of them fails.* void **servers_network_stop** (void)
- *Close the listening sockets of all running servers.* void **servers_output_settings** (void)
- *Log the full contents of the magma server instance settings.* **bool_t** **servers_set_value** (**server_keys_t** *setting, **server_t** *server, **stringer_t** *value)
- *Set the value of a key for a specified magma server configuration entry.* **bool_t** **servers_validate** (void)
- *Validate all of the configured magma server instances.* void **servers_output_help** (void)

Log all server key information to be returned via "magma -h".

Detailed Description

The types and functions involved in creating and accessing the server structure.

Definition in file **servers.h**.

Enumeration Type Documentation

enum **M_PORT**

Enumerator:

TCP_PORT
SSL_PORT

Definition at line 16 of file **servers.h**.

enum M_PROTOCOL

Enumerator:

MOLTEN
HTTP
POP
IMAP
SMTP
SUBMISSION

Definition at line 21 of file servers.h.

Function Documentation

server_t* servers_alloc (uint32_t *number*)

servers.c

servers.c

Parameters:

number the index of the magma server instance in the global magma server array.

Returns:

NULL on failure, or a pointer to the requested magma server configuration entry on success.

Definition at line 162 of file servers.c.

References log_critical, magma, mm_alloc(), server_t::network, server_keys, magma_t::servers, servers_set_value(), and server_t::sockd.

Referenced by servers_config().

bool_t servers_config (stringer_t * *name*, stringer_t * *value*)

Set the value of a magma server configuration entry by name.

Note:

This function is designed to operate on lines from a configuration source, like a file or database. When a named server is referenced for the first time, a new magma server configuration instance will be allocated.

Parameters:

name a managed string containing the human-readable magma server configuration parameter to be set.

value a managed string containing the new value of the magma server config key.

Returns:

true if the value was successfully set, or false on failure.

Definition at line 605 of file servers.c.

References bracket_extract_pl(), CONSTANT, log_critical, MAGMA_SERVER_INSTANCES, NULLER, pl_char_get(), pl_empty(), pl_length_get(), PLACER, placer_t, server_keys, servers_alloc(), servers_set_value(), st_char_get(), st_cmp_ci_eq(), st_cmp_ci_starts(), st_length_get(), st_length_int(), and uint32_conv_bl().

Referenced by `config_load_cmdline_settings()`, `config_load_database_settings()`, and `config_load_file_settings()`.

bool_t servers_encryption_start (void)

Create an SSL context for each SSL-configured server instance.

Returns:

true on success or false on failure.

Definition at line 74 of file `servers.c`.

References `server_t::certificate`, `magma`, `MAGMA_SERVER_INSTANCES`, `magma_t::servers`, `server_t::ssl`, and `ssl_server_create()`.

Referenced by `process_start()`.

void servers_encryption_stop (void)

Destroy the contexts of all SSL-enabled servers.

Returns:

This function returns no value.

Definition at line 101 of file `servers.c`.

References `server_t::context`, `magma`, `MAGMA_SERVER_INSTANCES`, `magma_t::servers`, `server_t::ssl`, and `ssl_server_destroy()`.

Referenced by `process_stop()`.

void servers_free (void)

Free magma's server configuration options.

Note:

This function assumes all worker threads have finished, and should only be called during the normal shutdown process.

Parameters:

This function returns no value.

Definition at line 115 of file `servers.c`.

References `log_pedantic`, `M_TYPE_BLOCK`, `M_TYPE_BOOLEAN`, `M_TYPE_DOUBLE`, `M_TYPE_ENUM`, `M_TYPE_FLOAT`, `M_TYPE_INT16`, `M_TYPE_INT32`, `M_TYPE_INT64`, `M_TYPE_INT8`, `M_TYPE_NULLER`, `M_TYPE_STRINGER`, `M_TYPE_UINT16`, `M_TYPE_UINT32`, `M_TYPE_UINT64`, `M_TYPE_UINT8`, `magma`, `MAGMA_SERVER_INSTANCES`, `mm_free()`, `server_keys_t::norm`, `ns_empty()`, `ns_free()`, `server_keys_t::offset`, `server_keys`, `magma_t::servers`, `st_empty()`, `st_free()`, `type()`, and `multi_t::type`.

Referenced by `config_free()`.

server_t* servers_get_by_socket (int sockd) [inline]

Lookup the server instance associated with a socket descriptor.

Parameters:

sockd the socket file descriptor to be looked up.

Returns:

NULL on failure, or a pointer to the server structure for the specified file descriptor on success.

Definition at line 39 of file servers.c.

References magma, MAGMA_SERVER_INSTANCES, server_t::network, magma_t::servers, and server_t::sockd.

Referenced by net_listen().

uint64_t servers_get_count_using_port (uint32_t port)

Get the number of servers that are configured to use a specified port.

Parameters:

port the port number to be matched against the server list.

Returns:

the number of servers configured to use the port.

Definition at line 22 of file servers.c.

References count, magma, MAGMA_SERVER_INSTANCES, server_t::network, server_t::port, and magma_t::servers.

Referenced by servers_validate(), and signal_shutdown().

bool_t servers_network_start (void)

Setup the listening sockets for all configured servers, or shut them all down if any one of them fails.

Note:

If any of the servers have an empty name or domain, the value of magma.host.name will be used instead.

Returns:

true if all server listening sockets were initialized successfully, or false on failure.

Definition at line 55 of file servers.c.

References server_t::domain, magma_t::host, magma, MAGMA_SERVER_INSTANCES, magma_t::name, server_t::name, net_init(), ns_length_get(), magma_t::servers, servers_network_stop(), st_empty(), and st_import().

Referenced by process_start().

void servers_network_stop (void)

Close the listening sockets of all running servers.

Returns:

This function returns no value.

Definition at line 88 of file servers.c.

References magma, MAGMA_SERVER_INSTANCES, net_shutdown(), server_t::network, magma_t::servers, and server_t::sockd.

Referenced by process_stop(), and servers_network_start().

void servers_output_help (void)

Log all server key information to be returned via "magma -h".

Returns:

This function returns no value.

Definition at line 662 of file servers.c.

References multi_t::bl, config_output_value_generic(), log_info, log_options, M_LOG_FILE_DISABLE, M_LOG_FUNCTION_DISABLE, M_LOG_LINE_DISABLE, M_LOG_LINE_FEED_DISABLE, M_LOG_STACK_TRACE_DISABLE, M_LOG_TIME_DISABLE, server_keys_t::norm, server_keys_t::required, server_keys, and multi_t::val.

Referenced by config_output_help().

void servers_output_settings (void)

Log the full contents of the magma server instance settings.

Returns:

This function returns no value.

Definition at line 324 of file servers.c.

References CONSTANT, EMPTY, HTTP, IMAP, log_info, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_ENUM, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_SERVER_INSTANCES, MOLTEN, server_keys_t::norm, ns_empty(), NULLER, server_keys_t::offset, POP, server_keys, magma_t::servers, SMTP, SSL_PORT, st_char_get(), st_cmp_cs_eq(), st_empty(), st_length_int(), SUBMISSION, TCP_PORT, multi_t::type, and type().

Referenced by config_output_settings().

bool_t servers_set_value (server_keys_t * *setting*, server_t * *server*, stringer_t * *value*)

Set the value of a key for a specified magma server configuration entry.

Note:

This function will allocate space for a copy of the key value, and return an error if it is not able to convert it to the proper key data type.

Parameters:

setting a pointer to the magma server key to be set.

server a pointer to the magma server configuration entry to be modified.

value a managed string containing the new value of the key.

Returns:

true if the value was successfully set, or false on failure.

Definition at line 427 of file servers.c.

References multi_t::binary, CONSTANT, CONTIGUOUS, HEAP, HTTP, multi_t::i32, multi_t::i64, multi_t::i8, IMAP, int32_conv_st(), int64_conv_st(), int8_conv_st(), log_critical, M_TYPE_BOOLEAN, M_TYPE_ENUM, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, MANAGED_T, MOLTEN, server_keys_t::name, server_keys_t::norm, multi_t::ns, ns_dupe(), ns_empty(), ns_free(), ns_import(), NULLER, server_keys_t::offset, POP, SMTP, SSL_PORT, multi_t::st, st_char_get(), st_cmp_ci_eq(), st_dupe_opts(), st_empty(), st_free(), st_length_get(), st_length_int(), SUBMISSION, TCP_PORT, type(), multi_t::type, multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, uint16_conv_st(), uint32_conv_st(), uint64_conv_st(), uint8_conv_st(), and multi_t::val.

Referenced by servers_alloc(), and servers_config().

bool_t servers_validate (void)

Validate all of the configured magma server instances.

Note:

This function also checks to see that no servers are binding to the same port.

Returns:

true if all servers have been validated, or false on failure.

Definition at line 189 of file servers.c.

References buflen, bufptr, server_t::certificate, CONSTANT, EMPTY, file_readwritable(), file_world_accessible(), multi_t::i16, multi_t::i32, multi_t::i64, multi_t::i8, log_critical, log_pedantic, M_TYPE_BOOLEAN, M_TYPE_ENUM, M_TYPE_INT16, M_TYPE_INT32, M_TYPE_INT64, M_TYPE_INT8, M_TYPE_NULLER, M_TYPE_STRINGER, M_TYPE_UINT16, M_TYPE_UINT32, M_TYPE_UINT64, M_TYPE_UINT8, magma, MAGMA_SERVER_INSTANCES, server_t::network, server_keys_t::norm, ns_empty(), NULLER, server_keys_t::offset, server_t::port, server_keys, magma_t::servers, servers_get_count_using_port(), server_t::ssl, SSL_PORT, st_cmp_ci_eq(), st_empty(), multi_t::type, type(), multi_t::u16, multi_t::u32, multi_t::u64, multi_t::u8, and multi_t::val.

Referenced by config_validate_settings().

magma/servers/servers.h File Reference

The inclusion point for the different server implementations.

```
#include "http/http.h"
#include "imap/imap.h"
#include "molten/molten.h"
#include "pop/pop.h"
#include "smtp/smtp.h"
```

Defines

- `#define M_SSL_BIO_NOCLOSE 0x00`
-

Detailed Description

The inclusion point for the different server implementations.

Definition in file **servers.h**.

Define Documentation

#define M_SSL_BIO_NOCLOSE 0x00

Definition at line 16 of file servers.h.

Referenced by `imap_starttls()`, `pop_starttls()`, and `smtp_starttls()`.

magma/engine/context/args.c File Reference

Functions for parsing command line arguments.

```
#include "magma.h"
```

Functions

- void **display_usage** (void)
- *Display the magma usage info if a user requests help or invokes it with incorrect options.* **bool_t args_parse** (int argc, char *argv[])

Process any command line arguments supplied to magma. Variables

- **stringer_t * cmdline_config_data**
- **bool_t exit_and_dump**

Detailed Description

Functions for parsing command line arguments.

Definition in file **args.c**.

Function Documentation

bool_t args_parse (int *argc*, char * *argv*[])

Process any command line arguments supplied to magma.

Note:

A few command line options are supported: -h (--help), -d (--dump), -v (--version), and -c (--config). Otherwise, the magma config file path will be loaded from the command line. If not an option starting with "-", the final command line argument is assumed to be the path of the magma configuration file.

Returns:

true if the command line arguments were parsed successfully, or false if there was an error.

Definition at line 44 of file args.c.

References `build_stamp()`, `build_version()`, `cmdline_config_data`, `magma_t::config`, `config_load_defaults()`, `config_output_help()`, `display_usage()`, `exit_and_dump`, `magma_t::file`, `HEAP`, `JOINTED`, `log_critical`, `log_info`, `magma`, `MAGMA_FILEPATH_MAX`, `MANAGED_T`, `mm_cmp_cs_eq()`, `NULLER`, `PLACER`, `st_append`, `st_cmp_cs_eq()`, and `st_dupe_opts()`.

Referenced by `main()`.

void display_usage (void)

Display the magma usage info if a user requests help or invokes it with incorrect options.

args.c

Returns:

This function returns no value.

Definition at line 24 of file args.c.

References log_info.

Referenced by args_parse().

Variable Documentation**stringer_t* cmdline_config_data**

Definition at line 19 of file global.c.

Referenced by args_parse(), and config_load_cmdline_settings().

bool_t exit_and_dump

Definition at line 18 of file global.c.

magma/engine/context/context.h File Reference

Functions involved in initializing, manipulating and verifying the operating context for the system.

Functions

- void **display_usage** (void)
- *args.c* **bool_t args_parse** (int argc, char *argv[])
- *Process any command line arguments supplied to magma.* **bool_t sanity_check** (void)
- *Perform a series of system-wide sanity checks at process startup.* void **process_stop** (void)
- *Execute the magma shutdown routines.* **bool_t process_start** (void)
- *Execute the magma initializations routines, making sure they all run properly.* **bool_t signal_start** (void)
- *signal.c* **bool_t signal_thread_start** (void)
- *Bind a SIGALRM handler for the calling thread.* void **signal_segfault** (int signal)
- *Handle signals that indicate the program was killed because of an invalid operation (including SIGSEGV).* void **signal_shutdown** (int signal)
- *A function to handle receipt of shutdown signals and allow for graceful exits.* void **signal_status** (int signal)
- *A generic worker thread signal handler entry point.* **bool_t system_change_root_directory** (void)
- *system.c* **bool_t system_fork_daemon** (void)
- *Daemonize into the background, if the magma.system.daemonize config option is set.* **bool_t system_init_core_dumps** (void)
- *Set the magma core dump size rlimit, if magma.system.enable_core_dumps was enabled.* **bool_t system_init_impersonation** (void)
- *Set process privileges to run as a specified user if magma.system.impersonate_user is set.* **bool_t system_init_resource_limits** (void)
- *Increase process resource limits, if magma.system.increase_resource_limits is set.* **bool_t system_init_umask** (void)
- *Set the process umask for new file/dir creation to O_RDWR.* uint64_t **system_limit_cur** (int_t resource)
- *Get the soft system limit for a specified resource.* uint64_t **system_limit_max** (int_t resource)
- *Get the hard system limit for a specified resource.* **bool_t thread_start** (void)
- *thread.c* void **thread_stop** (void)

Prepare a thread to exit by destroying its mysql and openssl-specific and associated mail cache data.

Detailed Description

Functions involved in initializing, manipulating and verifying the operating context for the system.

Definition in file **context.h**.

Function Documentation

bool_t args_parse (int argc, char * argv[])

Process any command line arguments supplied to magma.

Note:

A few command line options are supported: -h (--help), -d (--dump), -v (--version), and -c (--config). Otherwise, the magma config file path will be loaded from the command line. If not an option starting with "-", the final command line argument is assumed to be the path of the magma configuration file.

Returns:

true if the command line arguments were parsed successfully, or false if there was an error.

Definition at line 44 of file args.c.

References build_stamp(), build_version(), cmdline_config_data, magma_t::config, config_load_defaults(), config_output_help(), display_usage(), exit_and_dump, magma_t::file, HEAP, JOINTED, log_critical, log_info, magma, MAGMA_FILEPATH_MAX, MANAGED_T, mm_cmp_cs_eq(), NULLER, PLACER, st_append, st_cmp_cs_eq(), and st_dupe_opts().

Referenced by main().

void display_usage (void)

args.c

args.c

Returns:

This function returns no value.

Definition at line 24 of file args.c.

References log_info.

Referenced by args_parse().

bool_t process_start (void)

Execute the magma initializations routines, making sure they all run properly.

Note:

If any of the init routines returns with failure, this function fails and logs an error message. Certain checks are made that the init routines are run in a particular order, especially in waiting for daemonization and privilege dropping before starting logging. The system maintenance thread is also launched from here.

Returns:

true if all starter functions returned true, or false if any of them failed..

Definition at line 166 of file process.c.

References cache_start(), config_load_cmdline_settings(), config_load_database_settings(), config_load_defaults(), config_load_file_settings(), config_validate_settings(), dkim_start(), dspam_start(), ecies_start(), exit_and_dump, magma_t::host, http_content_start(), magma_t::init, lib_load(), log_critical, log_info, log_start(), magma, MAGMA_HOSTNAME_MAX, mail_cache_start(), maint, mm_alloc(), mm_free(), mm_sec_start(), magma_t::name, obj_cache_start(), magma_t::page_length, process_maint(), protocol_init(), queue_init(), rand_start(), sanity_check(), servers_encryption_start(), servers_network_start(), signal_start(), spf_start(), spool_start(), sql_start(), ssl_start(), stats_init(), status_process(), system_change_root_directory(), system_fork_daemon(), system_init_core_dumps(), system_init_impersonation(), system_init_resource_limits(), system_init_umask(), thread_launch(), virus_start(), warehouse_start(), and xml_start().

Referenced by main().

void process_stop (void)

Execute the magma shutdown routines.

Note:

This function will execute a shutdown routine for each of the startup initialization routines that was performed. It will also signal and terminate the maintenance thread.

Returns:

This function returns no value.

Definition at line 71 of file process.c.

References `cache_stop()`, `config_free()`, `dkim_stop()`, `dspam_stop()`, `ecies_stop()`, `http_content_stop()`, `magma_t::init`, `lib_unload()`, `log_critical`, `log_info`, `log_pedantic`, `magma`, `mail_cache_stop()`, `maint`, `mm_free()`, `mm_sec_stop()`, `obj_cache_stop()`, `queue_shutdown()`, `rand_stop()`, `servers_encryption_stop()`, `servers_network_stop()`, `spf_stop()`, `spool_stop()`, `sql_stop()`, `ssl_stop()`, `stats_shutdown()`, `status`, `thread_join()`, `thread_signal()`, `virus_stop()`, `warehouse_stop()`, and `xml_stop()`.

Referenced by `main()`.

bool_t sanity_check (void)

Perform a series of system-wide sanity checks at process startup.

Note:

This function ensures that the standard data types are their expected data sizes, and performs other checks, such as data primitive maximum values.

Returns:

true if the system environment passed all checks and should function correctly, or false if any checks failed.

Definition at line 20 of file sanity.c.

References `CONSTANT`, `MAGMA_CORE_POOL_OBJECTS_LIMIT`, `MAGMA_CORE_POOL_TIMEOUT_LIMIT`, `NULLER`, `placer_t`, `queries`, `SELECT_DOMAINS`, `st_alloc`, `st_avail_get()`, `st_cmp_cs_eq()`, `st_data_get()`, and `st_free()`.

Referenced by `process_start()`.

void signal_segfault (int *signal*)

Handle signals that indicate the program was killed because of an invalid operation (including SIGSEGV).

Note:

The signals handled by this function are: SIGSEGV, SIGFPE, SIGBUS, SIGSYS, and SIGABRT. The handler will respond by logging a stack backtrace, and terminating the program with `abort()`.

Parameters:

signal the number of the signal that was received.

Returns:

This function returns no value.

Definition at line 24 of file signal.c.

References `log_options`, `M_LOG_CRITICAL`, `M_LOG_STACK_TRACE`, `mm_wipe()`, and `signal_name()`.

Referenced by `signal_start()`.

void signal_shutdown (int *signal*)

A function to handle receipt of shutdown signals and allow for graceful exits.

Note:

This function handles the following signals: SIGINT, SIGQUIT, and SIGTERM. The shutdown procedure is as follows: 1. Set status to -1 and sleep for .1 second to allow for normal daemon termination. 2. Signal all worker threads to wake up blocked threads, and sleep one more second. 3. Loop through all possible file descriptors and if the descriptor matches a socket that is bound to a magma server, close it.

Parameters:

signal the number of the signal delivered to the process.

Returns:

This function returns no value.

Definition at line 57 of file `signal.c`.

References `ip_t::family`, `ip_t::ip4`, `ip_t::ip6`, `ip_presentation()`, `log_critical`, `log_info`, `mm_copy()`, `mm_wipe()`, `PLACER`, `queue_signal()`, `servers_get_count_using_port()`, `signal_name()`, `st_char_get()`, and `status_set()`.

Referenced by `signal_start()`.

bool_t signal_start (void)

signal.c

signal.c

Note:

The following handlers are established: `signal_shutdown`: SIGINT, SIGQUIT, SIGTERM, SIGHUP
`signal_segfault`: SIGSEGV, SIGFPE, SIGBUS, SIGSYS `signal_refresh`: SIGHUP [ignored]: SIGPIPE

Returns:

true if all signal handlers were successfully registered, or false on failure.

Definition at line 182 of file `signal.c`.

References `log_info`, `mm_wipe()`, `signal_refresh()`, `signal_segfault()`, and `signal_shutdown()`.

Referenced by `process_start()`.

void signal_status (int *signal*)

A generic worker thread signal handler entry point.

Note:

If the target thread is not in the process of shutting down, display the name of the caught signal.

Parameters:

signal the number of the signal caught by the signal handler.

Returns:

This function returns no value.

Definition at line 122 of file `signal.c`.

References `log_info`, `signal_name()`, and `status`.

Referenced by `signal_thread_start()`.

`bool_t signal_thread_start (void)`

Bind a SIGALRM handler for the calling thread.

See also:

`signal_status()`

Returns:

false on failure or true on success.

Definition at line 157 of file `signal.c`.

References `log_info`, `mm_wipe()`, and `signal_status()`.

Referenced by `thread_start()`.

`bool_t system_change_root_directory (void)`

`system.c`

`system.c`

Returns:

true on success or false on failure.

Definition at line 72 of file `system.c`.

References `buflen`, `bufptr`, `log_info`, `magma`, `magma_t::root_directory`, and `magma_t::system`.

Referenced by `process_start()`.

`bool_t system_fork_daemon (void)`

Daemonize into the background, if the `magma.system.daemonize` config option is set.

Returns:

true inside the child process, or false inside the parent process or if an error occurs.

Definition at line 87 of file `system.c`.

References `magma_t::daemonize`, `log_info`, `magma`, `pid`, `status_process()`, and `magma_t::system`.

Referenced by `process_start()`.

`bool_t system_init_core_dumps (void)`

Set the magma core dump size `rlimit`, if `magma.system.enable_core_dumps` was enabled.

Returns:

true if core dumps were successfully enabled or false on failure.

Definition at line 144 of file `system.c`.

References `buflen`, `bufptr`, `magma_t::core_dump_size_limit`, `magma_t::enable_core_dumps`, `log_info`, `magma`, and `magma_t::system`.

Referenced by `process_start()`.

`bool_t system_init_impersonation (void)`

Set process privileges to run as a specified user if `magma.system.impersonate_user` is set.

Note:

This function will set the user id and group id to the specified user, and `chdir()` to their home directory.

Returns:

true on success or if `magma.system.impersonate_user` is not set; false on failure to change privileges.

Definition at line 164 of file `system.c`.

References `buflen`, `bufptr`, `magma_t::impersonate_user`, `log_info`, `magma`, `mm_alloc()`, `mm_free()`, `mm_wipe()`, `status_process()`, and `magma_t::system`.

Referenced by `process_start()`.

`bool_t system_init_resource_limits (void)`

Increase process resource limits, if `magma.system.increase_resource_limits` is set.

Note:

Resource limits will be maximized for magma's virtual address space, data and stack segments, available file descriptors and sub-processes, and allowed file sizes. If `output_resource_limits` is enabled, the state of the process resource limits will be dumped to the log afterwards.

Returns:

This function always returns true (errors are logged).

Definition at line 233 of file `system.c`.

References `buflen`, `bufptr`, `magma_t::config`, `magma_t::increase_resource_limits`, `log_info`, `magma`, `magma_t::output_resource_limits`, and `magma_t::system`.

Referenced by `process_start()`.

`bool_t system_init_umask (void)`

Set the process umask for new file/dir creation to `O_RDWR`.

Returns:

true if the umask was set successfully, or false on failure.

Definition at line 126 of file `system.c`.

References `buflen`, `bufptr`, and `log_info`.

Referenced by `process_start()`.

`uint64_t system_limit_cur (int_t resource)`

Get the soft system limit for a specified resource.

Note:

This function will never return a value greater than `UINT64_MAX`.

See also:

`getrlimit64()`

Parameters:

resource the system rlimit resource identifier to be queried.

Returns:

-1 on failure, or the system soft limit of the specified resource identifier on success.

Definition at line 48 of file `system.c`.

References `buflen`, `bufptr`, `log_info`, and `log_pedantic`.

Referenced by `st_realloc()`.

uint64_t system_limit_max (int_t resource)

Get the hard system limit for a specified resource.

Note:

This function will never return a value greater than `UINT64_MAX`.

See also:

`getrlimit64()`

Parameters:

resource the system rlimit resource identifier to be queried.

Returns:

-1 on failure, or the system hard limit of the specified resource identifier on success.

Definition at line 22 of file `system.c`.

References `buflen`, `bufptr`, `log_info`, and `log_pedantic`.

Referenced by `config_validate_settings()`.

bool_t thread_start (void)

`thread.c`

`thread.c`

Returns:

true on success or false on failure.

Definition at line 32 of file `thread.c`.

References `signal_thread_start()`, and `sql_thread_start()`.

Referenced by `dequeue()`, and `process_maint()`.

void thread_stop (void)

Prepare a thread to exit by destroying its mysql and openssl-specific and associated mail cache data.

Returns:

This function returns no value.

Definition at line 19 of file thread.c.

References `mail_cache_thread_stop()`, `sql_thread_stop()`, and `ssl_thread_stop()`.

Referenced by `dequeue()`, and `process_maint()`.

magma/engine/context/sanity.c File Reference

A collection of checks performed at launch to make sure the system will operate as expected.

```
#include "magma.h"
```

Functions

- **bool_t sanity_check** (void)

Perform a series of system-wide sanity checks at process startup.

Detailed Description

A collection of checks performed at launch to make sure the system will operate as expected.

Definition in file **sanity.c**.

Function Documentation

bool_t sanity_check (void)

Perform a series of system-wide sanity checks at process startup.

Note:

This function ensures that the standard data types are their expected data sizes, and performs other checks, such as data primitive maximum values.

Returns:

true if the system environment passed all checks and should function correctly, or false if any checks failed.

Definition at line 20 of file sanity.c.

References `CONSTANT`, `MAGMA_CORE_POOL_OBJECTS_LIMIT`, `MAGMA_CORE_POOL_TIMEOUT_LIMIT`, `NULLER`, `placer_t`, `queries`, `SELECT_DOMAINS`, `st_alloc`, `st_avail_get()`, `st_cmp_cs_eq()`, `st_data_get()`, and `st_free()`.

Referenced by `process_start()`.

magma/engine/context/signal.c File Reference

A collection of functions used to register and handle signals.

```
#include "magma.h"
```

Functions

- void **signal_segfault** (int signal)
- *Handle signals that indicate the program was killed because of an invalid operation (including SIGSEGV).* void **signal_shutdown** (int signal)
- *A function to handle receipt of shutdown signals and allow for graceful exits.* void **signal_status** (int signal)
- *A generic worker thread signal handler entry point.* void **signal_refresh** (int signal)
- *A function to handle receipt of SIGHUP and refresh all web server content.* **bool_t** **signal_thread_start** (void)
- *Bind a SIGALRM handler for the calling thread.* **bool_t** **signal_start** (void)

Setup signal masks and register signal handlers for handling shutdowns, program termination, and reloads. Variables

- pthread_mutex_t **sig_hup_mutex** = PTHREAD_MUTEX_INITIALIZER

Detailed Description

A collection of functions used to register and handle signals.

Definition in file **signal.c**.

Function Documentation

void signal_refresh (int *signal*)

A function to handle receipt of SIGHUP and refresh all web server content.

Parameters:

signal the number of the delivered signal (should only be SIGHUP).

Returns:

This function returns no value.

Definition at line 136 of file signal.c.

References `http_content_refresh()`, `log_error`, `log_info`, `mutex_lock()`, `mutex_unlock()`, and `sig_hup_mutex`.

Referenced by `signal_start()`.

void signal_segfault (int *signal*)

Handle signals that indicate the program was killed because of an invalid operation (including SIGSEGV).

Note:

The signals handled by this function are: SIGSEGV, SIGFPE, SIGBUS, SIGSYS, and SIGABRT. The handler will respond by logging a stack backtrace, and terminating the program with abort().

Parameters:

signal the number of the signal that was received.

Returns:

This function returns no value.

Definition at line 24 of file signal.c.

References log_options, M_LOG_CRITICAL, M_LOG_STACK_TRACE, mm_wipe(), and signal_name().

Referenced by signal_start().

void signal_shutdown (int *signal*)

A function to handle receipt of shutdown signals and allow for graceful exits.

Note:

This function handles the following signals: SIGINT, SIGQUIT, and SIGTERM. The shutdown procedure is as follows: 1. Set status to -1 and sleep for .1 second to allow for normal daemon termination. 2. Signal all worker threads to wake up blocked threads, and sleep one more second. 3. Loop through all possible file descriptors and if the descriptor matches a socket that is bound to a magma server, close it.

Parameters:

signal the number of the signal delivered to the process.

Returns:

This function returns no value.

Definition at line 57 of file signal.c.

References ip_t::family, ip_t::ip4, ip_t::ip6, ip_presentation(), log_critical, log_info, mm_copy(), mm_wipe(), PLACER, queue_signal(), servers_get_count_using_port(), signal_name(), st_char_get(), and status_set().

Referenced by signal_start().

bool_t signal_start (void)

Setup signal masks and register signal handlers for handling shutdowns, program termination, and reloads.

signal.c**Note:**

The following handlers are established: signal_shutdown: SIGINT, SIGQUIT, SIGTERM, SIGHUP
signal_segfault: SIGSEGV, SIGFPE, SIGBUS, SIGSYS signal_refresh: SIGHUP [ignored]: SIGPIPE

Returns:

true if all signal handlers were successfully registered, or false on failure.

Definition at line 182 of file signal.c.

References log_info, mm_wipe(), signal_refresh(), signal_segfault(), and signal_shutdown().

Referenced by process_start().

void signal_status (int *signal*)

A generic worker thread signal handler entry point.

Note:

If the target thread is not in the process of shutting down, display the name of the caught signal.

Parameters:

signal the number of the signal caught by the signal handler.

Returns:

This function returns no value.

Definition at line 122 of file signal.c.

References log_info, signal_name(), and status.

Referenced by signal_thread_start().

bool_t signal_thread_start (void)

Bind a SIGALRM handler for the calling thread.

See also:

signal_status()

Returns:

false on failure or true on success.

Definition at line 157 of file signal.c.

References log_info, mm_wipe(), and signal_status().

Referenced by thread_start().

Variable Documentation**pthread_mutex_t sig_hup_mutex = PTHREAD_MUTEX_INITIALIZER**

Definition at line 15 of file signal.c.

Referenced by signal_refresh().

magma/engine/context/system.c File Reference

Functions used to interface with and configure the operating system.

```
#include "magma.h"
```

Functions

- `uint64_t system_limit_max (int_t resource)`
- *Get the hard system limit for a specified resource.* `uint64_t system_limit_cur (int_t resource)`
- *Get the soft system limit for a specified resource.* `bool_t system_change_root_directory (void)`
- *Perform a chroot() on the directory specified in the config option magma.system.root_directory, if it is set.* `bool_t system_fork_daemon (void)`
- *Daemonize into the background, if the magma.system.daemonize config option is set.* `bool_t system_init_umask (void)`
- *Set the process umask for new file/dir creation to O_RDWR.* `bool_t system_init_core_dumps (void)`
- *Set the magma core dump size rlimit, if magma.system.enable_core_dumps was enabled.* `bool_t system_init_impersonation (void)`
- *Set process privileges to run as a specified user if magma.system.impersonate_user is set.* `bool_t system_init_resource_limits (void)`

Increase process resource limits, if magma.system.increase_resource_limits is set.

Detailed Description

Functions used to interface with and configure the operating system.

Definition in file `system.c`.

Function Documentation

`bool_t system_change_root_directory (void)`

Perform a chroot() on the directory specified in the config option magma.system.root_directory, if it is set.

system.c

Returns:

true on success or false on failure.

Definition at line 72 of file system.c.

References `buflen`, `bufptr`, `log_info`, `magma`, `magma_t::root_directory`, and `magma_t::system`.

Referenced by `process_start()`.

`bool_t system_fork_daemon (void)`

Daemonize into the background, if the magma.system.daemonize config option is set.

Returns:

true inside the child process, or false inside the parent process or if an error occurs.

Definition at line 87 of file system.c.

References magma_t::daemonize, log_info, magma, pid, status_process(), and magma_t::system.

Referenced by process_start().

bool_t system_init_core_dumps (void)

Set the magma core dump size rlimit, if magma.system.enable_core_dumps was enabled.

Returns:

true if core dumps were successfully enabled or false on failure.

Definition at line 144 of file system.c.

References buflen, bufptr, magma_t::core_dump_size_limit, magma_t::enable_core_dumps, log_info, magma, and magma_t::system.

Referenced by process_start().

bool_t system_init_impersonation (void)

Set process privileges to run as a specified user if magma.system.impersonate_user is set.

Note:

This function will set the user id and group id to the specified user, and chdir() to their home directory.

Returns:

true on success or if magma.system.impersonate_user is not set; false on failure to change privileges.

Definition at line 164 of file system.c.

References buflen, bufptr, magma_t::impersonate_user, log_info, magma, mm_alloc(), mm_free(), mm_wipe(), status_process(), and magma_t::system.

Referenced by process_start().

bool_t system_init_resource_limits (void)

Increase process resource limits, if magma.system.increase_resource_limits is set.

Note:

Resource limits will be maximized for magma's virtual address space, data and stack segments, available file descriptors and sub-processes, and allowed file sizes. If output_resource_limits is enabled, the state of the process resource limits will be dumped to the log afterwards.

Returns:

This function always returns true (errors are logged).

Definition at line 233 of file system.c.

References buflen, bufptr, magma_t::config, magma_t::increase_resource_limits, log_info, magma, magma_t::output_resource_limits, and magma_t::system.

Referenced by process_start().

bool_t system_init_umask (void)

Set the process umask for new file/dir creation to O_RDWR.

Returns:

true if the umask was set successfully, or false on failure.

Definition at line 126 of file system.c.

References buflen, bufptr, and log_info.

Referenced by process_start().

uint64_t system_limit_cur (int_t resource)

Get the soft system limit for a specified resource.

Note:

This function will never return a value greater than UINT64_MAX.

See also:

getrlimit64()

Parameters:

resource the system rlimit resource identifier to be queried.

Returns:

-1 on failure, or the system soft limit of the specified resource identifier on success.

Definition at line 48 of file system.c.

References buflen, bufptr, log_info, and log_pedantic.

Referenced by st_realloc().

uint64_t system_limit_max (int_t resource)

Get the hard system limit for a specified resource.

Note:

This function will never return a value greater than UINT64_MAX.

See also:

getrlimit64()

Parameters:

resource the system rlimit resource identifier to be queried.

Returns:

-1 on failure, or the system hard limit of the specified resource identifier on success.

Definition at line 22 of file system.c.

References buflen, bufptr, log_info, and log_pedantic.

Referenced by config_validate_settings().

magma/engine/controller/controller.h File Reference

Functions involved with assigning tasks to worker threads and routing network connections to the proper protocol subsystem.

Functions

- void **dequeue** (void)
- *queue.c* void **enqueue** (void *function, void *data)
- *Push a function on the job queue to be executed asynchronously.* **bool_t queue_init** (void)
- *Create a queue of worker threads and set them into motion.* void **queue_shutdown** (void)
- *Attempt to wake any sleeping workers, wait for all worker threads to shutdown and exit, and destroy the worker queue.* void **queue_signal** (void)
- *Signal all worker threads with SIGALRM to force their return.* void **requeue** (void *function, void *requeue, void *data)
- *Push a function on the job queue to be executed asynchronously.* **bool_t protocol_init** (void)
- *protocol.c* void **protocol_process** (server_t *server, int sockd)

Create a connection object for an accepted connection, and enqueue it to be handled.

Detailed Description

Functions involved with assigning tasks to worker threads and routing network connections to the proper protocol subsystem.

Definition in file **controller.h**.

Function Documentation

void dequeue (void)

queue.c

queue.c

Note:

This is the thread pool entry point called from **queue_init()**.

Returns:

This function returns no value.

Definition at line 84 of file queue.c.

References `queue_t::data`, `queue_t::function`, `log_error`, `mm_free()`, `mutex_lock()`, `mutex_unlock()`, `queue_t::next`, `queue`, `queue_t::requeue()`, `status`, `thread_start()`, and `thread_stop()`.

Referenced by `queue_init()`.

void enqueue (void * function, void * data)

Push a function on the job queue to be executed asynchronously.

Note:

Warning: If this function fails to allocate a new **queue_t** object, the work unit is lost forever.

Parameters:

function a pointer to a function to be executed by the next available worker thread.

data a pointer to an arbitrary block of data to be passed to the specified function on execution.

Returns:

This function returns no value.

Definition at line 74 of file queue.c.

References `requeue()`.

Referenced by `con_reverse_enqueue()`, `decrypt_user_messages()`, `encrypt_user_messages()`, `http_process()`, `http_requeue()`, `imap_process()`, `imap_requeue()`, `meta_check_message_encryption()`, `molten_init()`, `molten_invalid()`, `molten_parse()`, `molten_stats()`, `pop_process()`, `pop_requeue()`, `protocol_enqueue()`, `protocol_process()`, `smtp_process()`, and `smtp_requeue()`.

bool_t protocol_init (void)

protocol.c

protocol.c

Returns:

This function always returns true.

Definition at line 20 of file protocol.c.

References `imap_sort()`, `molten_sort()`, `pop_sort()`, `portal_endpoint_sort()`, and `smtp_sort()`.

Referenced by `process_start()`.

void protocol_process (server_t * server, int sockd)

Create a connection object for an accepted connection, and enqueue it to be handled.

See also:

protocol_secure(), **protocol_enqueue()**

Note:

Depending on whether the server specifies a secure transport layer, **protocol_secure()** or **protocol_enqueue()** will be dispatched.

Parameters:

server a pointer to the server object of the server handling the connection.

sockd the socket descriptor of the newly accepted connection.

Definition at line 124 of file protocol.c.

References `con_init()`, `server_t::context`, `enqueue()`, `log_pedantic`, `net_set_timeout()`, `server_t::network`, `protocol_enqueue()`, `protocol_secure()`, `server_t::ssl`, `SSL_PORT`, `server_t::timeout`, and `server_t::type`.

Referenced by `net_listen()`.

bool_t queue_init (void)

Create a queue of worker threads and set them into motion.

Note:

Up to magma.system.worker_threads number of threads will be created.

Returns:

false on failure or true on success.

Definition at line 149 of file queue.c.

References dequeue(), log_error, magma, mm_alloc(), mutex_init(), queue, queue_shutdown(), stats_increment_by_name(), magma_t::system, thread_launch(), and magma_t::worker_threads.

Referenced by process_start().

void queue_shutdown (void)

Attempt to wake any sleeping workers, wait for all worker threads to shutdown and exit, and destroy the worker queue.

Returns:

This function returns no value.

Definition at line 183 of file queue.c.

References magma, mm_cleanup(), mutex_destroy(), queue, stats_decrement_by_name(), magma_t::system, thread_join(), and magma_t::worker_threads.

Referenced by process_stop(), and queue_init().

void queue_signal (void)

Signal all worker threads with SIGALRM to force their return.

Returns:

This function returns no value.

Definition at line 129 of file queue.c.

References log_info, magma, queue, status, magma_t::system, thread_signal(), and magma_t::worker_threads.

Referenced by signal_shutdown().

void requeue (void * *function*, void * *requeue*, void * *data*)

Push a function on the job queue to be executed asynchronously.

Note:

Warning: If this function fails to allocate a new **queue_t** object, the work unit is lost forever.

Parameters:

function a pointer to a function to be executed by the next available worker thread.

requeue an optional pointer to a requeue function to be called after function is executed.

data a pointer to an arbitrary block of data to be passed to function and/or requeue upon execution.

Returns:

This function returns no value.

Definition at line 38 of file queue.c.

References `queue_t::data`, `queue_t::function`, `log_critical`, `mm_alloc()`, `mutex_lock()`, `mutex_unlock()`, `queue_t::next`, `queue`, and `queue_t::requeue()`.

Referenced by `enqueue()`, `http_process()`, `http_requeue()`, `imap_process()`, `pop_process()`, `sess_release()`, `smtp_data()`, and `smtp_process()`.

magma/engine/controller/protocol.c File Reference

Functions used to route incoming network connections to their designated protocol modules. This layer also tracks overall protocol statistics and establishes SSL connections for connections that arrive on secure ports.

```
#include "magma.h"
```

Functions

- **bool_t protocol_init** (void)
- *Initialize all protocol modules, and prime their command arrays for binary searching.* void **protocol_enqueue** (**connection_t** *con)
- *Enqueue a protocol-specific handler to service a specified connection, and update any statistics accordingly.* void **protocol_secure** (**connection_t** *con)
- *Establish a secure channel for an inbound ssl connection before passing it off to the general server request handler.* void **protocol_process** (**server_t** *server, int sockd)

Create a connection object for an accepted connection, and enqueue it to be handled.

Detailed Description

Functions used to route incoming network connections to their designated protocol modules. This layer also tracks overall protocol statistics and establishes SSL connections for connections that arrive on secure ports.

Definition in file **protocol.c**.

Function Documentation

void protocol_enqueue (connection_t * con)

Enqueue a protocol-specific handler to service a specified connection, and update any statistics accordingly.

Note:

If an invalid protocol is specified, the connection will be destroyed gracefully.

Parameters:

con a pointer to the connection object to be serviced.

Returns:

This function returns no value.

Definition at line 35 of file protocol.c.

References `con_destroy()`, `con_secure()`, `enqueue()`, `HTTP`, `http_init()`, `IMAP`, `imap_init()`, `log_check`, `log_pedantic`, `MOLTEN`, `molten_init()`, `POP`, `pop_init()`, `server_t::protocol`, `connection_t::server`, `SMTP`, `smtp_init()`, `stats_increment_by_name()`, `SUBMISSION`, and `submission_init()`.

Referenced by `protocol_process()`, and `protocol_secure()`.

bool_t protocol_init (void)

Initialize all protocol modules, and prime their command arrays for binary searching.

protocol.c

Returns:

This function always returns true.

Definition at line 20 of file protocol.c.

References `imap_sort()`, `molten_sort()`, `pop_sort()`, `portal_endpoint_sort()`, and `smtp_sort()`.

Referenced by `process_start()`.

void protocol_process (server_t * server, int sockd)

Create a connection object for an accepted connection, and enqueue it to be handled.

See also:

`protocol_secure()`, **`protocol_enqueue()`**

Note:

Depending on whether the server specifies a secure transport layer, **`protocol_secure()`** or **`protocol_enqueue()`** will be dispatched.

Parameters:

server a pointer to the server object of the server handling the connection.

sockd the socket descriptor of the newly accepted connection.

Definition at line 124 of file protocol.c.

References `con_init()`, `server_t::context`, `enqueue()`, `log_pedantic`, `net_set_timeout()`, `server_t::network`, `protocol_enqueue()`, `protocol_secure()`, `server_t::ssl`, `SSL_PORT`, `server_t::timeout`, and `server_t::type`.

Referenced by `net_listen()`.

void protocol_secure (connection_t * con)

Establish a secure channel for an inbound ssl connection before passing it off to the general server request handler.

See also:

`protocol_enqueue()`

Note:

This function destroys the client connection completely and returns silently upon any ssl-related failure.

Parameters:

con the The connection object associated with the inbound ssl connection.

Returns:

This function returns no value.

Definition at line 93 of file protocol.c.

References `connection_t::buffer`, `connection_t::lock`, `log_check`, `log_pedantic`, `mm_free()`, `mutex_destroy()`, `connection_t::network`, `protocol_enqueue()`, `connection_t::server`, `connection_t::sockd`, `connection_t::ssl`, `ssl_alloc()`, `ssl_free()`, and `st_free()`.

Referenced by `protocol_process()`.

magma/engine/controller/queue.c File Reference

Functions used to distribute tasks to available worker threads.

```
#include "magma.h"
```

Data Structures

- struct **queue_t**

Functions

- void **requeue** (void *function, void *requeue, void *data)
- *Push a function on the job queue to be executed asynchronously.* void **enqueue** (void *function, void *data)
- *Push a function on the job queue to be executed asynchronously.* void **dequeue** (void)
- *Wait for work to appear on the queue and then perform the work; if the job is to be requeue'd then requeue it.* void **queue_signal** (void)
- *Signal all worker threads with SIGALRM to force their return.* bool_t **queue_init** (void)
- *Create a queue of worker threads and set them into motion.* void **queue_shutdown** (void)

Attempt to wake any sleeping workers, wait for all worker threads to shutdown and exit, and destroy the worker queue. Variables

- struct {
- sem_t **sema**
- pthread_t * **workers**
- pthread_mutex_t **lock**
- queue_t * **items**
- } **queue**

Detailed Description

Functions used to distribute tasks to available worker threads.

Definition in file **queue.c**.

Function Documentation

void dequeue (void)

Wait for work to appear on the queue and then perform the work; if the job is to be requeue'd then requeue it.

queue.c

Note:

This is the thread pool entry point called from **queue_init()**.

Returns:

This function returns no value.

Definition at line 84 of file queue.c.

References `queue_t::data`, `queue_t::function`, `log_error`, `mm_free()`, `mutex_lock()`, `mutex_unlock()`, `queue_t::next`, `queue`, `queue_t::requeue()`, `status`, `thread_start()`, and `thread_stop()`.

Referenced by `queue_init()`.

void enqueue (void * *function*, void * *data*)

Push a function on the job queue to be executed asynchronously.

Note:

Warning: If this function fails to allocate a new **queue_t** object, the work unit is lost forever.

Parameters:

function a pointer to a function to be executed by the next available worker thread.

data a pointer to an arbitrary block of data to be passed to the specified function on execution.

Returns:

This function returns no value.

Definition at line 74 of file `queue.c`.

References `requeue()`.

Referenced by `con_reverse_enqueue()`, `decrypt_user_messages()`, `encrypt_user_messages()`, `http_process()`, `http_requeue()`, `imap_process()`, `imap_requeue()`, `meta_check_message_encryption()`, `molten_init()`, `molten_invalid()`, `molten_parse()`, `molten_stats()`, `pop_process()`, `pop_requeue()`, `protocol_enqueue()`, `protocol_process()`, `smtp_process()`, and `smtp_requeue()`.

bool_t queue_init (void)

Create a queue of worker threads and set them into motion.

Note:

Up to `magma.system.worker_threads` number of threads will be created.

Returns:

false on failure or true on success.

Definition at line 149 of file `queue.c`.

References `dequeue()`, `log_error`, `magma`, `mm_alloc()`, `mutex_init()`, `queue`, `queue_shutdown()`, `stats_increment_by_name()`, `magma_t::system`, `thread_launch()`, and `magma_t::worker_threads`.

Referenced by `process_start()`.

void queue_shutdown (void)

Attempt to wake any sleeping workers, wait for all worker threads to shutdown and exit, and destroy the worker queue.

Returns:

This function returns no value.

Definition at line 183 of file `queue.c`.

References magma, mm_cleanup(), mutex_destroy(), queue, stats_decrement_by_name(), magma_t::system, thread_join(), and magma_t::worker_threads.

Referenced by process_stop(), and queue_init().

void queue_signal (void)

Signal all worker threads with SIGALRM to force their return.

Returns:

This function returns no value.

Definition at line 129 of file queue.c.

References log_info, magma, queue, status, magma_t::system, thread_signal(), and magma_t::worker_threads.

Referenced by signal_shutdown().

void requeue (void * function, void * requeue, void * data)

Push a function on the job queue to be executed asynchronously.

Note:

Warning: If this function fails to allocate a new **queue_t** object, the work unit is lost forever.

Parameters:

function a pointer to a function to be executed by the next available worker thread.

requeue an optional pointer to a requeue function to be called after function is executed.

data a pointer to an arbitrary block of data to be passed to function and/or requeue upon execution.

Returns:

This function returns no value.

Definition at line 38 of file queue.c.

References queue_t::data, queue_t::function, log_critical, mm_alloc(), mutex_lock(), mutex_unlock(), queue_t::next, queue, and queue_t::requeue().

Referenced by enqueue(), http_process(), http_requeue(), imap_process(), pop_process(), sess_release(), smtp_data(), and smtp_process().

Variable Documentation

queue_t* items

Definition at line 24 of file queue.c.

pthread_mutex_t lock

Definition at line 23 of file queue.c.

struct { ... } queue

Referenced by dequeue(), queue_init(), queue_shutdown(), queue_signal(), and requeue().

sem_t sema

Definition at line 21 of file queue.c.

pthread_t* workers

Definition at line 22 of file queue.c.

magma/engine/engine.h File Reference

The engine module that contains all of the logic to control the overall execution of the system.

```
#include "context/context.h"  
#include "config/config.h"  
#include "status/status.h"  
#include "controller/controller.h"
```

Detailed Description

The engine module that contains all of the logic to control the overall execution of the system.

Definition in file **engine.h**.

magma/engine/status/build.c File Reference

Functions for retrieving the version and build information.

```
#include "magma.h"
```

```
#include "build.h"
```

Functions

- `const char * build_version (void)`
- *Get the magma version string.* `const char * build_stamp (void)`

Get the magma build stamp.

Detailed Description

Functions for retrieving the version and build information.

Definition in file **build.c**.

Function Documentation

const char* build_stamp (void)

Get the magma build stamp.

Returns:

a pointer to a null-terminated string containing the magma build information string.

Definition at line 36 of file build.c.

References MAGMA_STAMP.

Referenced by `args_parse()`, `http_response_options()`, `imap_id()`, and `lib_load()`.

const char* build_version (void)

Get the magma version string.

Returns:

a pointer to a null-terminated string containing the magma version string.

Definition at line 28 of file build.c.

References MAGMA_BUILD.

Referenced by `args_parse()`, `http_response_options()`, `imap_id()`, `imap_init()`, `lib_load()`, and `pop_capa()`.

magma/engine/status/build.h File Reference

Defines

- `#define MAGMA_BUILD "6.0.1"`
 - `#define MAGMA_STAMP "20141223.0408"`
-

Define Documentation

`#define MAGMA_BUILD "6.0.1"`

Definition at line 1 of file build.h.

Referenced by build_version().

`#define MAGMA_STAMP "20141223.0408"`

Definition at line 2 of file build.h.

Referenced by build_stamp().

magma/engine/status/performance.c File Reference

A collection of functions used to measure and track system performance.

```
#include "magma.h"
```

Functions

- `uint64_t perf_rdtsc (void)`
-

Detailed Description

A collection of functions used to measure and track system performance.

Definition in file **performance.c**.

Function Documentation

uint64_t perf_rdtsc (void)

Returns the CPU cycle counter. By calculating variations in the time stamp, we can discover the exact number of elapsed CPU ticks, making it possible to precisely time code execution. Note that in a multithreaded environment its possible that not all of the elapsed ticks were consumed by the code being timed.

Returns:

The number of CPU cycles that have elapsed since boot.

Definition at line 185 of file performance.c.

magma/engine/status/statistics.c File Reference

A collection of functions used to track and access system statistics.

```
#include "magma.h"
```

Functions

- `uint64_t stats_sum_errors` (void)
- *Get the total sum of all error statistics maintained by magma (traditional and derived stats).* `uint64_t derived_count` (void)
- *Get the number of derived statistics that are tracked.* `char * derived_name` (uint64_t position)
- *Get the name of a derived statistic by index.* `uint64_t derived_value` (uint64_t position)
- *Get the value of a derived statistic by index.* `uint64_t stats_get_name_pos` (char *name)
- *Get the index of a statistic by name.* `char * stats_get_name` (uint64_t position)
- *Get the name of a statistic by its index.* `void stats_set_by_name` (char *name, uint64_t value)
- *Provided a statistic by name, set its value.* `void stats_set_by_num` (uint64_t position, uint64_t value)
- *Provided a statistic by index, set its value.* `uint64_t stats_get_value_by_name` (char *name)
- *Provided a statistic by name, get its value.* `uint64_t stats_get_value_by_num` (uint64_t position)
- *Provided a statistic by index, get its value.* `void stats_adjust_by_name` (char *name, int32_t value)
- *Provided a statistic by name, increment its value by a specified amount.* `void stats_adjust_by_num` (uint64_t position, int32_t value)
- *Provided a statistic by index, increment its value by a specified amount.* `void stats_increment_by_name` (char *name)
- *Provided a statistic by name, increment its value by 1.* `void stats_increment_by_num` (uint64_t position)
- *Provided a statistic by index, increment its value by 1.* `void stats_decrement_by_name` (char *name)
- *Provided a statistic by name, decrement its value by 1.* `void stats_decrement_by_num` (uint64_t position)
- *Provided a statistic by index, decrement its value by 1.* `uint64_t stats_get_count` (void)
- *Get the number of statistics being tracked.* `bool_t stats_init` (void)
- *Initialize and reset all statistics counters.* `void stats_shutdown` (void)

Destroy all statistics locks. Variables

- `struct {`
- `size_t count`
- `pthread_mutex_t locks [128][2]`
- `uint64_t values [128]`
- `char * names [128]`
- `} stats`
- `char * derived []`

Detailed Description

A collection of functions used to track and access system statistics.

Definition in file **statistics.c**.

Function Documentation

uint64_t derived_count (void)

Get the number of derived statistics that are tracked.

Returns:

the number of derived statistics being maintained by magma.

Definition at line 184 of file statistics.c.

References derived.

Referenced by derived_name(), molten_stats(), and stats_sum_errors().

char* derived_name (uint64_t *position*)

Get the name of a derived statistic by index.

Parameters:

position the zero-based index of the derived statistic to be queried.

Returns:

NULL on failure, or a pointer to a null-terminated string containing the name of the derived statistic on success.

Definition at line 194 of file statistics.c.

References derived, and derived_count().

Referenced by molten_stats(), and stats_sum_errors().

uint64_t derived_value (uint64_t *position*)

Get the value of a derived statistic by index.

See also:

mm_sec_stats()

Parameters:

position the zero-based index of the derived statistic to be queried.

Returns:

0 on failure, or the value of the specified derived statistic on success.

Definition at line 209 of file statistics.c.

References bytes, items, log_pedantic, mm_sec_stats(), spool_error_stats(), and stats_sum_errors().

Referenced by molten_stats(), and stats_sum_errors().

void stats_adjust_by_name (char * *name*, int32_t *value*)

Provided a statistic by name, increment its value by a specified amount.

Parameters:

name a null-terminated string containing the name of the statistic to be set.
value the value by which to increment the specified statistic.

Returns:

This function returns no value.

Definition at line 361 of file statistics.c.

References mutex_lock(), mutex_unlock(), stats, and stats_get_name_pos().

Referenced by dkim_check(), dkim_create(), meta_user_prune(), obj_cache_prune(), pattern_check(), and spf_check().

void stats_adjust_by_num (uint64_t *position*, int32_t *value*)

Provided a statistic by index, increment its value by a specified amount.

Parameters:

position the zero-based index of the statistic to be set.
value the value by which to increment the specified statistic.

Returns:

This function returns no value.

Definition at line 382 of file statistics.c.

References mutex_lock(), mutex_unlock(), and stats.

void stats_decrement_by_name (char * *name*)

Provided a statistic by name, decrement its value by 1.

Parameters:

name a null-terminated string containing the name of the statistic to be decremented.

Returns:

This function returns no value.

Definition at line 430 of file statistics.c.

References mutex_lock(), mutex_unlock(), stats, and stats_get_name_pos().

Referenced by con_destroy(), and queue_shutdown().

void stats_decrement_by_num (uint64_t *position*)

Provided a statistic by index, decrement its value by 1.

Parameters:

position the zero-based index of the statistic to be decremented.

Returns:

This function returns no value.

Definition at line 450 of file statistics.c.

References mutex_lock(), mutex_unlock(), and stats.

uint64_t stats_get_count (void)

Get the number of statistics being tracked.

Returns:

the total number of statistics counters maintained by magma.

Definition at line 463 of file statistics.c.

References stats.

Referenced by molten_stats(), and stats_sum_errors().

char* stats_get_name (uint64_t *position*)

Get the name of a statistic by its index.

Parameters:

position the zero-based index of the statistic to be queried.

Returns:

0 on failure, or the name of the requested statistic on success.

Definition at line 273 of file statistics.c.

References mutex_lock(), mutex_unlock(), and stats.

Referenced by molten_stats(), and stats_sum_errors().

uint64_t stats_get_name_pos (char * *name*)

Get the index of a statistic by name.

Parameters:

name the name of the statistic to be queried.

Returns:

0 on failure, or the zero-based index of the requested statistic on success.

Definition at line 252 of file statistics.c.

References log_info, mutex_lock(), mutex_unlock(), NULLER, st_cmp_cs_eq(), and stats.

Referenced by stats_adjust_by_name(), stats_decrement_by_name(), stats_get_value_by_name(), stats_increment_by_name(), and stats_set_by_name().

uint64_t stats_get_value_by_name (char * *name*)

Provided a statistic by name, get its value.

Parameters:

name a null-terminated string containing the name of the statistic to be queried.

Returns:

the specified statistic's value, as an unsigned 64 bit integer.

Definition at line 324 of file statistics.c.

References mutex_lock(), mutex_unlock(), stats, and stats_get_name_pos().

uint64_t stats_get_value_by_num (uint64_t *position*)

Provided a statistic by index, get its value.

Parameters:

position the zero-based index of the statistic to be queried.

Returns:

the specified statistic's value, as an unsigned 64 bit integer.

Definition at line 344 of file statistics.c.

References mutex_lock(), mutex_unlock(), and stats.

Referenced by molten_stats(), and stats_sum_errors().

void stats_increment_by_name (char * *name*)

Provided a statistic by name, increment its value by 1.

Parameters:

name a null-terminated string containing the name of the statistic to be incremented.

Returns:

This function returns no value.

Definition at line 396 of file statistics.c.

References mutex_lock(), mutex_unlock(), stats, and stats_get_name_pos().

Referenced by imap_starttls(), pop_starttls(), protocol_enqueue(), queue_init(), register_abuse_check_blocklist(), smtp_starttls(), virus_check(), virus_engine_refresh(), and virus_start().

void stats_increment_by_num (uint64_t *position*)

Provided a statistic by index, increment its value by 1.

Parameters:

position the zero-based index of the statistic to be incremented.

Returns:

This function returns no value.

Definition at line 416 of file statistics.c.

References mutex_lock(), mutex_unlock(), and stats.

bool_t stats_init (void)

Initialize and reset all statistics counters.

Returns:

false on failure or true on success.

Definition at line 472 of file statistics.c.

References log_critical, mm_wipe(), mutex_init(), and stats.

Referenced by process_start().

void stats_set_by_name (char * *name*, uint64_t *value*)

Provided a statistic by name, set its value.

Parameters:

name a null-terminated string containing the name of the statistic to be set.

value the new value of the specified statistic.

Returns:

This function returns no value.

Definition at line 290 of file statistics.c.

References mutex_lock(), mutex_unlock(), stats, and stats_get_name_pos().

Referenced by meta_user_prune(), obj_cache_prune(), virus_engine_refresh(), virus_start(), and virus_stop().

void stats_set_by_num (uint64_t *position*, uint64_t *value*)

Provided a statistic by index, set its value.

Parameters:

position the zero-based index of the statistic to be set.

value the new value of the specified statistic.

Returns:

This function returns no value.

Definition at line 310 of file statistics.c.

References mutex_lock(), mutex_unlock(), and stats.

void stats_shutdown (void)

Destroy all statistics locks.

Returns:

This function returns no value.

Definition at line 495 of file statistics.c.

References mutex_destroy(), and stats.

Referenced by process_stop().

uint64_t stats_sum_errors (void)

Get the total sum of all error statistics maintained by magma (traditional and derived stats).

Note:

Error statistics are all statistics that have a name that begins with "errors." or ends with ".errors".

Returns:

The total sum of all error statistics counters accumulated by magma.

Definition at line 138 of file statistics.c.

References `CONSTANT`, `derived_count()`, `derived_name()`, `derived_value()`, `NULLER`, `st_cmp_cs_ends()`, `st_cmp_cs_eq()`, `st_cmp_cs_starts()`, `stats_get_count()`, `stats_get_name()`, and `stats_get_value_by_num()`.

Referenced by `derived_value()`.

Variable Documentation

size_t count

Definition at line 16 of file statistics.c.

Referenced by `hashed_bucket()`, `hashed_cursor_active()`, `http_parse_pairs()`, `imap_copy()`, `imap_id()`, `imap_range_build()`, `imap_search_messages()`, `inx_count()`, `meta_user_prune()`, `nvp_parse()`, `obj_cache_prune()`, `pop_get_message()`, `pop_stat()`, `portal_endpoint_alert_acknowledge()`, `portal_endpoint_folders_add()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `portal_message_attachments()`, `portal_message_tags_array()`, `portal_parse_flags()`, `portal_parse_json_str_array()`, `portal_parse_sections()`, `register_captcha_random_font()`, `res_row_next()`, `servers_get_count_using_port()`, `stamp_counter_check()`, `str_tok_get_count_bl()`, `tank_count()`, `tok_get_count_bl()`, and `tree_truncate()`.

char* derived[]

```
Initial value: {  
  
    "system.secure.total",  
    "system.secure.allocated",  
    "system.secure.items",  
  
    "core.spool.errors",  
    "errors.total"  
}
```

Definition at line 121 of file statistics.c.

Referenced by `derived_count()`, and `derived_name()`.

pthread_mutex_t locks[128][2]

Definition at line 17 of file statistics.c.

char* names[128]

Definition at line 19 of file statistics.c.

struct { ... } stats

Referenced by portal_endpoint_folders_tags(), stats_adjust_by_name(), stats_adjust_by_num(), stats_decrement_by_name(), stats_decrement_by_num(), stats_get_count(), stats_get_name(), stats_get_name_pos(), stats_get_value_by_name(), stats_get_value_by_num(), stats_increment_by_name(), stats_increment_by_num(), stats_init(), stats_set_by_name(), stats_set_by_num(), and stats_shutdown().

uint64_t values[128]

Definition at line 18 of file statistics.c.

Referenced by imap_status().

magma/web/statistics/statistics.c File Reference

Generate a dynamic web page with statistics on server performance.

```
#include "magma.h"
```

Functions

- void **statistics_process** (connection_t *con)

Display the statistics page to the requesting connection. Variables

- statistics_vp_t portal_stats [portal_stat_users_num_statements]
-

Detailed Description

Generate a dynamic web page with statistics on server performance.

Definition in file **statistics.c**.

Function Documentation

void statistics_process (connection_t * con)

Display the statistics page to the requesting connection.

statistics.c

Parameters:

con a pointer to the connection object across which the server statistics will be transmitted.

Returns:

This function returns no value.

Definition at line 23 of file statistics.c.

References `con_write_st()`, `http_page_free()`, `http_page_get()`, `http_print_500()`, `http_response_header()`, `log_pedantic`, `portal_stat_emails_received_today`, `portal_stat_emails_received_week`, `portal_stat_emails_sent_today`, `portal_stat_emails_sent_week`, `portal_stat_total_users`, `portal_stat_users_checked_email_today`, `portal_stat_users_checked_email_week`, `portal_stat_users_registered_today`, `portal_stat_users_registered_week`, `portal_stat_users_sent_email_today`, `portal_stat_users_sent_email_week`, `st_free()`, `st_length_get()`, `statistics_refresh()`, `statistics_vp_t::val`, `xml_dump_doc()`, `xml_set_xpath_ns()`, and `xml_set_xpath_uint64()`.

Referenced by `http_response()`.

Variable Documentation

statistics_vp_t portal_stats[portal_stat_users_num_statements]

Definition at line 22 of file `datatier.c`.

magma/engine/status/status.c File Reference

Functions used to coordinate system state and worker thread operation and shutdown..

```
#include "magma.h"
```

Functions

- **bool_t status** (void)
- *Check to see if a worker thread should continue processing.* void **status_set** (int value)
- *Set the status level to a specified value.* int **status_get** (void)
- *Get the current status level.* void **status_process** (void)
- *Update magma's basic process information (pid and start time).* uint64_t **status_pid** (void)
- *Get the magma instance's pid.* uint64_t **status_startup** (void)

Get the timestamp corresponding to when magma was started. Variables

- struct {
- pid_t **pid**
- uint64_t **startup**
- } **process**
- int **status_level** = 0
- pthread_mutex_t **status_mutex** = PTHREAD_MUTEX_INITIALIZER

Detailed Description

Functions used to coordinate system state and worker thread operation and shutdown..

Definition in file **status.c**.

Function Documentation

bool_t status (void)

Check to see if a worker thread should continue processing.

status.c

Note:

This function should be called periodically by worker threads.

Returns:

true if the caller should continue processing (status level is positive) or false otherwise.

Definition at line 31 of file status.c.

References mutex_lock(), mutex_unlock(), status_level, and status_mutex.

int status_get (void)

Get the current status level.

See also:

`status()`

Returns:

the current value of the status level.

Definition at line 60 of file `status.c`.

References `mutex_lock()`, `mutex_unlock()`, `status_level`, and `status_mutex`.

uint64_t status_pid (void)

Get the magma instance's pid.

Returns:

the value of magma's pid.

Definition at line 82 of file `status.c`.

References `process`.

void status_process (void)

Update magma's basic process information (pid and start time).

Returns:

This function returns no value.

Definition at line 72 of file `status.c`.

References `process`.

Referenced by `process_start()`, `system_fork_daemon()`, and `system_init_impersonation()`.

void status_set (int *value*)

Set the status level to a specified value.

See also:

`status()`

Parameters:

value the integer value of the new status level.

Returns:

This function returns no value.

Definition at line 48 of file `status.c`.

References `mutex_lock()`, `mutex_unlock()`, `status_level`, and `status_mutex`.

Referenced by `main()`, `net_listen()`, and `signal_shutdown()`.

uint64_t status_startup (void)

Get the timestamp corresponding to when magma was started.

Returns:

the UNIX-style date-time value when magma was started.

Definition at line 90 of file status.c.

References process.

Variable Documentation

pid_t pid

Definition at line 16 of file status.c.

Referenced by process_kill(), and system_fork_daemon().

struct { ... } process

Referenced by status_pid(), status_process(), and status_startup().

uint64_t startup

Definition at line 17 of file status.c.

int status_level = 0

Definition at line 23 of file status.c.

Referenced by status(), status_get(), and status_set().

pthread_mutex_t status_mutex = PTHREAD_MUTEX_INITIALIZER

Definition at line 24 of file status.c.

Referenced by status(), status_get(), and status_set().

magma/engine/status/status.h File Reference

Functions involved with tracking system status and performance.

Functions

- `const char * build_stamp (void)`
 - *Get the magma build stamp.* `const char * build_version (void)`
 - *Get the magma version string.* `uint64_t perf_rdtsc (void)`
 - `bool_t status (void)`
 - *status.c* `int status_get (void)`
 - *Get the current status level.* `uint64_t status_pid (void)`
 - *Get the magma instance's pid.* `void status_process (void)`
 - *Update magma's basic process information (pid and start time).* `void status_set (int value)`
 - *Set the status level to a specified value.* `uint64_t status_startup (void)`
 - *Get the timestamp corresponding to when magma was started.* `uint64_t derived_count (void)`
 - *Get the number of derived statistics that are tracked.* `char * derived_name (uint64_t position)`
 - *Get the name of a derived statistic by index.* `uint64_t derived_value (uint64_t position)`
 - *Get the value of a derived statistic by index.* `bool_t stats_init (void)`
 - *Initialize and reset all statistics counters.* `void stats_shutdown (void)`
 - *Destroy all statistics locks.* `uint64_t stats_get_count (void)`
 - *Get the number of statistics being tracked.* `char * stats_get_name (uint64_t position)`
 - *Get the name of a statistic by its index.* `uint64_t stats_get_value_by_name (char *name)`
 - *Provided a statistic by name, get its value.* `uint64_t stats_get_value_by_num (uint64_t position)`
 - *Provided a statistic by index, get its value.* `void stats_decrement_by_name (char *name)`
 - *Provided a statistic by name, decrement its value by 1.* `void stats_decrement_by_num (uint64_t position)`
 - *Provided a statistic by index, decrement its value by 1.* `void stats_increment_by_name (char *name)`
 - *Provided a statistic by name, increment its value by 1.* `void stats_increment_by_num (uint64_t position)`
 - *Provided a statistic by index, increment its value by 1.* `void stats_set_by_name (char *name, uint64_t value)`
 - *Provided a statistic by name, set its value.* `void stats_set_by_num (uint64_t position, uint64_t value)`
 - *Provided a statistic by index, set its value.* `void stats_adjust_by_name (char *name, int32_t value)`
 - *Provided a statistic by name, increment its value by a specified amount.* `void stats_adjust_by_num (uint64_t position, int32_t value)`
 - *Provided a statistic by index, increment its value by a specified amount.* `stringer_t * host_platform (stringer_t *output)`
 - *Get a description of the local operating system.* `stringer_t * host_version (stringer_t *output)`
- Get release information about the local OS.*
-

Detailed Description

Functions involved with tracking system status and performance.

Definition in file **status.h**.

Function Documentation

const char* build_stamp (void)

Get the magma build stamp.

Returns:

a pointer to a null-terminated string containing the magma build information string.
Definition at line 36 of file build.c.

References MAGMA_STAMP.

Referenced by args_parse(), http_response_options(), imap_id(), and lib_load().

const char* build_version (void)

Get the magma version string.

Returns:

a pointer to a null-terminated string containing the magma version string.
Definition at line 28 of file build.c.

References MAGMA_BUILD.

Referenced by args_parse(), http_response_options(), imap_id(), imap_init(), lib_load(), and pop_capa().

uint64_t derived_count (void)

Get the number of derived statistics that are tracked.

Returns:

the number of derived statistics being maintained by magma.
Definition at line 184 of file statistics.c.

References derived.

Referenced by derived_name(), molten_stats(), and stats_sum_errors().

char* derived_name (uint64_t position)

Get the name of a derived statistic by index.

Parameters:

position the zero-based index of the derived statistic to be queried.

Returns:

NULL on failure, or a pointer to a null-terminated string containing the name of the derived statistic on success.

Definition at line 194 of file statistics.c.

References derived, and derived_count().

Referenced by molten_stats(), and stats_sum_errors().

uint64_t derived_value (uint64_t position)

Get the value of a derived statistic by index.

See also:

`mm_sec_stats()`

Parameters:

position the zero-based index of the derived statistic to be queried.

Returns:

0 on failure, or the value of the specified derived statistic on success.

Definition at line 209 of file statistics.c.

References `bytes`, `items`, `log_pedantic`, `mm_sec_stats()`, `spool_error_stats()`, and `stats_sum_errors()`.

Referenced by `molten_stats()`, and `stats_sum_errors()`.

`stringer_t* host_platform (stringer_t * output)`

Get a description of the local operating system.

host.c

Parameters:

output a pointer to a managed string to receive the OS description.

Returns:

NULL on failure, or the user-specified managed string containing the OS info on success.

Definition at line 21 of file host.c.

References `log_pedantic`, `ns_length_get()`, `st_avail_get()`, and `st_sprint()`.

Referenced by `lib_load()`.

`stringer_t* host_version (stringer_t * output)`

Get release information about the local OS.

Parameters:

output a pointer to a managed string to receive the release information.

Returns:

NULL on failure, or the user-specified managed string containing the release info on success.

Definition at line 47 of file host.c.

References `log_pedantic`, `ns_length_get()`, `st_avail_get()`, and `st_sprint()`.

Referenced by `lib_load()`.

`uint64_t perf_rdtsc (void)`

Returns the CPU cycle counter. By calculating variations in the time stamp, we can discover the exact number of elapsed CPU ticks, making it possible to precisely time code execution. Note that in a multithreaded environment its possible that not all of the elapsed ticks were consumed by the code being timed.

Returns:

The number of CPU cycles that have elapsed since boot.

Definition at line 185 of file performance.c.

void stats_adjust_by_name (char * *name*, int32_t *value*)

Provided a statistic by name, increment its value by a specified amount.

Parameters:

name a null-terminated string containing the name of the statistic to be set.

value the value by which to increment the specified statistic.

Returns:

This function returns no value.

Definition at line 361 of file statistics.c.

References mutex_lock(), mutex_unlock(), stats, and stats_get_name_pos().

Referenced by dkim_check(), dkim_create(), meta_user_prune(), obj_cache_prune(), pattern_check(), and spf_check().

void stats_adjust_by_num (uint64_t *position*, int32_t *value*)

Provided a statistic by index, increment its value by a specified amount.

Parameters:

position the zero-based index of the statistic to be set.

value the value by which to increment the specified statistic.

Returns:

This function returns no value.

Definition at line 382 of file statistics.c.

References mutex_lock(), mutex_unlock(), and stats.

void stats_decrement_by_name (char * *name*)

Provided a statistic by name, decrement its value by 1.

Parameters:

name a null-terminated string containing the name of the statistic to be decremented.

Returns:

This function returns no value.

Definition at line 430 of file statistics.c.

References mutex_lock(), mutex_unlock(), stats, and stats_get_name_pos().

Referenced by con_destroy(), and queue_shutdown().

void stats_decrement_by_num (uint64_t *position*)

Provided a statistic by index, decrement its value by 1.

Parameters:

position the zero-based index of the statistic to be decremented.

Returns:

This function returns no value.

Definition at line 450 of file statistics.c.

References mutex_lock(), mutex_unlock(), and stats.

uint64_t stats_get_count (void)

Get the number of statistics being tracked.

Returns:

the total number of statistics counters maintained by magma.

Definition at line 463 of file statistics.c.

References stats.

Referenced by molten_stats(), and stats_sum_errors().

char* stats_get_name (uint64_t *position*)

Get the name of a statistic by its index.

Parameters:

position the zero-based index of the statistic to be queried.

Returns:

0 on failure, or the name of the requested statistic on success.

Definition at line 273 of file statistics.c.

References mutex_lock(), mutex_unlock(), and stats.

Referenced by molten_stats(), and stats_sum_errors().

uint64_t stats_get_value_by_name (char * *name*)

Provided a statistic by name, get its value.

Parameters:

name a null-terminated string containing the name of the statistic to be queried.

Returns:

the specified statistic's value, as an unsigned 64 bit integer.

Definition at line 324 of file statistics.c.

References mutex_lock(), mutex_unlock(), stats, and stats_get_name_pos().

uint64_t stats_get_value_by_num (uint64_t *position*)

Provided a statistic by index, get its value.

Parameters:

position the zero-based index of the statistic to be queried.

Returns:

the specified statistic's value, as an unsigned 64 bit integer.

Definition at line 344 of file statistics.c.

References mutex_lock(), mutex_unlock(), and stats.

Referenced by molten_stats(), and stats_sum_errors().

void stats_increment_by_name (char * *name*)

Provided a statistic by name, increment its value by 1.

Parameters:

name a null-terminated string containing the name of the statistic to be incremented.

Returns:

This function returns no value.

Definition at line 396 of file statistics.c.

References mutex_lock(), mutex_unlock(), stats, and stats_get_name_pos().

Referenced by imap_starttls(), pop_starttls(), protocol_enqueue(), queue_init(), register_abuse_check_blocklist(), smtp_starttls(), virus_check(), virus_engine_refresh(), and virus_start().

void stats_increment_by_num (uint64_t *position*)

Provided a statistic by index, increment its value by 1.

Parameters:

position the zero-based index of the statistic to be incremented.

Returns:

This function returns no value.

Definition at line 416 of file statistics.c.

References mutex_lock(), mutex_unlock(), and stats.

bool_t stats_init (void)

Initialize and reset all statistics counters.

Returns:

false on failure or true on success.

Definition at line 472 of file statistics.c.

References log_critical, mm_wipe(), mutex_init(), and stats.

Referenced by process_start().

void stats_set_by_name (char * *name*, uint64_t *value*)

Provided a statistic by name, set its value.

Parameters:

name a null-terminated string containing the name of the statistic to be set.

value the new value of the specified statistic.

Returns:

This function returns no value.

Definition at line 290 of file statistics.c.

References mutex_lock(), mutex_unlock(), stats, and stats_get_name_pos().

Referenced by meta_user_prune(), obj_cache_prune(), virus_engine_refresh(), virus_start(), and virus_stop().

void stats_set_by_num (uint64_t *position*, uint64_t *value*)

Provided a statistic by index, set its value.

Parameters:

position the zero-based index of the statistic to be set.

value the new value of the specified statistic.

Returns:

This function returns no value.

Definition at line 310 of file statistics.c.

References mutex_lock(), mutex_unlock(), and stats.

void stats_shutdown (void)

Destroy all statistics locks.

Returns:

This function returns no value.

Definition at line 495 of file statistics.c.

References mutex_destroy(), and stats.

Referenced by process_stop().

bool_t status (void)

status.c

status.c

Note:

This function should be called periodically by worker threads.

Returns:

true if the caller should continue processing (status level is positive) or false otherwise.

Definition at line 31 of file status.c.

References mutex_lock(), mutex_unlock(), status_level, and status_mutex.

int status_get (void)

Get the current status level.

See also:

status()

Returns:

the current value of the status level.

Definition at line 60 of file status.c.

References mutex_lock(), mutex_unlock(), status_level, and status_mutex.

uint64_t status_pid (void)

Get the magma instance's pid.

Returns:

the value of magma's pid.

Definition at line 82 of file status.c.

References process.

void status_process (void)

Update magma's basic process information (pid and start time).

Returns:

This function returns no value.

Definition at line 72 of file status.c.

References process.

Referenced by process_start(), system_fork_daemon(), and system_init_impersonation().

void status_set (int value)

Set the status level to a specified value.

See also:

status()

Parameters:

value the integer value of the new status level.

Returns:

This function returns no value.

Definition at line 48 of file status.c.

References mutex_lock(), mutex_unlock(), status_level, and status_mutex.

Referenced by main(), net_listen(), and signal_shutdown().

uint64_t status_startup (void)

Get the timestamp corresponding to when magma was started.

Returns:

the UNIX-style date-time value when magma was started.

Definition at line 90 of file status.c.

References process.

magma/magma.c File Reference

The main executable entry point.

```
#include "magma.h"
```

Functions

- `int kill_all` (int *signal*, int *names*, char ***namelist*, struct passwd **pwent*)
 - `int main` (int *argc*, char **argv*[])
-

Detailed Description

The main executable entry point.

Definition in file **magma.c**.

Function Documentation

`int kill_all` (int *signal*, int *names*, char ** *namelist*, struct passwd * *pwent*)

`int main` (int *argc*, char * *argv*[])

Definition at line 17 of file magma.c.

References `args_parse()`, `cache_flush()`, `net_listen()`, `process_start()`, `process_stop()`, and `status_set()`.

magma/magma.h File Reference

The global include file. This header includes both system headers and Magma module headers.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <stddef.h>
#include <limits.h>
#include <signal.h>
#include <string.h>
#include <dirent.h>
#include <time.h>
#include <pwd.h>
#include <errno.h>
#include <fcntl.h>
#include <inttypes.h>
#include <pthread.h>
#include <stdarg.h>
#include <dlfcn.h>
#include <execinfo.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/resource.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <sys/utsname.h>
#include <sys/epoll.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <arpa/nameser.h>
#include <netdb.h>
#include <resolv.h>
#include <regex.h>
#include <ftw.h>
#include <search.h>
#include <semaphore.h>
#include <sys/mman.h>
#include <gnu/libc-version.h>
#include <spf.h>
#include <spf_dns_zone.h>
#include <clamav.h>
#include <mysql.h>
#include <openssl/conf.h>
#include <openssl/ssl.h>
#include <openssl/crypto.h>
#include <openssl/rand.h>
#include <openssl/engine.h>
#include <openssl/x509.h>
#include <openssl/x509v3.h>
```

```

#include <openssl/ec.h>
#include <openssl/dh.h>
#include <openssl/err.h>
#include <lzoconf.h>
#include <lzodefs.h>
#include <lzoutil.h>
#include <lzo1x.h>
#include <xmlmemory.h>
#include <tree.h>
#include <valid.h>
#include <xpath.h>
#include <xpathInternals.h>
#include <parserInternals.h>
#include <xmlerror.h>
#include <zlib.h>
#include <bzlib.h>
#include <tcutil.h>
#include <tcadb.h>
#include <tchdb.h>
#include <tcbdb.h>
#include <libmemcached/memcached.h>
#include <dkim.h>
#include <libdspam.h>
#include <mysql_drv.h>
#include <jansson.h>
#include <gd.h>
#include <png.h>
#include <jpeglib.h>
#include <ft2build.h>
#include "core/core.h"
#include "providers/providers.h"
#include "engine/engine.h"
#include "network/network.h"
#include "objects/objects.h"
#include "servers/servers.h"
#include "web/web.h"
#include "queries.h"

```

Defines

- `#define __USE_GNU`
- `#define lint`
- `#define CONFIG_DEFAULT ""`
- `#define LOGDIR "~/"`
- `#define bufptr (char *)&(threadBuffer)`
- `#define buflen sizeof(threadBuffer)`

Variables

- `magma_t magma`
- `__thread char threadBuffer [1024]`

Detailed Description

The global include file. This header includes both system headers and Magma module headers.

Definition in file **magma.h**.

Define Documentation

#define __USE_GNU

Definition at line 16 of file magma.h.

#define buflen sizeof(threadBuffer)

Definition at line 150 of file magma.h.

Referenced by client_write(), con_print(), con_reverse_lookup(), con_write_bl(), ip_presentation(), net_listen(), net_set_buffer_length(), net_set_keepalive(), net_set_linger(), net_set_nodelay(), net_set_non_blocking(), net_set_reuseable_address(), net_set_timeout(), rand_get_int16(), rand_get_int32(), rand_get_int64(), rand_get_int8(), rand_get_uint16(), rand_get_uint32(), rand_get_uint64(), rand_get_uint8(), servers_validate(), spool_check_file(), ssl_alloc(), ssl_print(), ssl_read(), ssl_write(), system_change_root_directory(), system_init_core_dumps(), system_init_impersonation(), system_init_resource_limits(), system_init_umask(), system_limit_cur(), and system_limit_max().

#define bufptr (char *)&(threadBuffer)

Definition at line 149 of file magma.h.

Referenced by client_write(), con_print(), con_reverse_lookup(), con_write_bl(), ip_presentation(), net_listen(), net_set_buffer_length(), net_set_keepalive(), net_set_linger(), net_set_nodelay(), net_set_non_blocking(), net_set_reuseable_address(), net_set_timeout(), rand_get_int16(), rand_get_int32(), rand_get_int64(), rand_get_int8(), rand_get_uint16(), rand_get_uint32(), rand_get_uint64(), rand_get_uint8(), servers_validate(), spool_check_file(), ssl_alloc(), ssl_print(), ssl_read(), ssl_write(), system_change_root_directory(), system_init_core_dumps(), system_init_impersonation(), system_init_resource_limits(), system_init_umask(), system_limit_cur(), and system_limit_max().

#define CONFIG_DEFAULT ""

Definition at line 115 of file magma.h.

#define lint

Definition at line 110 of file magma.h.

#define LOGDIR "~/"

Definition at line 116 of file magma.h.

Variable Documentation

magma_t magma

Definition at line 17 of file global.c.

Referenced by args_parse(), cache_alloc(), cache_free(), cache_output_settings(), cache_start(), cache_stop(), cache_validate(), con_init_network_buffer(), config_fetch_host_number(), config_fetch_settings(), contact_business(), credential_alloc_auth(), credential_username(), dkim_create(), http_body(), http_content_load_fonts(), http_content_refresh(), http_content_start(), http_data_value_parse(), http_load_file(), http_parse_header(), http_parse_method(), http_print_301(), http_response(), http_response_connection(), http_response_header(), http_response_options(), imap_append_message(), imap_command_parser(), lib_load(), lib_unload(), log_internal(), log_rotate(), log_start(), mail_add_forward_headers(), mail_add_outbound_headers(), mail_create_directory(), mail_db_insert_duplicate_message(), mail_db_insert_message(), mail_message_cleanup(), mail_message_path(), mm_sec_start(), net_init(), net_listen(), portal_endpoint(), portal_endpoint_error(), portal_endpoint_response(), portal_process(), portal_upload(), process_start(), process_stop(), queue_init(), queue_shutdown(), queue_signal(), rand_start(), rand_thread_start(), register_business_step2(), register_captcha_random_font(), register_data_insert_user(), relay_alloc(), relay_counter(), relay_free(), relay_output_settings(), relay_validate(), servers_alloc(), servers_encryption_start(), servers_encryption_stop(), servers_free(), servers_get_by_socket(), servers_get_count_using_port(), servers_network_start(), servers_network_stop(), servers_output_settings(), servers_validate(), sess_create(), sess_get(), sess_token(), smtp_accept_message(), smtp_add_bypass_entry(), smtp_bounce(), smtp_bypass_check(), smtp_check_rbl(), smtp_client_connect(), smtp_client_send_helo(), smtp_data(), smtp_ehlo(), smtp_mail_from(), smtp_parse_helo_domain(), smtp_parse_mail_from_path(), smtp_parse_rcpt_to(), smtp_rcpt_to(), spf_start(), spf_stop(), spool_path(), sql_open(), sql_start(), sql_stop(), ssl_start(), st_alloc_opts(), st_realloc(), stmt_start(), stmt_stop(), system_change_root_directory(), system_fork_daemon(), system_init_core_dumps(), system_init_impersonation(), system_init_resource_limits(), tank_start(), thread_launch(), virus_check(), virus_engine_create(), virus_engine_refresh(), virus_sigs_total(), virus_start(), and virus_stop().

__thread char threadBuffer[1024]

Definition at line 16 of file global.c.

magma/network/addresses.c File Reference

A collection of functions used to allocate, configure and destroy connection structures.

```
#include "magma.h"
```

Functions

- **ip_t * ip_copy** (**ip_t** *dst, **ip_t** *src)
- *Create a copy of an IP address object.* **octet_t ip_octet** (**ip_t** *address, **int_t** position)
- *Extract a specified 8 bit octet from an IPv4 or IPv6 address.* **segment_t ip_segment** (**ip_t** *address, **int_t** position)
- *Extract a specified 16 bit segment from an IPv4 or IPv6 address.* **uint32_t ip_word** (**ip_t** *address, **int_t** position)
- *Get a specified 32-bit segment of an IP address.* **stringer_t * ip_subnet** (**ip_t** *address, **stringer_t** *output)
- *Display the simple subnet string for an IP address.* **stringer_t * ip_presentation** (**ip_t** *address, **stringer_t** *output)
- **stringer_t * ip_standard** (**ip_t** *address, **stringer_t** *output)
- *Convert an IP address structure to string representation.* **stringer_t * ip_reversed** (**ip_t** *address, **stringer_t** *output)
- *Get a reversed-IP string, for use in an RBL lookup.* **bool_t ip_address_equal** (**ip_t** *ip1, **ip_t** *ip2)
- *Determine whether two IP addresses are equal.* **ip_t * con_addr** (**connection_t** *con, **ip_t** *output)
- *Return the IP address information of the remote end of a client connection.* **stringer_t * con_addr_presentation** (**connection_t** *con, **stringer_t** *output)
- *Return a textual representation of the IP address of a specified connection handle.* **stringer_t * con_addr_standard** (**connection_t** *con, **stringer_t** *output)
- *Get the IP address string for the client connection.* **stringer_t * con_addr_reversed** (**connection_t** *con, **stringer_t** *output)
- *Get the reversed-IP address string for a client connection.* **stringer_t * con_addr_subnet** (**connection_t** *con, **stringer_t** *output)
- *Get the subnet string for a specified connection.* **uint32_t con_addr_word** (**connection_t** *con, **int_t** position)
- *Get a specified 32-bit segment of a connection's peer IP address.* **octet_t con_addr_octet** (**connection_t** *con, **int_t** position)
- *Get a specified 8-bit octet of a connection's peer IP address.* **segment_t con_addr_segment** (**connection_t** *con, **int_t** position)
- *Extract a specified 16 bit segment from a connection's peer IP address.* **bool_t ip_str_addr** (**chr_t** *ipstr, **ip_t** *out)
- *Convert an IP address string to an IP address object.* **bool_t ip_str_subnet** (**chr_t** *substr, **subnet_t** *out)
- *Convert a string to an IP address object.* **bool_t ip_matches_subnet** (**subnet_t** *subnet, **ip_t** *addr)

Determines whether a given IP address matches a subnet mask.

Detailed Description

A collection of functions used to allocate, configure and destroy connection structures.

Definition in file **addresses.c**.

Function Documentation

ip_t* con_addr (**connection_t** * con, **ip_t** * output)

Return the IP address information of the remote end of a client connection.

addresses.c

Parameters:

con the input client connection.

output a pointer to an **ip_t** structure to receive the remote IP address, which is allocated for the caller if output is NULL.

Returns:

NULL on error, or a pointer to the IP address of the remote host.

Definition at line 395 of file addresses.c.

References `ip_t::family`, `ip_t::ip4`, `ip_t::ip6`, `log_pedantic`, `mm_alloc()`, `mm_copy()`, `mm_free()`, `connection_t::network`, and `connection_t::sockd`.

Referenced by `con_addr_octet()`, `con_addr_presentation()`, `con_addr_reversed()`, `con_addr_segment()`, `con_addr_standard()`, `con_addr_subnet()`, `con_addr_word()`, `sess_create()`, `sess_get()`, `smtp_bypass_check()`, and `smtp_rcpt_to()`.

octet_t con_addr_octet (connection_t * con, int_t position)

Get a specified 8-bit octet of a connection's peer IP address.

Parameters:

con a pointer to the connection object to be examined.

position a zero-based index into the 8-bit octets that comprise the target IP address.

Returns:

-1 on error, or the specified 8-bit octet of the passed address on success.

Definition at line 525 of file addresses.c.

References `con_addr()`, and `ip_octet()`.

stringer_t* con_addr_presentation (connection_t * con, stringer_t * output)

Return a textual representation of the IP address of a specified connection handle.

Parameters:

con the remote connection to be queried.

output a managed string to store the result, which will be allocated for the caller if output is NULL.

Returns:

NULL on failure, or a pointer to a managed string containing a textual representation of the IP address.

Definition at line 435 of file addresses.c.

References `con_addr()`, and `ip_presentation()`.

Referenced by `contact_abuse_checks()`, `contact_abuse_increment_history()`, `contact_business()`, `imap_id()`, `imap_login()`, `mail_add_inbound_headers()`, `mail_add_outbound_headers()`, `pop_pass()`, `portal_meta()`, `register_abuse_checks()`, `register_data_insert_user()`, `register_session_cache()`, `register_session_get()`, and `smtp_rcpt_to()`.

stringer_t* con_addr_reversed (connection_t * con, stringer_t * output)

Get the reversed-IP address string for a client connection.

Parameters:

con the client connection to be queried.

output a managed string to receive the output, which will be allocated for the caller if passed as NULL.

Returns:

NULL on failure, or a pointer to a managed string containing a textual representation of the IP address on success.

Definition at line 471 of file addresses.c.

References `con_addr()`, and `ip_reversed()`.

Referenced by `smtp_check_greylist()`, and `smtp_check_rbl()`.

segment_t con_addr_segment (connection_t * *con*, int_t *position*)

Extract a specified 16 bit segment from a connection's peer IP address.

Parameters:

con a pointer to the connection object to be examined.

position the zero-indexed (starting at least-significant word) 16-bit segment of the IP address to be evaluated.

Returns:

-1 on failure, or the 32 bit-widened segment extracted from the supplied IP address.

Definition at line 543 of file addresses.c.

References `con_addr()`, and `ip_segment()`.

stringer_t* con_addr_standard (connection_t * *con*, stringer_t * *output*)

Get the IP address string for the client connection.

Parameters:

con the client connection to be queried.

output a managed string to receive the output, which will be allocated for the caller if output is NULL.

Returns:

NULL on failure, or a pointer to a managed string containing a textual representation of the IP address.

Definition at line 453 of file addresses.c.

References `con_addr()`, and `ip_standard()`.

Referenced by `register_abuse_check_blocklist()`, `register_abuse_checks()`, and `register_abuse_increment_history()`.

stringer_t* con_addr_subnet (connection_t * *con*, stringer_t * *output*)

Get the subnet string for a specified connection.

Parameters:

con the client connection to be queried.

output a managed string that will store the output of the subnet string lookup.

Returns:

NULL on failure, or a managed string containing a textual representation of a subnet address on success.

Definition at line 489 of file addresses.c.

References `con_addr()`, and `ip_subnet()`.

Referenced by `smtp_check_receive_quota()`, and `smtp_update_receive_stats()`.

uint32_t con_addr_word (connection_t * con, int_t position)

Get a specified 32-bit segment of a connection's peer IP address.

Parameters:

con a pointer to the connection object to be examined.

position a zero-based index into the 32-bit word(s) that comprise the target IP address.

Returns:

-1 if the connecting address can't be looked up, 0 on general error, or the specified 32-bit segment of the passed address on success.

Definition at line 507 of file addresses.c.

References `con_addr()`, and `ip_word()`.

Referenced by `portal_endpoint()`, `portal_process()`, and `portal_upload()`.

bool_t ip_address_equal (ip_t * ip1, ip_t * ip2)

Determine whether two IP addresses are equal.

Parameters:

ip1 a pointer to the first ip address object in the comparison.

ip2 a pointer to the second ip address object in the comparison.

Returns:

true if the two addresses are equal, or false if they are not.

Definition at line 373 of file addresses.c.

References `ip_t::family`, `ip_t::ip4`, and `ip_t::ip6`.

Referenced by `sess_get()`.

ip_t* ip_copy (ip_t * dst, ip_t * src)

Create a copy of an IP address object.

LOW: Create a set of `sock_addr_*` functions and update the `con_addr_*` functions to use use them. Then add a matching set of `sock_peer_*` and `con_peer_*` functions based around the `getpeername()` function. Add functions for getting a socket's port number as well. Also note that `peer_should_` mean the remote address, and `addr_should_` mean the local address.

```
ip_t ip; char working[64]; struct stat64 info; struct sockaddr_in6 saddr; socklen_t len = sizeof(struct
sockaddr_in6);
```

```
if (!fstat64(connection, &info) && S_ISSOCK(info.st_mode) && !(ret = getpeername(connection,
&saddr, &len))) { if (len == sizeof(struct sockaddr_in6) && saddr.sin6_family == AF_INET6) {
mm_copy(&(ip.ip6), &(saddr.sin6_addr), sizeof(struct in6_addr)); ip.family = AF_INET6;
log_info("%s:%u accepted.", st_char_get(ip_presentation(&ip, PLACER(working, 64))),
ntohs(saddr.sin6_port)); } else if (len == sizeof(struct sockaddr_in) && ((struct sockaddr_in
*)&saddr)->sin_family == AF_INET) { mm_copy(&(ip.ip4), &(((struct sockaddr_in
*)&saddr)->sin_addr), sizeof(struct in_addr)); ip.family = AF_INET; log_info("%s:%u accepted.",
st_char_get(ip_presentation(&ip, PLACER(working, 64))), ntohs(((struct sockaddr_in
*)&saddr)->sin_port)); } }
```

Parameters:

dst a pointer to the destination IP address object to receive the copy.

src a pointer to the source IP address object to be copied.

NULL on failure, or a pointer to the destination IP address buffer on success.

Definition at line 45 of file addresses.c.

References mm_copy().

Referenced by ip_str_subnet(), and sess_create().

bool_t ip_matches_subnet (subnet_t * *subnet*, ip_t * *addr*)

Determines whether a given IP address matches a subnet mask.

Parameters:

subnet a pointer to a subnet that the address will be compared against.

addr a pointer to the IP address of interest.

Returns:

true if the specified IP address falls into the subnet, or false if it does not.

Definition at line 671 of file addresses.c.

References subnet_t::address, ip_t::family, ip_t::ip, and subnet_t::mask.

Referenced by smtp_bypass_check().

octet_t ip_octet (ip_t * *address*, int_t *position*)

Extract a specified 8 bit octet from an IPv4 or IPv6 address.

Parameters:

address the IP address object to be examined.

position the zero-indexed (starting at least-significant byte) byte number of the IP address to be evaluated.

Returns:

-1 on failure, or the 16 bit-widened octet extracted from the supplied IP address.

Definition at line 60 of file addresses.c.

References ip_t::family, ip_t::ip4, and ip_t::ip6.

Referenced by con_addr_octet().

stringer_t* ip_presentation (ip_t * address, stringer_t * output)

Definition at line 184 of file addresses.c.

References buflen, bufptr, ip_t::family, ip_t::ip4, ip_t::ip6, log_pedantic, ns_length_get(), st_alloc, st_avail_get(), st_char_get(), st_free(), st_length_set(), st_valid_destination(), and st_valid_tracked().

Referenced by con_addr_presentation(), and signal_shutdown().

stringer_t* ip_reversed (ip_t * address, stringer_t * output)

Get a reversed-IP string, for use in an RBL lookup.

Parameters:

address a pointer to the IP address to be displayed as a string.

output a pointer to the managed string to receive the reversed-IP string, which will be allocated for the caller if output is NULL.

Returns:

NULL on failure, or a pointer to a managed string containing the reversed-IP string on success.

Definition at line 301 of file addresses.c.

References ip_t::family, ip_t::ip4, ip_t::ip6, log_pedantic, st_alloc, st_avail_get(), st_char_get(), st_length_set(), st_sprint(), st_valid_destination(), and st_valid_tracked().

Referenced by con_addr_reversed().

segment_t ip_segment (ip_t * address, int_t position)

Extract a specified 16 bit segment from an IPv4 or IPv6 address.

Parameters:

address the IP address object to be examined.

position the 0-indexed (starting at least-significant word) 16-bit segment of the IP address to be evaluated.

Returns:

-1 on failure, or the 32 bit-widened segment extracted from the supplied IP address.

Definition at line 85 of file addresses.c.

References ip_t::family, ip_t::ip4, and ip_t::ip6.

Referenced by con_addr_segment().

stringer_t* ip_standard (ip_t * address, stringer_t * output)

Convert an IP address structure to string representation.

Parameters:

address a pointer to the IP address to be converted.

output a pointer to the managed string to receive the converted value, which will be allocated for the caller if output is NULL.

Returns:

NULL on failure, or a pointer to a managed string containing a textual representation of the IP address.
Definition at line 226 of file addresses.c.

References `ip_t::family`, `ip_t::ip4`, `ip_t::ip6`, `log_pedantic`, `st_alloc`, `st_avail_get()`, `st_char_get()`, `st_free()`, `st_length_set()`, `st_sprint()`, `st_valid_destination()`, `st_valid_tracked()`, and `st_wipe()`.

Referenced by `con_addr_standard()`.

bool_t ip_str_addr (chr_t * *ipstr*, ip_t * *out*)

Convert an IP address string to an IP address object.

Parameters:

ipstr a pointer to a null-terminated string containing the IP string to be parsed.
out a pointer to an IP address object that will receive result of the operation.

Returns:

true if the input string was a valid IP address or false if it was unable to be parsed.

Definition at line 561 of file addresses.c.

References `ip_t::family`, `ip_t::ip4`, `ip_t::ip6`, and `ns_length_get()`.

Referenced by `ip_str_subnet()`.

bool_t ip_str_subnet (chr_t * *substr*, subnet_t * *out*)

Convert a string to an IP address object.

Parameters:

ipstr a pointer to a null-terminated string containing the IP string to be parsed.
out a pointer to an IP address object that will receive result of the operation.

Returns:

true if the input string was a valid IP address or false if it was unable to be parsed.

Definition at line 611 of file addresses.c.

References `subnet_t::address`, `ip_t::family`, `ip_copy()`, `ip_str_addr()`, `subnet_t::mask`, `ns_free()`, `ns_import()`, `ns_length_get()`, `NULLER`, `pl_char_get()`, `pl_length_get()`, `placer_t`, `tok_get_count_st()`, `tok_get_ns()`, and `uint8_conv_bl()`.

Referenced by `smtp_add_bypass_entry()`.

stringer_t* ip_subnet (ip_t * *address*, stringer_t * *output*)

Display the simple subnet string for an IP address.

Note:

IPv4 addresses will yield a /24 subnet address, and IPv6 addresses will result in a /48 subnet address.
Neither address will be displayed with the trailing zero-octet(s).

Parameters:

address a pointer to the ip address to be examined.

output a pointer to a managed string to receive the subnet string, or if passed as NULL, it will be allocated for the caller.

Returns:

NULL on failure, or a pointer to a managed string containing the result on success.

Definition at line 135 of file addresses.c.

References `ip_t::family`, `ip_t::ip4`, `ip_t::ip6`, `log_pedantic`, `st_alloc`, `st_avail_get()`, `st_free()`, `st_length_get()`, `st_sprint()`, `st_valid_avail()`, and `st_valid_destination()`.

Referenced by `con_addr_subnet()`.

uint32_t ip_word (ip_t * *address*, int_t *position*)

Get a specified 32-bit segment of an IP address.

Note:

This function operates on both ipv4 and ipv6 addresses.

Parameters:

address a pointer to the IP address object to be examined.

position a zero-based index into the 32-bit word(s) that comprise the target IP address.

Returns:

0 on error, or the specified 32-bit segment of the passed address.

Definition at line 113 of file addresses.c.

References `ip_t::family`, `ip_t::ip4`, and `ip_t::ip6`.

Referenced by `con_addr_word()`.

magma/network/clients.c File Reference

Functions for handling network client connections.

```
#include "magma.h"
```

Functions

- **int_t client_status** (**client_t** *client)
- *Get the status of a network client.* **int_t client_secure** (**client_t** *client)
- *Establish an ssl connection with a network client instance.* **client_t * client_connect** (**chr_t** *host, uint32_t port)
- *Establish a network client connection to a remote host.* void **client_close** (**client_t** *client)

Close a network client connection.

Detailed Description

Functions for handling network client connections.

Definition in file **clients.c**.

Function Documentation

void client_close (**client_t** * *client*)

Close a network client connection.

clients.c

Note:

If ssl is in use, this will also destroy the overlying ssl session.

Returns:

This function returns no value.

Definition at line 137 of file clients.c.

References client_t::buffer, mm_free(), client_t::sockd, client_t::ssl, ssl_free(), and st_cleanup().

Referenced by smtp_client_close(), and smtp_client_connect().

client_t* client_connect (**chr_t** * *host*, **uint32_t** *port*)

Establish a network client connection to a remote host.

Parameters:

host a pointer to a null-terminated string containing the hostname of the remote server.

port the port number of the server to which the connection will be established.

Returns:

NULL on failure or a pointer to a newly initialized network client object for the connection upon success.

Definition at line 62 of file clients.c.

References `client_t::buffer`, `FOREIGNDATA`, `JOINTED`, `client_t::line`, `log_pedantic`, `MEMORYBUF`, `mm_alloc()`, `mm_free()`, `PLACER_T`, `client_t::sockd`, `st_alloc`, `STACK`, and `client_t::status`.

Referenced by `smtp_client_connect()`.

`int_t client_secure (client_t * client)`

Establish an ssl connection with a network client instance.

Parameters:

client a pointer to the network client object to have its transport security upgraded.

Returns:

-1 on failure or 0 on success.

Definition at line 37 of file `clients.c`.

References `client_t::sockd`, `client_t::ssl`, `ssl_client_create()`, and `client_t::status`.

Referenced by `smtp_client_connect()`.

`int_t client_status (client_t * client)`

Get the status of a network client.

Parameters:

client a pointer to the network client object to be queried.

Returns:

-1 on error state, 0 for unknown status, or 1 if connected.

Definition at line 21 of file `clients.c`.

References `client_t::sockd`, and `client_t::status`.

magma/network/connections.c File Reference

A collection of functions to allocate, configure and destroy connection structures.

```
#include "magma.h"
```

Functions

- **int_t con_secure** (**connection_t** *con)
- *Return the security level of a specified connection.* **int_t con_status** (**connection_t** *con)
- *Return the status of a specified connection.* **void con_destroy** (**connection_t** *con)
- *Destroy and free a generic connection object after executing its protocol-specific destructor; update any statistics accordingly.* **uint64_t con_increment_refs** (**connection_t** *con)
- *Increment a connection object's reference counter.* **uint64_t con_decrement_refs** (**connection_t** *con)
- *Decrement a connection object's reference counter.* **bool_t con_init_network_buffer** (**connection_t** *con)
- *Allocate and initialize a new network buffer for a connection, if it is not already associated with one.*
connection_t * con_init (int cond, **server_t** *server)

Create a new connection object for a client connection.

Detailed Description

A collection of functions to allocate, configure and destroy connection structures.

Definition in file **connections.c**.

Function Documentation

uint64_t con_decrement_refs (**connection_t** * con)

Decrement a connection object's reference counter.

connections.c

Parameters:

con the connection object being referenced.

Returns:

an integer containing the new number of references to the specified connection.

Definition at line 149 of file connections.c.

References connection_t::lock, mutex_lock(), mutex_unlock(), and connection_t::refs.

Referenced by con_destroy().

void con_destroy (**connection_t** * con)

Destroy and free a generic connection object after executing its protocol-specific destructor; update any statistics accordingly.

Parameters:

con a pointer to the connection to be destroyed.

Returns:

This function returns no value.

Definition at line 53 of file connections.c.

References `connection_t::buffer`, `con_decrement_refs()`, `HTTP`, `http_session_destroy()`, `IMAP`, `imap_session_destroy()`, `connection_t::lock`, `mm_free()`, `MOLTEN`, `molten_session_destroy()`, `mutex_destroy()`, `connection_t::network`, `POP`, `pop_session_destroy()`, `server_t::protocol`, `connection_t::reverse`, `connection_t::server`, `SMTP`, `smtp_session_destroy()`, `connection_t::sockd`, `connection_t::ssl`, `ssl_free()`, `st_cleanup()`, `stats_decrement_by_name()`, and `SUBMISSION`.

Referenced by `con_reverse_lookup()`, `http_close()`, `imap_logout()`, `molten_quit()`, `pop_quit()`, `protocol_enqueue()`, and `smtp_quit()`.

uint64_t con_increment_refs (connection_t * con)

Increment a connection object's reference counter.

Parameters:

con the connection object being referenced.

Returns:

an integer containing the new number of references to the specified connection.

Definition at line 133 of file connections.c.

References `connection_t::lock`, `mutex_lock()`, `mutex_unlock()`, and `connection_t::refs`.

Referenced by `con_init()`.

connection_t* con_init (int cond, server_t * server)

Create a new connection object for a client connection.

Parameters:

cond the socket descriptor of the inbound client connection that was just accepted.

server the handle of the server that serviced the inbound request.

Definition at line 189 of file connections.c.

References `con_increment_refs()`, `connection_t::lock`, `mm_alloc()`, `mm_free()`, `mutex_init()`, `connection_t::network`, `connection_t::server`, and `connection_t::sockd`.

Referenced by `protocol_process()`.

bool_t con_init_network_buffer (connection_t * con)

Allocate and initialize a new network buffer for a connection, if it is not already associated with one.

Note:

A network buffer of size `magma.system.network_buffer` bytes will be allocated for the connection if one does not exist.

Returns:

true if the connection object has been associated with a network buffer or false on failure.

Definition at line 165 of file connections.c.

References `connection_t::buffer`, `connection_t::line`, `log_info`, `magma`, `connection_t::network`, `magma_t::network_buffer`, `pl_null()`, `st_alloc`, and `magma_t::system`.

Referenced by `con_read()`, and `con_read_line()`.

int_t con_secure (connection_t * con)

Return the security level of a specified connection.

Parameters:

con the input client connection.

Returns:

1 for a secure connection, 0 for insecure, and -1 if the server does not support SSL/TLS.

Definition at line 20 of file `connections.c`.

References `server_t::context`, `connection_t::network`, `connection_t::server`, `server_t::ssl`, and `connection_t::ssl`.

Referenced by `contact_process()`, `http_print_301()`, `http_response_cookie()`, `imap_capability()`, `imap_init()`, `imap_login()`, `imap_starttls()`, `pop_capa()`, `pop_pass()`, `pop_starttls()`, `portal_endpoint()`, `portal_endpoint_auth()`, `portal_process()`, `portal_upload()`, `protocol_enqueue()`, `register_process()`, `sess_create()`, `sess_get()`, `smtp_ehlo()`, `smtp_rcpt_to()`, `smtp_starttls()`, and `teacher_process()`.

int_t con_status (connection_t * con)

Return the status of a specified connection.

Parameters:

con the input client connection.

Returns:

-1 on error, 0 for unknown status, 1 for connected, or 2 if the socket has been shutting down.

Definition at line 37 of file `connections.c`.

References `connection_t::network`, `connection_t::sockd`, and `connection_t::status`.

Referenced by `http_requeue()`, `imap_fetch()`, `imap_logout()`, `imap_requeue()`, `imap_search()`, `pop_quit()`, `pop_requeue()`, `smtp_quit()`, and `smtp_requeue()`.

magma/network/http.h File Reference

The HTTP server control structures.

Data Structures

- struct **http_data_t**
- struct **http_content_t**
- struct **http_page_t**
- struct **__attribute__**

Enumerations

- enum **http_method_t** { **HTTP_METHOD_NONE**, **HTTP_METHOD_GET**, **HTTP_METHOD_POST**, **HTTP_METHOD_PUT**, **HTTP_METHOD_DELETE**, **HTTP_METHOD_HEAD**, **HTTP_METHOD_TRACE**, **HTTP_METHOD_OPTIONS**, **HTTP_METHOD_CONNECT**, **HTTP_METHOD_UNSUPPORTED** }
- enum **HTTP_DATA** { **HTTP_DATA_ANY**, **HTTP_DATA_HEADER**, **HTTP_DATA_GET**, **HTTP_DATA_POST** }
- enum { **HTTP_MERGED**, **HTTP_PORTAL** }

Detailed Description

The HTTP server control structures.

Definition in file **http.h**.

Enumeration Type Documentation

anonymous enum

Enumerator:

HTTP_MERGED
HTTP_PORTAL

Definition at line 36 of file http.h.

enum HTTP_DATA

Enumerator:

HTTP_DATA_ANY
HTTP_DATA_HEADER
HTTP_DATA_GET
HTTP_DATA_POST

Definition at line 29 of file http.h.

enum http_method_t

Enumerator:

HTTP_METHOD_NONE
HTTP_METHOD_GET
HTTP_METHOD_POST
HTTP_METHOD_PUT
HTTP_METHOD_DELETE
HTTP_METHOD_HEAD
HTTP_METHOD_TRACE
HTTP_METHOD_OPTIONS
HTTP_METHOD_CONNECT
HTTP_METHOD_UNSUPPORTED

Definition at line 16 of file http.h.

magma/servers/http/http.h File Reference

The entry point for the HTTP server module.

Enumerations

- enum { **HTTP_CONNECTION_CLOSE** = -1, **HTTP_CONNECTION_NEUTRAL** = 0, **HTTP_CONNECTION_KEEPALIVE** = 1 }
- enum { **HTTP_COOKIE_DELETE** = -1, **HTTP_COOKIE_NEUTRAL** = 0, **HTTP_COOKIE_SET** = 1 }
- enum { **HTTP_READY** = 0, **HTTP_PARSE_HEADER** = 1, **HTTP_PARSE_PAIRS** = 2, **HTTP_PARSE_COOKIE** = 3, **HTTP_READ_BODY** = 8, **HTTP_RESPOND** = 10, **HTTP_COMPLETE** = 12, **HTTP_ERROR_400** = 400, **HTTP_ERROR_401** = 401, **HTTP_ERROR_403** = 403, **HTTP_ERROR_404** = 404, **HTTP_ERROR_405** = 405, **HTTP_ERROR_500** = 500, **HTTP_ERROR_501** = 501, **HTTP_CLOSE** = 1000 }

Functions

- **bool_t http_content_load_directory** (int_t template, chr_t *directory)
- *content.c* **bool_t http_content_load_fonts** (void)
- *Load all fonts into the http server repository.* **bool_t http_content_refresh** (void)
- *Refresh the stored contents of the web app templates, static content, and fonts directories.* **bool_t http_content_start** (void)
- *Prime the the web app templates, static content, and fonts directories for future use.* void **http_content_stop** (void)
- *Purge the http contents repository of stored fonts, templates, and static web pages.* void **http_free_content** (**http_content_t** *page)
- *LOW: We should use basename() and dirname() to cleanup path strings.* **http_content_t** * **http_get_static** (**stringer_t** *location)
- *Get a cached copy of a static web page.* **http_content_t** * **http_get_template** (chr_t *location)
- *Return a template by location.* **bool_t http_load_file** (int_t template, chr_t *filename)
- *Load file content into the http server repository.* void **http_page_free** (**http_page_t** *page)
- *Free an http page object and all its underlying data.* **http_page_t** * **http_page_get** (chr_t *location)
- *Get a template page and prepare its xml document root for use.* void **http_data_free** (**http_data_t** *data)
- *data.c* **http_data_t** * **http_data_get** (**connection_t** *con, **HTTP_DATA** source, chr_t *name)
- *Get a name/value pair associated with an http connection, by name.* **http_data_t** * **http_data_header_parse_line** (chr_t *buf, size_t len)
- *Parse a data buffer into a http header name/value pair.* **http_data_t** * **http_data_header_parse** (**connection_t** *con)
- *Parse the current line of input from an http client connection into a http header name/value pair.* void **http_data_value_decode** (**stringer_t** *string)
- *Decode an escaped URI component into its original data.* **int_t http_data_value_parse** (**connection_t** *con, **HTTP_DATA** source, **placer_t** pair)
- *Parse a string containing a name/value pair, and place it in the specified connection's pairs holder.* void **http_print_301** (**connection_t** *con, chr_t *location, **int_t** ssl)
- *errors.c* void **http_print_400** (**connection_t** *con)
- *Return an HTTP 400 bad client request response to the client.* void **http_print_403** (**connection_t** *con)
- *Return an HTTP 403 access denied response to the client.* void **http_print_404** (**connection_t** *con)
- *Return an HTTP 404 location not found response to the client.* void **http_print_405** (**connection_t** *con)
- *Return an HTTP 405 method not allowed response to the client.* void **http_print_500** (**connection_t** *con)
- *Return an HTTP 500 internal server error response to the client.* void **http_print_500_log** (**connection_t** *con, chr_t *logmsg)

- Return an HTTP 500 internal server error response to the client, with additional logging information. void **http_print_501** (**connection_t** *con)
 - Return an HTTP 501 method not implemented response to the client. void **http_body** (**connection_t** *con)
 - *http.c* void **http_close** (**connection_t** *con)
 - Close a connection corresponding to an http session. void **http_init** (**connection_t** *con)
 - Handle a new http client connection. void **http_process** (**connection_t** *con)
 - Process data sent by an http client. void **http_requeue** (**connection_t** *con)
 - The main http server requeue entry point state machine for processing client data. void **http_parse_context** (**connection_t** *con, **stringer_t** *application, **stringer_t** *path)
 - Attempt to retrieve a connected user's associated session, by searching the cookie, the POST "session" variable, and the URL. void **http_parse_header** (**connection_t** *con)
 - Parse and process the current line of input from an http client connection as an http request header, storing data in the connection's **http.headers** member. void **http_parse_method** (**connection_t** *con)
 - Parse an http request and determine the request method and location. **int_t** **http_parse_origin** (**stringer_t** *s, **placer_t** *output)
 - Get the origin of a resource from a url. void **http_parse_pairs** (**connection_t** *con)
 - Parse all the GET and POST parameters present in an http client request, and store them with the connection. **placer_t** **get_header_value_noopt** (**stringer_t** *vstring)
 - Get the simple value of an http header, with any optional parameters stripped away. **placer_t** **get_header_opt** (**stringer_t** *vstring, **stringer_t** *optname)
 - Get the value of a named optional parameter from an http header value. **bool_t** **multipart_get_boundary** (**connection_t** *con, **placer_t** *output)
 - Get the boundary delimiter for a request by a connection specifying a Content-Type of multipart/form-data. void **http_response** (**connection_t** *con)
 - *response.c* **stringer_t** * **http_response_allow_cross** (**connection_t** *con)
 - **stringer_t** * **http_response_connection** (**connection_t** *con, **int_t** force)
 - Get an appropriate value for the Connection header of an http response. **stringer_t** * **http_response_cookie** (**connection_t** *con)
 - void **http_response_header** (**connection_t** *con, **int_t** status, **stringer_t** *type, size_t len)
 - Send a full set of http response headers to the remote client. void **http_response_options** (**connection_t** *con)
 - Return a response to an http OPTIONS request. **chr_t** * **http_response_status** (**int_t** status)
 - Get a descriptive string for a numerical http status code. void **http_session_destroy** (**connection_t** *con)
 - *sessions.c* void **http_session_reset** (**connection_t** *con)
- Reset a client http connection to its original, uninitialized state.
-

Detailed Description

The entry point for the HTTP server module.

Definition in file **http.h**.

Enumeration Type Documentation

anonymous enum

Enumerator:

HTTP_CONNECTION_CLOSE
HTTP_CONNECTION_NEUTRAL
HTTP_CONNECTION_KEEPALIVE

Definition at line 16 of file http.h.

anonymous enum

Enumerator:

HTTP_COOKIE_DELETE
HTTP_COOKIE_NEUTRAL
HTTP_COOKIE_SET

Definition at line 22 of file http.h.

anonymous enum

Enumerator:

HTTP_READY
HTTP_PARSE_HEADER
HTTP_PARSE_PAIRS
HTTP_PARSE_COOKIE
HTTP_READ_BODY
HTTP_RESPOND
HTTP_COMPLETE
HTTP_ERROR_400
HTTP_ERROR_401
HTTP_ERROR_403
HTTP_ERROR_404
HTTP_ERROR_405
HTTP_ERROR_500
HTTP_ERROR_501
HTTP_CLOSE

Definition at line 28 of file http.h.

Function Documentation

placer_t get_header_opt (stringer_t * vstring, stringer_t * optname)

Get the value of a named optional parameter from an http header value.

Note:

Only the value after the ":" in a header field is passed to this function (in other words, the header name is omitted).

Parameters:

vstring a managed string containing the single http header line value to be parsed.
optname a managed string with the name of the optional parameter to be found.

Returns:

a placer pointing to the named optional parameter of the specified http header value.

Definition at line 337 of file parse.c.

References `pl_null()`, `pl_trim_start()`, `placer_t`, `st_cmp_cs_eq()`, `tok_get_count_st()`, and `tok_get_st()`.
Referenced by `multipart_get_boundary()`, and `portal_upload()`.

`placer_t get_header_value_noopt (stringer_t * vstring)`

Get the simple value of an http header, with any optional parameters stripped away.

Note:

Only the value after the ":" in a header field is passed to this function (in other words, the header name is omitted).

Parameters:

vstring a managed string containing the single http header line value to be parsed.

Returns:

a placer pointing to the option-free header value of the specified input line.

Definition at line 314 of file `parse.c`.

References `pl_init()`, `placer_t`, `st_char_get()`, `st_length_get()`, `tok_get_count_st()`, and `tok_get_st()`.

Referenced by `http_parse_header()`, and `multipart_get_boundary()`.

`void http_body (connection_t * con)`

http.c

http.c

Note:

Any request errors will be handled directly without returns. This function sets the value of the connection's `http.body` member.

Parameters:

con a pointer to the connection object of the remote http client.

Returns:

This function returns no value.

Definition at line 78 of file `http.c`.

References `connection_t::buffer`, `con_read()`, `CONTIGUOUS`, `data`, `HEAP`, `magma_t::http`, `connection_t::http`, `http_data_get()`, `HTTP_DATA_HEADER`, `HTTP_ERROR_400`, `HTTP_RESPOND`, `JOINTED`, `length`, `magma_t::log`, `log_pedantic`, `magma`, `MANAGED_T`, `MAPPED_T`, `connection_t::network`, `size_conv_bl()`, `st_alloc_opts()`, `st_append_opts()`, `st_char_get()`, `st_data_get()`, `st_length_get()`, `st_length_int()`, and `http_data_t::value`.

Referenced by `http_requeue()`.

`void http_close (connection_t * con)`

Close a connection corresponding to an http session.

Returns:

This function returns no value.

Definition at line 19 of file `http.c`.

References `con_destroy()`.

Referenced by `http_process()`, and `http_requeue()`.

`bool_t http_content_load_directory (int_t template, chr_t * directory)`

content.c

content.c

See also:

`http_load_file()`

Parameters:

template a value specifying whether the specified directory contains templates or regular web content.

directory a null-terminated string specifying the pathname of the directory to be scanned recursively.

Returns:

0 on failure or 1 on success.

Definition at line 317 of file `content.c`.

References `http_content_load_directory()`, `http_load_file()`, `log_info`, `MEMORYBUF`, `NULLER`, `PLACER`, `st_char_get()`, `st_cmp_cs_ends()`, `st_cmp_cs_eq()`, `st_free()`, and `st_merge`.

Referenced by `http_content_load_directory()`, `http_content_refresh()`, and `http_content_start()`.

`bool_t http_content_load_fonts (void)`

Load all fonts into the http server repository.

Note:

This function will load all fonts of extension ".ttf" from the directory configured in **`magma.http.fonts`**.

Returns:

0 on failure or 1 on success.

Definition at line 366 of file `content.c`.

References `magma_t::content`, `content`, `magma_t::http`, `inx_insert()`, `magma_t::log`, `log_info`, `log_pedantic`, `M_TYPE_UINT64`, `magma`, `MEMORYBUF`, `NULLER`, `PLACER`, `st_cmp_ci_ends()`, `st_cmp_cs_ends()`, `st_free()`, `st_merge`, `multi_t::u64`, and `multi_t::val`.

Referenced by `http_content_refresh()`, and `http_content_start()`.

`bool_t http_content_refresh (void)`

Refresh the stored contents of the web app templates, static content, and fonts directories.

Returns:

true if all content directories were successfully refreshed, or false on failure.

Definition at line 453 of file `content.c`.

References `content`, `magma_t::http`, `http_content_load_directory()`, `http_content_load_fonts()`, `http_free_content()`, `inx_alloc()`, `inx_free()`, `M_INX_LINKED`, `magma`, and `st_free()`.

Referenced by `signal_refresh()`.

bool_t http_content_start (void)

Prime the the web app templates, static content, and fonts directories for future use.

Note:

This function makes sure that the web templates, static content, and fonts directory have been properly set, and loads and caches their content for future use.

Returns:

true on success or false on failure.

Definition at line 411 of file content.c.

References `content`, `magma_t::http`, `http_content_load_directory()`, `http_content_load_fonts()`, `http_free_content()`, `inx_alloc()`, `log_pedantic`, `M_INX_LINKED`, `magma`, `NULLER`, `PLACER`, `st_cmp_cs_ends()`, and `st_free()`.

Referenced by `process_start()`.

void http_content_stop (void)

Purge the http contents repository of stored fonts, templates, and static web pages.

Returns:

This function returns no value.

Definition at line 439 of file content.c.

References `content`, and `inx_cleanup()`.

Referenced by `process_stop()`.

void http_data_free (http_data_t * *data*)

data.c

data.c

Parameters:

data the http data object to be freed.

Returns:

This function returns no value.

Definition at line 20 of file data.c.

References `mm_free()`, `http_data_t::name`, `st_cleanup()`, and `http_data_t::value`.

Referenced by `http_data_value_parse()`, `http_parse_header()`, `http_parse_pairs()`, and `portal_upload()`.

http_data_t* http_data_get (connection_t * *con*, HTTP_DATA *source*, chr_t * *name*)

Get a name/value pair associated with an http connection, by name.

Note:

The name/value pairs searched can be supplied by a client through http request headers, or through GET or POST data.

Parameters:

con the connection object to be queried.

source the source of the client supplied value pair: HTTP_DATA_GET, HTTP_DATA_POST or HTTP_DATA_ANY.

name the name associated with the name/value pair to be retrieved.

Returns:

NULL on failure, or a pointer to the http name/value data pair requested on success.

TODO: We could just use the index find function.

Definition at line 39 of file data.c.

References `data`, `connection_t::http`, `HTTP_DATA_ANY`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `http_data_t::name`, `NULLER`, `http_data_t::source`, and `st_cmp_ci_eq()`.

Referenced by `contact_business()`, `contact_print_form()`, `contact_process()`, `http_body()`, `http_response_allow_cross()`, `multipart_get_boundary()`, `register_business_step1()`, `register_business_step2()`, `register_process()`, and `teacher_process()`.

http_data_t* http_data_header_parse (connection_t * con)

Parse the current line of input from an http client connection into a http header name/value pair.

Parameters:

con the connection object of the http client to be read.

Returns:

NULL on failure, or a pointer to an http header name/value pair on success.

Definition at line 231 of file data.c.

References `http_data_header_parse_line()`, `connection_t::line`, `connection_t::network`, `st_char_get()`, and `st_length_get()`.

Referenced by `http_parse_header()`.

http_data_t* http_data_header_parse_line (chr_t * buf, size_t len)

Parse a data buffer into a http header name/value pair.

Parameters:

con the connection object of the http client to be read.

Returns:

NULL on failure, or a pointer to an http header name/value pair on success.

Definition at line 168 of file data.c.

References `HTTP_DATA_HEADER`, `mm_alloc()`, `http_data_t::name`, `http_data_t::source`, `st_free()`, `st_import()`, and `http_data_t::value`.

Referenced by `http_data_header_parse()`, and `portal_upload()`.

void http_data_value_decode (stringer_t * *string*)

Decode an escaped URI component into its original data.

Note:

Since the un-escaped string will always be at least as small as the encoded value, the input managed string is transformed in place.

Parameters:

string a managed string containing the escaped URI data to be decoded.

Returns:

This function returns no value.

Definition at line 73 of file data.c.

References `hex_decode_chr()`, `length`, `st_char_get()`, `st_length_get()`, and `st_length_set()`.

Referenced by `http_data_value_parse()`.

int_t http_data_value_parse (connection_t * *con*, HTTP_DATA *source*, placer_t *pair*)

Parse a string containing a name/value pair, and place it in the specified connection's pairs holder.

Note:

Each name/value pair is assumed to be delimited with a '=' character, and each half is URI-decoded before storage.

Parameters:

con a pointer to the connection object of the http client submitting the user data to be parsed.

source an HTTP_DATA value specifying the source of the name/value pair (can be HTTP_DATA_HEADER, HTTP_DATA_GET, or HTTP_DATA_POST).

pair a placer pointing to a string containing the name/value pair to be parsed.

Returns:

0 on general or parsing failure, or 1 if the specified input buffer was successfully parsed and stored.

Definition at line 116 of file data.c.

References `CONTIGUOUS`, `data`, `HEAP`, `connection_t::http`, `magma_t::http`, `http_data_free()`, `http_data_value_decode()`, `inx_insert()`, `magma_t::log`, `log_pedantic`, `M_TYPE_STRINGER`, `magma`, `MANAGED_T`, `mm_alloc()`, `http_data_t::name`, `pl_empty()`, `placer_t`, `http_data_t::source`, `multi_t::st`, `st_char_get()`, `st_dupe_opts()`, `st_free()`, `st_length_int()`, `tok_get_pl()`, `multi_t::val`, and `http_data_t::value`.

Referenced by `http_parse_pairs()`.

void http_free_content (http_content_t * *page*)

LOW: We should use `basename()` and `dirname()` to cleanup path strings.

LOW: These functions should all be renamed to `http_content_XXXX`. The file and directory functions should be updated to use the `x64/reentrant` alternatives. Specifically `open64/fstat64/readdir64_r`. An update function that can be triggered with a `SIGHUP` would also be nice. Free a stored piece of http content and all its underlying data.

Parameters:

page a pointer to the loaded content page to be freed.

Returns:

This function returns no value.

Definition at line 32 of file content.c.

References `http_content_t::location`, `mm_free()`, `http_content_t::resource`, `st_cleanup()`, and `http_content_t::type`.

Referenced by `http_content_refresh()`, `http_content_start()`, and `http_load_file()`.

http_content_t* http_get_static (stringer_t * location)

Get a cached copy of a static web page.

Parameters:

location a pointer to a managed string containing the location of the requested resource.

Returns:

NULL on failure, or a pointer to an http content object with the contents of the requested resource on success.

Definition at line 72 of file content.c.

References `content`, `inx_find()`, `M_TYPE_STRINGER`, and `http_content_t::type`.

Referenced by `http_response()`.

http_content_t* http_get_template (chr_t * location)

Return a template by location.

Parameters:

location a string specifying the location of the template.

Returns:

NULL on failure, or a pointer to an **http_content_t** object containing the template.

Definition at line 88 of file content.c.

References `content`, `inx_find()`, `M_TYPE_STRINGER`, `NULLER`, and `http_content_t::type`.

Referenced by `http_page_get()`, `register_print_message()`, `register_print_step1()`, `register_print_step2()`, and `register_print_step3()`.

void http_init (connection_t * con)

Handle a new http client connection.

Parameters:

con a pointer to the http client connection that was just accepted.

Returns:

This function returns no value.

Definition at line 172 of file http.c.

References `con_reverse_enqueue()`, and `http_process()`.

Referenced by protocol_enqueue().

bool_t http_load_file (int_t *template*, chr_t * *filename*)

Load file content into the http server repository.

Note:

Each file that is loaded will be cached for retrieval by http clients, with its mime type determined automatically. Any files passed as a template will have the ".template" extension trimmed from the end of its resource path, and any file named 'index.html' will be read as the default directory index path. Each file will also be added to its respective parent page or template inx holder.

Parameters:

template a value specifying whether or not the specified file is a template.

filename a null-terminated string specifying the pathname of the file to be loaded.

Returns:

0 on failure or 1 on success.

Definition at line 154 of file content.c.

References magma_t::content, content, CONTIGUOUS, data, HEAP, magma_t::http, http_free_content(), inx_insert(), magma_t::log, log_info, log_pedantic, M_TYPE_STRINGER, magma, MANAGED_T, MEMORYBUF, mm_alloc(), ns_length_get(), NULLER, PLACER, multi_t::st, st_alloc, st_char_get(), st_cmp_ci_ends(), st_cmp_cs_ends(), st_cmp_cs_eq(), st_dupe(), st_dupe_opts(), st_free(), st_import(), st_length_get(), st_length_int(), st_length_set(), and multi_t::val.

Referenced by http_content_load_directory().

void http_page_free (http_page_t * *page*)

Free an http page object and all its underlying data.

Parameters:

page a pointer to the http page object to be freed.

Definition at line 48 of file content.c.

References http_page_t::doc_ctx, http_page_t::doc_obj, mm_free(), xml_free_doc(), xml_free_parser_ctx(), xml_free_xpath_ctx(), and http_page_t::xpath_ctx.

Referenced by contact_print_form(), contact_print_message(), http_page_get(), portal_print_login(), statistics_process(), teacher_print_form(), and teacher_print_message().

http_page_t* http_page_get (chr_t * *location*)

Get a template page and prepare its xml document root for use.

Note:

Each page is affixed with an xpath with a namespace after passing through the xml parser.

Parameters:

location a pointer to a null-terminated string with the pathname of the template to be returned.

Returns:

NULL on failure, or a pointer to the http page object of the requested template.

Definition at line 105 of file content.c.

References `http_page_t::content`, `http_page_t::doc_ctx`, `http_page_t::doc_obj`, `http_get_template()`, `http_page_free()`, `log_pedantic`, `mm_alloc()`, `http_content_t::resource`, `st_char_get()`, `st_length_get()`, `xml_create_doc()`, `xml_create_parser_ctx()`, `xml_create_xpath_ctx()`, `xml_xpath_set_namespace()`, and `http_page_t::xpath_ctx`.

Referenced by `contact_business_add_error()`, `contact_print_form()`, `contact_print_message()`, `portal_print_login()`, `statistics_process()`, `teacher_add_error()`, `teacher_print_form()`, and `teacher_print_message()`.

void http_parse_context (connection_t * *con*, stringer_t * *application*, stringer_t * *path*)

Attempt to retrieve a connected user's associated session, by searching the cookie, the POST "session" variable, and the URL.

TODO: The header and body parsers need limits on how many headers/bytes they will accept. `parse.c`

Parameters:

con a pointer to the connection object to be queried for a session id.

application a managed string containing the application associated with the connection's pending request.

path a managed string containing the path associated with the connection's pending request.

Returns:

This function returns no value.

TODO: Develop better logic for handling cookies. Namely add the ability to forcibly trigger the Set-Cookie entity and if necessary delete the existing cookie.

Definition at line 120 of file `parse.c`.

References `connection_t::http`, `HTTP_METHOD_POST`, `json_decref_d`, `json_loads_d`, `json_object_get_d`, `json_string_value_d`, `NULLER`, `pl_init()`, `placer_t`, `sess_get()`, `st_char_get()`, `st_cmp_ci_starts()`, `st_length_get()`, `tok_get_pl()`, and `tok_get_st()`.

Referenced by `portal_endpoint()`, `portal_process()`, and `portal_upload()`.

void http_parse_header (connection_t * *con*)

Parse and process the current line of input from an http client connection as an http request header, storing data in the connection's `http.headers` member.

Note:

If no more http headers can be read, control is returned by setting the connection `http.mode` to `HTTP_RESPOND`. Special actions are taken to store the "Host", "User-Agent", "Cookie", and "Connection" headers.

Parameters:

con the connection object of the http client to be read, and to store the results of the operation.

Returns:

This function returns no value.

LOW: Should we bother to throw an error if `con->http.connection` isn't `HTTP_CONNECTION_NEUTRAL`?

Definition at line 178 of file parse.c.

References `con_print()`, `CONTIGUOUS`, `data`, `get_header_value_noopt()`, `HEAP`, `magma_t::http`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `HTTP_CONNECTION_KEEPALIVE`, `http_data_free()`, `http_data_header_parse()`, `HTTP_ERROR_500`, `HTTP_RESPOND`, `int32_conv_bl()`, `inx_alloc()`, `inx_insert()`, `magma_t::log`, `log_info`, `lower_st()`, `M_INX_LINKED`, `M_TYPE_STRINGER`, `magma`, `MANAGED_T`, `http_data_t::name`, `PLACER`, `placer_t`, `multi_t::st`, `st_char_get()`, `st_cmp_ci_eq()`, `st_dupe_opts()`, `st_length_get()`, `st_length_int()`, `st_length_set()`, `st_search_cs()`, `multi_t::val`, and `http_data_t::value`.

Referenced by `http_process()`.

void http_parse_method (connection_t * con)

Parse an http request and determine the request method and location.

Note:

This function returns no value but sets the internal method, location, and state of the underlying connection object.

Parameters:

con the client connection making an http request.

Returns:

This function returns no value.

Definition at line 270 of file parse.c.

References `CONTIGUOUS`, `HEAP`, `magma_t::http`, `connection_t::http`, `HTTP_METHOD_CONNECT`, `HTTP_METHOD_DELETE`, `HTTP_METHOD_GET`, `HTTP_METHOD_HEAD`, `HTTP_METHOD_OPTIONS`, `HTTP_METHOD_POST`, `HTTP_METHOD_PUT`, `HTTP_METHOD_TRACE`, `HTTP_METHOD_UNSUPPORTED`, `HTTP_PARSE_HEADER`, `connection_t::line`, `magma_t::log`, `log_info`, `magma`, `MANAGED_T`, `connection_t::network`, `PLACER`, `placer_t`, `st_char_get()`, `st_cmp_ci_starts()`, `st_dupe_opts()`, `st_length_int()`, `tok_get_count_st()`, and `tok_get_pl()`.

Referenced by `http_process()`.

int_t http_parse_origin (stringer_t * s, placer_t * output)

Get the origin of a resource from a url.

Note:

Usually we'll be give the value of the Origin header field to work with so we'll end up returning the entire string, but if we had to fall back and use the Referrer instead this should strip off the path portion.

Parameters:

s a managed string containing the url to be parsed.

output a pointer to a placer that will be set to point to the origin string inside the user-supplied url.

Returns:

0 on success or -1 on failure.

Definition at line 23 of file parse.c.

References `pl_init()`, `PLACER`, `st_cmp_ci_starts()`, `st_data_get()`, `st_empty_out()`, and `st_uchar_get()`.

Referenced by `http_response_allow_cross()`.

void http_parse_pairs (connection_t * con)

Parse all the GET and POST parameters present in an http client request, and store them with the connection.

Note:

All valid field parameters will be stored in the connection's http.pairs object.

Parameters:

con a pointer to the connection object generating the http requesting being processed.

Returns:

This function returns no value.

Definition at line 78 of file parse.c.

References count, data, connection_t::http, http_data_free(), HTTP_DATA_GET, HTTP_DATA_POST, http_data_value_parse(), HTTP_ERROR_500, inx_alloc(), M_INX_LINKED, pl_init(), PLACER, placer_t, st_char_get(), st_length_get(), st_search_cs(), tok_get_count_st(), and tok_get_st().

Referenced by http_requeue(), and http_response().

void http_print_301 (connection_t * con, chr_t * location, int_t ssl)

errors.c

errors.c

Note:

SSL downgrades are not possible; in order for an ssl upgrade, magma.web.ssl_redirect must first be set.

Parameters:

con the client's http connection handle.

location the url to which the client will be redirected.

ssl if 1, direct to https:// else direct to http:// address.

Returns:

This function returns no value.

LOW: This function should probably move to the response file and be updated to use the cookie/xss helper functions.

Definition at line 23 of file errors.c.

References con_print(), con_secure(), connection_t::http, HTTP_COMPLETE, http_print_403(), http_print_500(), log_pedantic, magma, magma_t::ssl_redirect, st_char_get(), st_cleanup(), st_dupe(), st_free(), st_length_int(), st_merge, and magma_t::web.

Referenced by contact_process(), portal_endpoint(), portal_process(), portal_upload(), register_process(), and teacher_process().

void http_print_400 (connection_t * con)

Return an HTTP 400 bad client request response to the client.

Parameters:

con the client's http connection handle.

Returns:

This function returns no value.

Definition at line 84 of file errors.c.

References `con_write_st()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, and `PLACER`.

Referenced by `http_requeue()`.

void http_print_403 (connection_t * con)

Return an HTTP 403 access denied response to the client.

Parameters:

con the client's http connection handle.

Returns:

This function returns no value.

Definition at line 98 of file errors.c.

References `con_write_st()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, and `PLACER`.

Referenced by `http_print_301()`, and `http_requeue()`.

void http_print_404 (connection_t * con)

Return an HTTP 404 location not found response to the client.

Parameters:

con the client's http connection handle.

Returns:

This function returns no value.

Definition at line 112 of file errors.c.

References `con_write_st()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, and `PLACER`.

Referenced by `http_requeue()`.

void http_print_405 (connection_t * con)

Return an HTTP 405 method not allowed response to the client.

Parameters:

con the client's http connection handle.

Returns:

This function returns no value.

Definition at line 126 of file errors.c.

References `con_write_st()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, and `PLACER`.

Referenced by `http_requeue()`.

void http_print_500 (connection_t * con)

Return an HTTP 500 internal server error response to the client.

Parameters:

con the client's http connection handle.

Returns:

This function returns no value.

Definition at line 140 of file `errors.c`.

References `con_write_st()`, `debug_hook()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, `log_options`, `M_LOG_CRITICAL`, `M_LOG_STACK_TRACE`, and `PLACER`.

Referenced by `contact_print_message()`, `http_print_301()`, `http_requeue()`, `portal_print_login()`, `register_print_captcha()`, `register_print_message()`, `register_print_step3()`, `statistics_process()`, `teacher_print_form()`, and `teacher_print_message()`.

void http_print_500_log (connection_t * con, chr_t * logmsg)

Return an HTTP 500 internal server error response to the client, with additional logging information.

Parameters:

con the client's http connection handle.

logmsg a pointer to a null-terminated string with a message describing the cause of the http 500 error.

Returns:

This function returns no value.

Definition at line 159 of file `errors.c`.

References `con_write_st()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, `log_options`, `M_LOG_CRITICAL`, and `PLACER`.

Referenced by `contact_print_message()`, `register_print_step1()`, and `register_print_step2()`.

void http_print_501 (connection_t * con)

Return an HTTP 501 method not implemented response to the client.

Parameters:

con the client's http connection handle.

Returns:

This function returns no value.

Definition at line 174 of file `errors.c`.

References `con_write_st()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, and `PLACER`.

Referenced by `http_requeue()`.

void http_process (connection_t * con)

Process data sent by an http client.

Note:

This is performed in two stages: first read the http method with **http_parse_method()**, then transfer control to **http_parse_header()**. If a failure occurs reading a line of data, or too many protocol violations occur, **http_close()** is called to drop the connection.

Parameters:

con the connection object from which to read data.

Returns:

This function returns no value.

Definition at line 131 of file http.c.

References *con_read_line()*, *server_t::cutoff*, *enqueue()*, *connection_t::http*, *http_close()*, *http_parse_header()*, *HTTP_PARSE_HEADER*, *http_parse_method()*, *http_process()*, *HTTP_READY*, *http_requeue()*, *connection_t::line*, *log_pedantic*, *connection_t::network*, *pl_empty()*, *connection_t::protocol*, *requeue()*, *connection_t::server*, *connection_t::spins*, *server_t::violations*, and *connection_t::violations*.

Referenced by *http_init()*, *http_process()*, and *http_requeue()*.

void http_requeue (connection_t * con)

The main http server requeue entry point state machine for processing client data.

Returns:

This function returns no value.

Definition at line 29 of file http.c.

References *con_status()*, *server_t::cutoff*, *enqueue()*, *connection_t::http*, *http_body()*, *http_close()*, *HTTP_CLOSE*, *HTTP_COMPLETE*, *HTTP_ERROR_400*, *HTTP_ERROR_403*, *HTTP_ERROR_404*, *HTTP_ERROR_405*, *HTTP_ERROR_500*, *HTTP_ERROR_501*, *http_parse_pairs()*, *HTTP_PARSE_PAIRS*, *http_print_400()*, *http_print_403()*, *http_print_404()*, *http_print_405()*, *http_print_500()*, *http_print_501()*, *http_process()*, *HTTP_READ_BODY*, *http_requeue()*, *HTTP_RESPOND*, *http_response()*, *http_session_reset()*, *connection_t::protocol*, *requeue()*, *connection_t::server*, *status*, *server_t::violations*, and *connection_t::violations*.

Referenced by *http_process()*, and *http_requeue()*.

void http_response (connection_t * con)

response.c

response.c

Note:

The following http methods aren't supported: PUT, DELETE, HEAD, TRACE, and CONNECT. The http server will first attempt to retrieve the requested url as a static page; otherwise the following special locations are supported: /portal, /portal/camel, /register, /contact, /report_abuse, /teacher, and /statistics.

Returns:

This function returns no value.

Definition at line 424 of file response.c.

References magma_t::abuse, magma_t::admin, con_write_st(), magma_t::contact, contact_process(), content, connection_t::http, HTTP_ERROR_403, HTTP_ERROR_404, HTTP_ERROR_405, HTTP_ERROR_500, HTTP_ERROR_501, http_get_static(), HTTP_METHOD_CONNECT, HTTP_METHOD_DELETE, HTTP_METHOD_HEAD, HTTP_METHOD_OPTIONS, HTTP_METHOD_POST, HTTP_METHOD_PUT, HTTP_METHOD_TRACE, HTTP_METHOD_UNSUPPORTED, http_parse_pairs(), HTTP_READ_BODY, HTTP_RESPOND, http_response_header(), http_response_options(), magma, NULLER, PLACER, portal_endpoint(), portal_process(), portal_upload(), register_process(), magma_t::registration, http_content_t::resource, st_cmp_ci_starts(), st_cmp_cs_eq(), st_cmp_cs_starts(), st_length_get(), magma_t::statistics, statistics_process(), teacher_process(), http_content_t::type, and magma_t::web.

Referenced by http_queue().

stringer_t* http_response_allow_cross (connection_t * con)

Definition at line 250 of file response.c.

References http_data_get(), HTTP_DATA_HEADER, http_parse_origin(), MANAGEDBUF, PLACER, placer_t, st_append, st_char_get(), st_cmp_ci_eq(), st_length_int(), st_quick(), upper_st(), and http_data_t::value.

Referenced by http_response_header(), and http_response_options().

stringer_t* http_response_connection (connection_t * con, int_t force)

Get an appropriate value for the Connection header of an http response.

Note:

If magma.http.close was set in the configuration options, the connection will be closed immediately.

Parameters:

con a pointer to the connection object of the outgoing response.

force either HTTP_CONNECTION_CLOSE, HTTP_CONNECTION_NEUTRAL, or HTTP_CONNECTION_KEEPALIVE. HTTP_CONNECTION_CLOSE will result in the connection being terminated immediately, HTTP_CONNECTION_KEEPALIVE will result in a "Connection: keep-alive" response, and HTTP_CONNECTION_NEUTRAL will not result in any output.

Returns:

a pointer to a managed string containing the http Connection header field that should be used for the response.

Definition at line 293 of file response.c.

References CONTIGUOUS, HEAP, connection_t::http, magma_t::http, HTTP_CLOSE, HTTP_CONNECTION_CLOSE, HTTP_CONNECTION_KEEPALIVE, magma, MANAGED_T, PLACER, and st_dupe_opts().

Referenced by http_response_header(), and http_response_options().

stringer_t* http_response_cookie (connection_t * con)

Definition at line 195 of file response.c.

References con_secure(), HEAP, connection_t::http, HTTP_COOKIE_DELETE, HTTP_COOKIE_SET, JOINED, MANAGED_T, MANAGEDBUF, PLACER, st_append, st_aprint_opts(), st_char_get(), st_length_int(), st_quick(), and time_print_gmt().

Referenced by http_response_header().

void http_response_header (connection_t * con, int_t status, stringer_t * type, size_t len)

Send a full set of http response headers to the remote client.

Note:

If the mode is HTTP_RESPOND it will be changed to HTTP_COMPLETE, to tell the http requeue function to reset the context and enqueue request processor.

Parameters:

con a pointer to the connection object across which the response will be sent.

status an integer containing the http status code for the response.

type a managed string containing the value of the Content-Type header.

len the value of the Content-Length header.

Returns:

This function returns no value.

Definition at line 369 of file response.c.

References con_print(), magma_t::http, connection_t::http, HTTP_COMPLETE, HTTP_CONNECTION_NEUTRAL, HTTP_RESPOND, http_response_allow_cross(), http_response_connection(), http_response_cookie(), http_response_status(), magma, MANAGEDBUF, mm_wipe(), st_char_get(), st_cleanup(), st_length_int(), and time_print_gmt().

Referenced by contact_print_form(), contact_print_message(), http_print_400(), http_print_403(), http_print_404(), http_print_405(), http_print_500(), http_print_500_log(), http_print_501(), http_response(), portal_endpoint(), portal_endpoint_attachments_progress(), portal_endpoint_error(), portal_endpoint_response(), portal_endpoint_search(), portal_print_login(), register_print_message(), register_print_step1(), register_print_step2(), register_print_step3(), statistics_process(), and teacher_print_form().

void http_response_options (connection_t * con)

Return a response to an http OPTIONS request.

Parameters:

con the client connection which made the OPTIONS request.

Returns:

This function returns no value.

LOW: I couldn't find a definitive answer about whether the OPTION response should always return a Content-Type of text/plain or use the Content-Type associated with the location provided in the request.

Definition at line 321 of file response.c.

References build_stamp(), build_version(), con_print(), magma_t::http, connection_t::http, HTTP_COMPLETE, HTTP_CONNECTION_NEUTRAL, HTTP_RESPOND, http_response_allow_cross(), http_response_connection(), magma, MANAGEDBUF, mm_wipe(), st_char_get(), st_cleanup(), st_length_int(), and time_print_gmt().

Referenced by http_response().

chr_t* http_response_status (int_t status)

Get a descriptive string for a numerical http status code.

Parameters:

status the value of the http status code to be looked up.

Returns:

NULL on failure, or a pointer to a null-terminated string containing a description of the specified http status code on success.

Definition at line 20 of file response.c.

References log_pedantic.

Referenced by http_response_header().

void http_session_destroy (connection_t * con)

sessions.c

sessions.c

Parameters:

con the connection object to have its http session destroyed.

Returns:

This function returns no value.

Definition at line 82 of file sessions.c.

References http_session_reset().

Referenced by con_destroy().

void http_session_reset (connection_t * con)

Reset a client http connection to its original, uninitialized state.

Parameters:

con the connection object to have its http session state reset.

Returns:

This function returns no value.

Definition at line 20 of file sessions.c.

References connection_t::http, HTTP_CLOSE, HTTP_CONNECTION_CLOSE, HTTP_CONNECTION_NEUTRAL, HTTP_MERGED, HTTP_METHOD_NONE, HTTP_PORTAL, HTTP_READY, inx_cleanup(), json_decref_d, sess_release(), and st_cleanup().

Referenced by http_requeue(), and http_session_destroy().

bool_t multipart_get_boundary (connection_t * con, placer_t * output)

Get the boundary delimiter for a request by a connection specifying a Content-Type of multipart/form-data.

Parameters:

con a pointer to the connection object of the client making the http request.

output a pointer to the address of a managed string that will receive a copy of the value of the boundary string on success.

Returns:

true on success or false on failure.

Definition at line 374 of file parse.c.

References `get_header_opt()`, `get_header_value_noopt()`, `http_data_get()`, `HTTP_DATA_HEADER`, `NULLER`, `pl_empty()`, `placer_t`, and `http_data_t::value`.

Referenced by `portal_upload()`.

magma/network/imap.h File Reference

The IMAP server control structures.

Data Structures

- struct **imap_folder_status_t**
- struct **imap_fetch_dataitems_t**
- struct **imap_fetch_response_t**
- struct **__attribute__**

Typedefs

- typedef **array_t** **imap_arguments_t**

Detailed Description

The IMAP server control structures.

Definition in file **imap.h**.

Typedef Documentation

typedef array_t **imap_arguments_t**

Definition at line 16 of file **imap.h**.

magma/servers/imap/imap.h File Reference

The entry point for the IMAP server module.

Defines

- `#define IMAP_ARRAY_RECURSION_LIMIT 16`
- `#define IMAP_SEARCH_RECURSION_LIMIT 16`
- `#define IMAP_FOLDER_RECURSION_LMIIT 16`
- `#define IMAP_ARGUMENT_TYPE_EMPTY 0`
- `#define IMAP_ARGUMENT_TYPE_ARRAY 1`
- `#define IMAP_ARGUMENT_TYPE_ASTRING 2`
- `#define IMAP_ARGUMENT_TYPE_QSTRING 3`
- `#define IMAP_ARGUMENT_TYPE_NSTRING 4`
- `#define IMAP_ARGUMENT_TYPE_LITERAL 5`
- `#define IMAP_FETCH_BODY_HEADER_FIELDS_NOT 1`
- `#define IMAP_FETCH_BODY_HEADER_FIELDS 2`
- `#define IMAP_FETCH_BODY_HEADER 3`
- `#define IMAP_FETCH_BODY_TEXT 4`
- `#define IMAP_FETCH_BODY_MIME 5`
- `#define IMAP_FETCH_BODY_PART 6`
- `#define IMAP_FLAG_SILENT 1`
- `#define IMAP_FLAG_ADD 2`
- `#define IMAP_FLAG_REMOVE 4`
- `#define IMAP_FLAG_REPLACE 8`

Functions

- `int_t imap_compare (const void *compare, const void *command)`
- *commands.c* void `imap_process (connection_t *con)`
- *Perform client command processing on an established imap session.* void `imap_requeue (connection_t *con)`
- *Requeue an imap connection for processing, or log it out if there was an error or excess of protocol violations.* void `imap_sort (void)`
- *Sort the IMAP command table to be ready for binary searches.* `imap_fetch_response_t *`
`imap_fetch_response_add (imap_fetch_response_t *response, stringer_t *key, stringer_t *value)`
- *fetch_response.c* void `imap_fetch_response_free (imap_fetch_response_t *response)`
- `inx_t * imap_duplicate_messages (inx_t *messages)`
- *fetch.c* `imap_fetch_response_t * imap_fetch_body (array_t *outer, array_t *partial, connection_t *con, meta_message_t *meta, mail_message_t **message, stringer_t **header, imap_fetch_response_t *output)`
- `stringer_t * imap_fetch_body_header (placer_t header, imap_arguments_t *array, int_t not)`
- `stringer_t * imap_fetch_body_mime (placer_t header)`
- `mail_mime_t * imap_fetch_body_part (mail_message_t *message, placer_t portion)`
- `placer_t imap_fetch_body_portion (stringer_t *part)`
- `stringer_t * imap_fetch_body_tag (stringer_t *tag, array_t *items)`
- `stringer_t * imap_fetch_bodystructure (mail_mime_t *mime)`
- `stringer_t * imap_fetch_envelope (stringer_t *header)`
- void `imap_fetch_free_items (imap_fetch_dataitems_t *items)`
- `imap_fetch_response_t * imap_fetch_message (connection_t *con, meta_message_t *meta, imap_fetch_dataitems_t *items)`
- `int_t imap_fetch_parse_partial (stringer_t *partial, size_t *start, size_t *length)`
- `stringer_t * imap_fetch_return_header (connection_t *con, meta_message_t *meta, mail_message_t **message, stringer_t **header, imap_fetch_response_t *output)`

- **mail_message_t * imap_fetch_return_message** (**connection_t** *con, **meta_message_t** *meta, **mail_message_t** **message, **stringer_t** **header, **imap_fetch_response_t** *output)
- **mail_mime_t * imap_fetch_return_mime** (**connection_t** *con, **meta_message_t** *meta, **mail_message_t** **message, **stringer_t** **header, **imap_fetch_response_t** *output)
- **stringer_t * imap_fetch_return_text** (**connection_t** *con, **meta_message_t** *meta, **mail_message_t** **message, **stringer_t** **header, **imap_fetch_response_t** *output)
- **inx_t * imap_narrow_messages** (**inx_t** *messages, uint64_t selected, **stringer_t** *range, **int_t** uid)
- **imap_fetch_dataitems_t * imap_parse_dataitems** (**imap_arguments_t** *arguments)
- **int_t imap_valid_sequence** (**stringer_t** *range)
- **int_t imap_flag_action** (**stringer_t** *string)
- *flags.c* **uint32_t imap_flag_parse** (void *ptr, **int_t** type)
- **uint32_t imap_get_flag** (**stringer_t** *string)
- **void imap_update_flags** (**meta_user_t** *user, **inx_t** *messages, uint64_t foldernum, **int_t** action, **uint32_t** flags)
- **int_t imap_count_folder_levels** (**stringer_t** *name)
- *folders.c* **int_t imap_folder_compare** (**stringer_t** *name, **stringer_t** *compare)
- **int_t imap_folder_create** (uint64_t usernum, **inx_t** *folders, **stringer_t** *name)
- *Create a new imap folder.* **stringer_t * imap_folder_name_escaped** (**inx_t** *folders, **meta_folder_t** *active)
- *Get an imap folder's escaped name.* **int_t imap_folder_remove** (uint64_t usernum, **inx_t** *folders, **inx_t** *messages, **stringer_t** *name)
- *Remove an imap folder, both in memory and on the database.* **int_t imap_folder_rename** (uint64_t usernum, **inx_t** *folders, **stringer_t** *original, **stringer_t** *rename)
- *Rename an imap folder.* **int_t imap_folder_status** (**inx_t** *folders, **inx_t** *messages, **stringer_t** *name, **imap_folder_status_t** *status)
- *Get the status of a folder.* **inx_t * imap_narrow_folders** (**inx_t** *folders, **stringer_t** *reference, **stringer_t** *mailbox)
- **uint64_t imap_next_folder_order** (**inx_t** *folders, uint64_t parent)
- *Get the next order value for an imap folder.* **bool_t imap_valid_folder_name** (**stringer_t** *name)
- *Check to see if a name is a valid imap folder name.* **void imap_append** (**connection_t** *con)
- *imap.c* **void imap_capability** (**connection_t** *con)
- *Display the capability string for the IMAP server.* **void imap_check** (**connection_t** *con)
- **void imap_close** (**connection_t** *con)
- **void imap_copy** (**connection_t** *con)
- **void imap_create** (**connection_t** *con)
- *Create a new imap folder in response to the imap "CREATE" command.* **void imap_delete** (**connection_t** *con)
- *Handle the imap "DELETE" command and delete the specified imap folder.* **void imap_examine** (**connection_t** *con)
- **void imap_expunge** (**connection_t** *con)
- **void imap_fetch** (**connection_t** *con)
- **void imap_id** (**connection_t** *con)
- **void imap_idle** (**connection_t** *con)
- **void imap_init** (**connection_t** *con)
- *The main imap entry point for all inbound client connections, as dispatched by the generic protocol handler (display banner greeting).* **void imap_invalid** (**connection_t** *con)
- *Respond to an invalid imap command from a client.* **void imap_list** (**connection_t** *con)
- **void imap_login** (**connection_t** *con)
- *Attempt to perform a user login on an imap client connection.* **void imap_logout** (**connection_t** *con)
- *Terminate an IMAP session gracefully with a BYE message and destroy the underlying connection.* **void imap_lsub** (**connection_t** *con)
- **void imap_noop** (**connection_t** *con)
- **void imap_rename** (**connection_t** *con)
- **void imap_search** (**connection_t** *con)
- **void imap_select** (**connection_t** *con)

- void **imap_starttls** (**connection_t** *con)
- *Create a secure connection for an IMAP session.* void **imap_status** (**connection_t** *con)
- void **imap_store** (**connection_t** *con)
- void **imap_subscribe** (**connection_t** *con)
- void **imap_unsubscribe** (**connection_t** *con)
- **int_t** **imap_append_message** (**connection_t** *con, **meta_folder_t** *folder, **uint32_t** flags, **stringer_t** *message, **uint64_t** *outnum)
- *messages.c* **int_t** **imap_message_copier** (**connection_t** *con, **meta_message_t** *message, **uint64_t** target, **uint64_t** *outnum)
- **int_t** **imap_message_expunge** (**connection_t** *con, **meta_message_t** *message)
- **stringer_t** * **imap_build_array** (**chr_t** *format,...)
- *output.c* **stringer_t** * **imap_build_array_isliteral** (**placer_t** data)
- **stringer_t** * **imap_parse_address** (**stringer_t** *address)
- *parse_address.c* **placer_t** **imap_parse_address_breaker** (**stringer_t** *address, **uint32_t** part)
- **stringer_t** * **imap_parse_address_part** (**placer_t** input)
- void **imap_parse_address_put** (**stringer_t** *buffer, **chr_t** c)
- *LOW: Do we need an imap_parse_address_group() function?* **int_t** **imap_command_parser** (**connection_t** *con)
- *parse.c* **imap_arguments_t** * **imap_get_ar_ar** (**imap_arguments_t** *array, **size_t** element)
- *Return the value of a specified object in an array as an imap arguments array.* void * **imap_get_ptr** (**imap_arguments_t** *array, **size_t** element)
- *Return the value of a specified object in an array as a generic pointer.* **stringer_t** * **imap_get_st_ar** (**imap_arguments_t** *array, **size_t** element)
- *Return the value of a specified object in an array as a managed string.* **int_t** **imap_get_type_ar** (**imap_arguments_t** *array, **size_t** element)
- *TODO: The logic here is just plain ugly. Specifically the logic used to read from the network and advance the buffers.* **int_t** **imap_parse_arguments** (**connection_t** *con, **chr_t** **start, **size_t** *length)
- *Parse a chunk of input into an array of IMAP arguments.* **int_t** **imap_parse_array** (**int_t** recursion, **connection_t** *con, **imap_arguments_t** **array, **chr_t** **start, **size_t** *length)
- *Extract the contents of an array argument and advance the position of the parser stream.* **int_t** **imap_parse_istring** (**stringer_t** **output, **chr_t** **start, **size_t** *length)
- *Extract the contents of an atomic string and advance the position of the parser stream.* **int_t** **imap_parse_literal** (**connection_t** *con, **stringer_t** **output, **chr_t** **start, **size_t** *length)
- *Extract the contents of a literal string and advance the position of the parser stream.* **int_t** **imap_parse_nstring** (**stringer_t** **output, **chr_t** **start, **size_t** *length, **chr_t** type)
- **int_t** **imap_parse_qstring** (**stringer_t** **output, **chr_t** **start, **size_t** *length)
- *Extract the contents of a quoted string and advance the position of the parser stream.* **stringer_t** * **imap_range_build** (**size_t** length, **uint64_t** *numbers)
- *range.c* **int_t** **imap_search_flag** (**uint32_t** status, **uint32_t** flag, **int_t** has)
- *search.c* **inx_t** * **imap_search_messages** (**connection_t** *con)
- **int_t** **imap_search_messages_body** (**meta_user_t** *user, **mail_message_t** **data, **meta_message_t** *active, **stringer_t** *value)
- **int_t** **imap_search_messages_date** (**meta_user_t** *user, **mail_message_t** **data, **stringer_t** **header, **meta_message_t** *active, **stringer_t** *date, **int_t** internal, **int_t** expected)
- **int_t** **imap_search_messages_date_compare** (**stringer_t** *one, **stringer_t** *two)
- **int_t** **imap_search_messages_header** (**meta_user_t** *user, **mail_message_t** **data, **stringer_t** **header, **meta_message_t** *active, **stringer_t** *field, **stringer_t** *value)
- **int_t** **imap_search_messages_inner** (**meta_user_t** *user, **mail_message_t** **message, **stringer_t** **header, **meta_message_t** *current, **imap_arguments_t** *array, unsigned recursion)
- **int_t** **imap_search_messages_range** (**meta_message_t** *active, **stringer_t** *range, **int_t** uid)
- **int_t** **imap_search_messages_size** (**meta_message_t** *active, **stringer_t** *value, **int_t** expected)
- **int_t** **imap_search_messages_text** (**meta_user_t** *user, **mail_message_t** **data, **meta_message_t** *active, **stringer_t** *value)
- void **imap_session_destroy** (**connection_t** *con)

- *sessions.c* **int_t** **imap_session_update** (**connection_t** *con)

Detailed Description

The entry point for the IMAP server module.

Definition in file **imap.h**.

Define Documentation

#define IMAP_ARGUMENT_TYPE_ARRAY 1

Definition at line 22 of file *imap.h*.

Referenced by *imap_append()*, *imap_copy()*, *imap_create()*, *imap_delete()*, *imap_examine()*, *imap_fetch()*, *imap_fetch_body()*, *imap_fetch_body_header()*, *imap_flag_parse()*, *imap_get_ar_ar()*, *imap_get_st_ar()*, *imap_id()*, *imap_list()*, *imap_login()*, *imap_lsub()*, *imap_parse_arguments()*, *imap_parse_array()*, *imap_parse_dataitems()*, *imap_rename()*, *imap_search_messages_inner()*, *imap_select()*, *imap_status()*, *imap_store()*, and *imap_subscribe()*.

#define IMAP_ARGUMENT_TYPE_ASTRING 2

Definition at line 23 of file *imap.h*.

Referenced by *imap_parse_arguments()*.

#define IMAP_ARGUMENT_TYPE_EMPTY 0

Definition at line 21 of file *imap.h*.

Referenced by *imap_get_type_ar()*.

#define IMAP_ARGUMENT_TYPE_LITERAL 5

Definition at line 26 of file *imap.h*.

Referenced by *imap_append()*, *imap_parse_arguments()*, and *imap_parse_array()*.

#define IMAP_ARGUMENT_TYPE_NSTRING 4

Definition at line 25 of file *imap.h*.

Referenced by *imap_parse_array()*.

#define IMAP_ARGUMENT_TYPE_QSTRING 3

Definition at line 24 of file *imap.h*.

Referenced by *imap_parse_arguments()*, and *imap_parse_array()*.

#define IMAP_ARRAY_RECURSION_LIMIT 16

Definition at line 16 of file imap.h.

Referenced by imap_parse_array().

#define IMAP_FETCH_BODY_HEADER 3

Definition at line 31 of file imap.h.

#define IMAP_FETCH_BODY_HEADER_FIELDS 2

Definition at line 30 of file imap.h.

#define IMAP_FETCH_BODY_HEADER_FIELDS_NOT 1

Definition at line 29 of file imap.h.

#define IMAP_FETCH_BODY_MIME 5

Definition at line 33 of file imap.h.

#define IMAP_FETCH_BODY_PART 6

Definition at line 34 of file imap.h.

#define IMAP_FETCH_BODY_TEXT 4

Definition at line 32 of file imap.h.

#define IMAP_FLAG_ADD 2

Definition at line 38 of file imap.h.

Referenced by imap_flag_action(), and imap_update_flags().

#define IMAP_FLAG_REMOVE 4

Definition at line 39 of file imap.h.

Referenced by imap_flag_action(), and imap_update_flags().

#define IMAP_FLAG_REPLACE 8

Definition at line 40 of file imap.h.

Referenced by imap_flag_action(), and imap_update_flags().

#define IMAP_FLAG_SILENT 1

Definition at line 37 of file imap.h.

Referenced by `imap_flag_action()`, and `imap_store()`.

#define IMAP_FOLDER_RECURSION_LMIIT 16

Definition at line 18 of file imap.h.

Referenced by `contact_folder_create()`, `imap_count_folder_levels()`, `imap_create()`, `imap_folder_create()`, `imap_folder_rename()`, `imap_rename()`, and `imap_subscribe()`.

#define IMAP_SEARCH_RECURSION_LIMIT 16

Definition at line 17 of file imap.h.

Referenced by `imap_search_messages_inner()`.

Function Documentation

void imap_append (connection_t * con)

imap.c

BUG: If the user is over their quota check whether rollout is enabled and if so. If enabled make room for the appended message using the rollout logic.

Definition at line 1400 of file imap.c.

References `ar_length_get()`, `con_print()`, `meta_message_t::foldernum`, `meta_folder_t::foldernum`, `connection_t::imap`, `imap_append_message()`, `IMAP_ARGUMENT_TYPE_ARRAY`, `IMAP_ARGUMENT_TYPE_LITERAL`, `imap_flag_parse()`, `imap_get_ptr()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `log_pedantic`, `MAIL_STATUS_EMPTY`, `MAIL_STATUS_RECENT`, `meta_folders_by_name()`, `META_USER_OVERQUOTA`, `meta_user_unlock()`, `meta_user_wlock()`, `st_char_get()`, `st_length_int()`, and `meta_message_t::status`.

int_t imap_append_message (connection_t * con, meta_folder_t * folder, uint32_t flags, stringer_t * message, uint64_t * outnum)

messages.c

Definition at line 15 of file messages.c.

References `magma_t::active`, `meta_folder_t::foldernum`, `connection_t::imap`, `inx_insert()`, `log_pedantic`, `M_TYPE_UINT64`, `magma`, `MAIL_STATUS_APPENDED`, `MAIL_STATUS_RECENT`, `mail_store_message()`, `meta_messages_update_sequences()`, `META_USER_ENCRYPT_DATA`, `mm_alloc()`, `mm_free()`, `OBJECT_MESSAGES`, `serial_get()`, `serial_increment()`, `st_char_get()`, `st_length_get()`, `st_length_int()`, `magma_t::storage`, `multi_t::u64`, and `multi_t::val`.

Referenced by `imap_append()`.

stringer_t* imap_build_array (chr_t * *format*, ...)

output.c

Definition at line 42 of file output.c.

References `imap_build_array_isliteral()`, `length`, `log_error`, `log_pedantic`, `ns_length_get()`, `number`, `pl_data_get()`, `pl_empty()`, `pl_init()`, `pl_length_get()`, `placer_t`, `st_alloc`, `st_char_get()`, `st_free()`, `st_length_get()`, and `st_length_set()`.

Referenced by `imap_fetch_bodystructure()`, `imap_fetch_envelope()`, `imap_parse_address()`, and `imap_parse_address_part()`.

stringer_t* imap_build_array_isliteral (placer_t *data*)

Definition at line 15 of file output.c.

References `length`, `pl_data_get()`, `pl_empty()`, `pl_length_get()`, and `st_merge`.

Referenced by `imap_build_array()`, and `imap_fetch_bodystructure()`.

void imap_capability (connection_t * *con*)

Display the capability string for the IMAP server.

Note:

This string always needs to stay in sync with the banner greeting.

Returns:

This function returns no value.

Definition at line 1764 of file imap.c.

References `con_print()`, `con_secure()`, `connection_t::imap`, `st_char_get()`, and `st_length_int()`.

void imap_check (connection_t * *con*)

Definition at line 237 of file imap.c.

References `con_print()`, `connection_t::imap`, `imap_session_update()`, `st_char_get()`, and `st_length_int()`.

void imap_close (connection_t * *con*)

Definition at line 1025 of file imap.c.

References `ar_length_get()`, `con_print()`, `meta_message_t::foldernum`, `connection_t::imap`, `imap_message_expunge()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_DELETED`, `MAIL_STATUS_RECENT`, `meta_messages_update_sequences()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_MESSAGES`, `serial_get()`, `serial_increment()`, `st_char_get()`, `st_length_int()`, `meta_message_t::status`, `user_lock()`, and `user_unlock()`.

int_t imap_command_parser (connection_t * *con*)

`parse.c`

parse.c

Note:

This function updates the protocol-specific IMAP structure with parsed values for the session's tag, command, and arguments fields. Special handling is performed for any command that is preceded by a "UID" prefix.

Parameters:

con the IMAP client connection issuing the command.

Returns:

1 on success or < 0 on error. -1: the tag could not be read. -2: the IMAP command could not be read. -3: the arguments to the IMAP command could not be read.

Definition at line 890 of file parse.c.

References `ar_free()`, `magma_t::imap`, `connection_t::imap`, `imap_command_log_safe()`, `imap_parse_arguments()`, `imap_parse_astring()`, `length`, `connection_t::line`, `magma_t::log`, `log_pedantic`, `magma`, `connection_t::network`, `PLACER`, `st_cleanup()`, `st_cmp_ci_eq()`, `st_data_get()`, `st_free()`, and `st_length_get()`.

Referenced by `imap_process()`.

int_t imap_compare (const void * *compare*, const void * *command*)

commands.c

commands.c

Note:

This is an internal function used to sort imap commands and search for them.

Parameters:

compare a pointer to the first command to be compared.

command a pointer to the second command to be compared.

Returns:

-1 if `compare < command`, 1 if `command < compare`, or 0 if the two commands are equal.

Definition at line 23 of file commands.c.

References `command_t::length`, `PLACER`, `st_cmp_ci_eq()`, `st_cmp_ci_starts()`, and `command_t::string`.

Referenced by `imap_process()`, and `imap_sort()`.

void imap_copy (connection_t * *con*)

Definition at line 1234 of file imap.c.

References `ar_length_get()`, `con_print()`, `count`, `meta_message_t::foldernum`, `meta_folder_t::foldernum`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_get_st_ar()`, `imap_get_type_ar()`, `imap_message_copier()`, `imap_narrow_messages()`, `imap_range_build()`, `imap_valid_sequence()`, `inx_count()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_free()`, `log_pedantic`, `MAIL_STATUS_RECENT`, `meta_message_t::messagenum`, `meta_folders_by_name()`, `meta_user_unlock()`, `meta_user_wlock()`, `mm_alloc()`, `mm_free()`, `meta_message_t::sequencenum`, `st_char_get()`, `st_length_get()`, `st_length_int()`, `meta_message_t::status`, `user_lock()`, and `user_unlock()`.

int_t imap_count_folder_levels (stringer_t * *name*)

folders.c

folders.c

Note:

The smallest possible node level value is 1.

Parameters:

name a managed string containing the name of the imap folder.

Returns:

0 on failure, or the number of node levels indicated by the specified imap folder name, on success.

Definition at line 223 of file folders.c.

References IMAP_FOLDER_RECURSION_LMIIT, log_pedantic, and st_empty_out().

Referenced by imap_folder_create(), and imap_folder_rename().

void imap_create (connection_t * con)

Create a new imap folder in response to the imap "CREATE" command.

See also:

imap_folder_create()

Parameters:

con the connection across which the folder creation request was made.

Returns:

This function returns no value.

Definition at line 412 of file imap.c.

References ar_length_get(), con_print(), FOLDER_LENGTH_LIMIT, connection_t::imap, IMAP_ARGUMENT_TYPE_ARRAY, imap_folder_create(), IMAP_FOLDER_RECURSION_LMIIT, imap_get_st_ar(), imap_get_type_ar(), meta_user_unlock(), meta_user_wlock(), OBJECT_FOLDERS, serial_get(), serial_increment(), st_char_get(), and st_length_int().

void imap_delete (connection_t * con)

Handle the imap "DELETE" command and delete the specified imap folder.

See also:

imap_folder_remove()

Parameters:

con a pointer to the connection object of the imap session generating the delete request.

Returns:

This function returns no value.

Definition at line 476 of file imap.c.

References ar_length_get(), con_print(), connection_t::imap, IMAP_ARGUMENT_TYPE_ARRAY, imap_folder_remove(), imap_get_st_ar(), imap_get_type_ar(), meta_user_unlock(), meta_user_wlock(), OBJECT_FOLDERS, serial_get(), serial_increment(), st_char_get(), and st_length_int().

inx_t* imap_duplicate_messages (inx_t * messages)

fetch.c

fetch.c

See also:

meta_message_dupe()

Parameters:

messages a collection of meta message objects to be duplicated.

Returns:

the copied collection of meta messages on success, or NULL on failure.

Definition at line 1245 of file fetch.c.

References `inx_alloc()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_free()`, `inx_insert()`, `log_error`, `M_INX_LINKED`, `M_TYPE_UINT64`, `meta_message_t::messagenum`, `meta_message_dupe()`, `meta_message_free()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `imap_fetch()`, and `imap_store()`.

void imap_examine (connection_t * con)

Definition at line 740 of file imap.c.

References `ar_length_get()`, `con_print()`, `imap_folder_status_t::first`, `imap_folder_status_t::foldernum`, `meta_message_t::foldernum`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_folder_status()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_RECENT`, `imap_folder_status_t::messages`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_user_wlock()`, `imap_folder_status_t::recent`, `st_char_get()`, `st_length_int()`, `meta_message_t::status`, `status`, and `imap_folder_status_t::uidnext`.

void imap_expunge (connection_t * con)

Definition at line 1135 of file imap.c.

References `ar_length_get()`, `con_print()`, `meta_message_t::foldernum`, `connection_t::imap`, `imap_message_expunge()`, `imap_session_update()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_DELETED`, `meta_messages_update_sequences()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_MESSAGES`, `meta_message_t::sequencenum`, `serial_get()`, `serial_increment()`, `st_char_get()`, `st_length_int()`, `meta_message_t::status`, `user_lock()`, and `user_unlock()`.

void imap_fetch (connection_t * con)

Definition at line 1485 of file imap.c.

References `ar_length_get()`, `con_print()`, `con_status()`, `con_write_bl()`, `con_write_st()`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_duplicate_messages()`, `imap_fetch_free_items()`, `imap_fetch_message()`, `imap_fetch_response_free()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `imap_narrow_messages()`, `imap_parse_dataitems()`, `imap_valid_sequence()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_reset()`, `inx_cursor_value_next()`, `inx_free()`, `items`, `imap_fetch_response_t::key`, `MAIL_STATUS_SEEN`, `meta_data_flags_add()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_user_wlock()`, `imap_fetch_response_t::next`, `imap_fetch_dataitems_t::normal`, `OBJECT_MESSAGES`, `PLACER`, `imap_fetch_dataitems_t::rfc822`, `imap_fetch_dataitems_t::rfc822_header`, `imap_fetch_dataitems_t::rfc822_text`, `meta_message_t::sequencenum`, `serial_get()`, `serial_increment()`, `st_char_get()`, `st_cmp_cs_eq()`, `st_length_int()`, `status`, `meta_message_t::status`, `meta_message_t::updated`, and `imap_fetch_response_t::value`.

imap_fetch_response_t* imap_fetch_body (array_t * *outer*, array_t * *partial*, connection_t * *con*, meta_message_t * *meta*, mail_message_t ** *message*, stringer_t ** *header*, imap_fetch_response_t * *output*)

Definition at line 767 of file fetch.c.

References ar_field_ar(), ar_field_type(), ar_length_get(), ARRAY_TYPE_ARRAY, mail_mime_t::body, mail_mime_t::header, IMAP_ARGUMENT_TYPE_ARRAY, imap_fetch_body_header(), imap_fetch_body_mime(), imap_fetch_body_part(), imap_fetch_body_portion(), imap_fetch_body_tag(), imap_fetch_parse_partial(), imap_fetch_response_add(), imap_fetch_response_free(), imap_fetch_return_header(), imap_fetch_return_message(), imap_fetch_return_mime(), imap_fetch_return_text(), imap_get_ar_ar(), imap_get_ptr(), imap_get_st_ar(), imap_get_type_ar(), length, mail_destroy(), mail_destroy_header(), mail_message_t::mime, number, pl_data_get(), pl_empty(), pl_init(), pl_length_get(), pl_null(), PLACER, placer_t, st_char_get(), st_cleanup(), st_cmp_ci_eq(), st_free(), st_import(), st_length_get(), and st_merge.

Referenced by imap_fetch_message().

stringer_t* imap_fetch_body_header (placer_t *header*, imap_arguments_t * *array*, int_t *not*)

Definition at line 391 of file fetch.c.

References ar_length_get(), IMAP_ARGUMENT_TYPE_ARRAY, imap_get_st_ar(), imap_get_type_ar(), mail_header_pop(), mm_cmp_ci_eq(), number, pl_char_get(), pl_data_get(), pl_empty(), placer_t, st_char_get(), st_cleanup(), st_length_get(), and st_merge.

Referenced by imap_fetch_body().

stringer_t* imap_fetch_body_mime (placer_t *header*)

Definition at line 442 of file fetch.c.

References mail_header_fetch_all(), PLACER, st_cleanup(), and st_merge.

Referenced by imap_fetch_body().

mail_mime_t* imap_fetch_body_part (mail_message_t * *message*, placer_t *portion*)

Definition at line 560 of file fetch.c.

References ar_field_ptr(), mail_mime_t::children, mail_message_t::mime, number, placer_t, tok_get_count_st(), tok_get_st(), and uint32_conv_st().

Referenced by imap_fetch_body().

placer_t imap_fetch_body_portion (stringer_t * *part*)

Definition at line 517 of file fetch.c.

References length, pl_init(), pl_null(), st_char_get(), and st_empty_out().

Referenced by imap_fetch_body().

stringer_t* imap_fetch_body_tag (stringer_t * *tag*, array_t * *items*)

Definition at line 469 of file fetch.c.

References `ar_length_get()`, `imap_get_st_ar()`, `number`, `st_cleanup()`, `st_free()`, `st_import()`, `st_merge`, and `upper_st()`.

Referenced by `imap_fetch_body()`.

`stringer_t* imap_fetch_bodystructure (mail_mime_t * mime)`

Definition at line 234 of file `fetch.c`.

References `ar_field_ptr()`, `ar_field_st()`, `ar_free()`, `ar_length_get()`, `mail_mime_t::body`, `mail_mime_t::children`, `mail_mime_t::header`, `imap_build_array()`, `imap_build_array_isliteral()`, `imap_fetch_bodystructure()`, `items`, `length`, `mail_header_fetch_cleaned()`, `mail_mime_content_encoding()`, `mail_mime_content_id()`, `mail_mime_type_group()`, `mail_mime_type_parameters()`, `mail_mime_type_sub()`, `ns_length_get()`, `pl_init()`, `PLACER`, `st_char_get()`, `st_cleanup()`, `st_cmp_cs_eq()`, `st_data_get()`, `st_free()`, `st_import()`, `st_length_get()`, `st_merge`, and `upper_st()`.

Referenced by `imap_fetch_bodystructure()`, and `imap_fetch_message()`.

`stringer_t* imap_fetch_envelope (stringer_t * header)`

Definition at line 189 of file `fetch.c`.

References `imap_build_array()`, `imap_parse_address()`, `mail_header_fetch_cleaned()`, `pl_init()`, `pl_null()`, `PLACER`, `placer_t`, `st_char_get()`, `st_cleanup()`, and `st_length_get()`.

Referenced by `imap_fetch_message()`.

`void imap_fetch_free_items (imap_fetch_dataitems_t * items)`

Definition at line 37 of file `fetch.c`.

References `mm_cleanup()`, `mm_free()`, `imap_fetch_dataitems_t::normal`, `imap_fetch_dataitems_t::normal_partial`, `imap_fetch_dataitems_t::peek`, and `imap_fetch_dataitems_t::peek_partial`.

Referenced by `imap_fetch()`, and `imap_parse_dataitems()`.

`imap_fetch_response_t* imap_fetch_message (connection_t * con, meta_message_t * meta, imap_fetch_dataitems_t * items)`

Definition at line 1048 of file `fetch.c`.

References `imap_fetch_dataitems_t::body`, `mail_mime_t::body`, `imap_fetch_dataitems_t::bodystructure`, `CONTIGUOUS`, `meta_message_t::created`, `imap_fetch_dataitems_t::envelope`, `imap_fetch_dataitems_t::flags`, `HEAP`, `connection_t::imap`, `imap_fetch_body()`, `imap_fetch_bodystructure()`, `imap_fetch_envelope()`, `imap_fetch_response_add()`, `imap_fetch_response_free()`, `imap_fetch_return_header()`, `imap_fetch_dataitems_t::internaldate`, `log_pedantic`, `mail_destroy()`, `mail_destroy_header()`, `mail_load_message()`, `mail_mime_update()`, `MAIL_STATUS_ANSWERED`, `MAIL_STATUS_DELETED`, `MAIL_STATUS_DRAFT`, `MAIL_STATUS_FLAGGED`, `MAIL_STATUS_RECENT`, `MAIL_STATUS_SEEN`, `MANAGED_T`, `meta_message_t::messagenum`, `mail_message_t::mime`, `imap_fetch_dataitems_t::normal`, `imap_fetch_dataitems_t::normal_partial`, `ns_length_get()`, `imap_fetch_dataitems_t::peek`, `imap_fetch_dataitems_t::peek_partial`, `PLACER`, `imap_fetch_dataitems_t::rfc822`, `imap_fetch_dataitems_t::rfc822_header`, `imap_fetch_dataitems_t::rfc822_size`, `imap_fetch_dataitems_t::rfc822_text`, `connection_t::server`, `meta_message_t::size`, `st_aprint_opts()`, `st_import()`, `st_length_get()`, `st_merge`, `meta_message_t::status`, `mail_message_t::text`, `imap_fetch_dataitems_t::uid`, and `meta_message_t::updated`.

Referenced by `imap_fetch()`.

`int_t imap_fetch_parse_partial (stringer_t * partial, size_t * start, size_t * length)`

Definition at line 614 of file `fetch.c`.

References `int32_conv_bl()`, `number`, `st_char_get()`, and `st_length_get()`.

Referenced by `imap_fetch_body()`.

`imap_fetch_response_t* imap_fetch_response_add (imap_fetch_response_t * response, stringer_t * key, stringer_t * value)`

`fetch_response.c`

Definition at line 30 of file `fetch_response.c`.

References `CONTIGUOUS`, `HEAP`, `imap_fetch_response_t::key`, `log_error`, `MANAGED_T`, `mm_alloc()`, `mm_free()`, `imap_fetch_response_t::next`, `st_dupe_opts()`, `st_free()`, and `imap_fetch_response_t::value`.

Referenced by `imap_fetch_body()`, and `imap_fetch_message()`.

`void imap_fetch_response_free (imap_fetch_response_t * response)`

Definition at line 15 of file `fetch_response.c`.

References `imap_fetch_response_t::key`, `mm_free()`, `imap_fetch_response_t::next`, `st_cleanup()`, and `imap_fetch_response_t::value`.

Referenced by `imap_fetch()`, `imap_fetch_body()`, `imap_fetch_message()`, `imap_fetch_return_header()`, `imap_fetch_return_message()`, `imap_fetch_return_mime()`, and `imap_fetch_return_text()`.

`stringer_t* imap_fetch_return_header (connection_t * con, meta_message_t * meta, mail_message_t ** message, stringer_t ** header, imap_fetch_response_t * output)`

Definition at line 690 of file `fetch.c`.

References `connection_t::imap`, `imap_fetch_response_free()`, `mail_destroy()`, `mail_load_header()`, `st_char_get()`, and `st_import()`.

Referenced by `imap_fetch_body()`, and `imap_fetch_message()`.

`mail_message_t* imap_fetch_return_message (connection_t * con, meta_message_t * meta, mail_message_t ** message, stringer_t ** header, imap_fetch_response_t * output)`

Definition at line 733 of file `fetch.c`.

References `connection_t::imap`, `imap_fetch_response_free()`, `mail_destroy()`, `mail_destroy_header()`, `mail_load_message()`, `mail_mime_update()`, and `connection_t::server`.

Referenced by `imap_fetch_body()`.

`mail_mime_t* imap_fetch_return_mime (connection_t * con, meta_message_t * meta, mail_message_t ** message, stringer_t ** header, imap_fetch_response_t * output)`

Definition at line 747 of file `fetch.c`.

References connection_t::imap, imap_fetch_response_free(), mail_destroy(), mail_destroy_header(), mail_load_message(), mail_mime_update(), and connection_t::server.

Referenced by imap_fetch_body().

**stringer_t* imap_fetch_return_text (connection_t * con, meta_message_t * meta,
mail_message_t ** message, stringer_t ** header, imap_fetch_response_t * output)**

Definition at line 719 of file fetch.c.

References connection_t::imap, imap_fetch_response_free(), mail_destroy(), mail_destroy_header(), mail_load_message(), and connection_t::server.

Referenced by imap_fetch_body().

int_t imap_flag_action (stringer_t * string)

flags.c

Definition at line 46 of file flags.c.

References IMAP_FLAG_ADD, IMAP_FLAG_REMOVE, IMAP_FLAG_REPLACE, IMAP_FLAG_SILENT, PLACER, and st_cmp_ci_eq().

Referenced by imap_store().

uint32_t imap_flag_parse (void * ptr, int_t type)

Definition at line 70 of file flags.c.

References ar_length_get(), IMAP_ARGUMENT_TYPE_ARRAY, imap_get_flag(), imap_get_st_ar(), imap_get_type_ar(), MAIL_STATUS_EMPTY, and number.

Referenced by imap_append(), and imap_store().

int_t imap_folder_compare (stringer_t * name, stringer_t * compare)

Definition at line 96 of file folders.c.

References PLACER, st_char_get(), st_cmp_ci_eq(), and st_length_get().

Referenced by imap_narrow_folders().

int_t imap_folder_create (uint64_t usernum, inx_t * folders, stringer_t * name)

Create a new imap folder.

Note:

If they don't already exist, any parent folders specified in the fully qualified folder name will automatically be created first.

Parameters:

usenum the numerical id of the user to whom the new imap folder will belong.

folders an inx holder containing the set of folders into which the new imap folder will be inserted.

name a managed string containing the fully qualified name of the new imap folder to be created.

Returns:

1 on success or <= 0 on failure. 0: An invalid parameter was passed to the function, or an internal failure occurred. -1: The specified new folder name was invalid. -2: The node depth of the new folder is too large (imap folder recursion limit reached [IMAP_FOLDER_RECURSION_LMIIT]). -3: Special folder "Inbox" cannot contain subfolders. -4: Part of the folder path name exceeds 16 characters (FOLDER_LENGTH_LIMIT) after being unescaped for quotation marks. -5: The specified folder name already exists.

Definition at line 269 of file folders.c.

References CONTIGUOUS, FOLDER_LENGTH_LIMIT, meta_folder_t::foldernum, HEAP, imap_count_folder_levels(), IMAP_FOLDER_RECURSION_LMIIT, imap_next_folder_order(), imap_valid_folder_name(), inx_insert(), log_pedantic, M_TYPE_UINT64, MANAGED_T, meta_data_insert_folder(), meta_folders_by_name(), mm_alloc(), mm_copy(), mm_free(), meta_folder_t::name, meta_folder_t::order, meta_folder_t::parent, pl_data_get(), pl_length_get(), PLACER, placer_t, st_cleanup(), st_cmp_ci_eq(), st_dupe(), st_dupe_opts(), st_free(), st_merge, st_replace(), tok_get_st(), multi_t::u64, and multi_t::val.

Referenced by imap_create(), imap_subscribe(), and portal_folder_mail_add().

stringer_t* imap_folder_name_escaped (inx_t * *folders*, meta_folder_t * *active*)

Get an imap folder's escaped name.

TODO: Since the Portal needs access to some of these folder manipulation functions the common logic should be extracted and moved into the folder object interface.

Parameters:

folders an inx holder containing the ancestor folders of the specified imap folder.

active a pointer to the meta folder object of the folder to have its name escaped.

Returns:

NULL on failure, or a pointer to a managed string containing the escaped name of the specified imap folder on success.

Definition at line 24 of file folders.c.

References meta_folders_name(), PLACER, st_free(), st_merge, and st_replace().

Referenced by imap_list(), and imap_lsub().

int_t imap_folder_remove (uint64_t *userid*, inx_t * *folders*, inx_t * *messages*, stringer_t * *name*)

Remove an imap folder, both in memory and on the database.

Note:

Any child messages of the specified folder will be deleted, but if it has child folders, the folder will not be deleted. The function also protects the special "Inbox" folder from deletion.

Parameters:

userid the numerical id of the user that owns the imap folder to be deleted.

folders an inx holder containing a collection of folders for looking up the specified imap folder by name.

messages an inx holder containing a collection of messages for looking up the contents of the specified imap folder.

name a managed string containing the name of the imap folder to be deleted.

Returns:

<= 0 on failure, or 1 on success. 0: General failure or if there was an error removing the folder from the database. -1: The specified folder name was invalid. -2: Removal failed because the "Inbox" folder was specified. -3: The specified folder could not be found.

Definition at line 417 of file folders.c.

References meta_folder_t::foldernum, meta_message_t::foldernum, imap_valid_folder_name(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_delete(), log_error, log_pedantic, M_TYPE_UINT64, mail_remove_message(), meta_message_t::messagenum, meta_data_delete_folder(), meta_folders_by_name(), meta_folders_children(), PLACER, placer_t, meta_message_t::server, meta_message_t::size, st_cmp_ci_eq(), tok_get_st(), multi_t::u64, and multi_t::val.

Referenced by imap_delete(), and portal_folder_mail_remove().

```
int_t imap_folder_rename (uint64_t usernum, inx_t * folders, stringer_t * original, stringer_t * rename)
```

Rename an imap folder.

Note:

Any parent folder components of the folder's fully qualified name will be created if they do not already exist.

Parameters:

usernum the numerical id of the user requesting the folder renaming.

folders an inx holder containing the folders to be searched for the folder specified to be renamed.

original a managed string containing the original name of the folder to be renamed.

rename a managed string containing the new name of the specified folder.

Returns:

1 on success or 0 or less on failure. 0: One of the parameters passed to the function was invalid, or there was a memory allocation failure. -1: Either the original or new folder name was invalid. -2: Was unable to rename the "Inbox" folder. -3: The specified imap folder did not exist. -4: Either the original or new name exceeded the imap folder recursion limit. -5: A folder already exists with the new folder name. -6: A segment of the folder name was larger than FOLDER_LENGTH_LIMIT (16 bytes).

Definition at line 497 of file folders.c.

References CONTIGUOUS, FOLDER_LENGTH_LIMIT, meta_folder_t::foldernum, HEAP, imap_count_folder_levels(), IMAP_FOLDER_RECURSION_LMIIT, imap_next_folder_order(), imap_valid_folder_name(), inx_insert(), log_error, log_pedantic, M_TYPE_UINT64, MANAGED_T, meta_data_insert_folder(), meta_data_update_folder_name(), meta_folders_by_name(), mm_alloc(), mm_copy(), mm_free(), mm_wipe(), meta_folder_t::name, meta_folder_t::order, meta_folder_t::parent, pl_data_get(), pl_length_get(), PLACER, placer_t, st_cleanup(), st_cmp_ci_eq(), st_dupe(), st_dupe_opts(), st_free(), st_merge, st_replace(), tok_get_st(), multi_t::u64, and multi_t::val.

Referenced by imap_rename(), and portal_endpoint_folders_rename().

```
int_t imap_folder_status (inx_t * folders, inx_t * messages, stringer_t * name,  
imap_folder_status_t * status)
```

Get the status of a folder.

Note:

This function will count the number of messages in a folder, as well as the number of messages marked recent or unseen, as well as the numerical id of the first message in the folder and the UIDNEXT of the specified folder.

Parameters:

folders an inx holder containing a list of folders to be searched for the specified folder.
messages an inx holder containing a complete list of a user's messages to be examined for gathering statistics.
name a managed string containing the name of the imap folder to be queried.
status a pointer to an imap folder status object to receive the folder's status information.

Returns:

1 on success or <= 0 on failure. 0: General failure. -1: The specified folder name was invalid. -2: The folder did not exist.

Definition at line 687 of file folders.c.

References `imap_folder_status_t::first`, `meta_message_t::foldernum`, `meta_folder_t::foldernum`, `imap_folder_status_t::foldernum`, `imap_valid_folder_name()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `log_pedantic`, `MAIL_STATUS_RECENT`, `MAIL_STATUS_SEEN`, `meta_message_t::messagenum`, `imap_folder_status_t::messages`, `meta_folders_by_name()`, `mm_wipe()`, `imap_folder_status_t::recent`, `meta_message_t::status`, `imap_folder_status_t::uidnext`, and `imap_folder_status_t::unseen`.

Referenced by `imap_examine()`, `imap_select()`, and `imap_status()`.

imap_arguments_t* imap_get_ar_ar (imap_arguments_t * array, size_t element)

Return the value of a specified object in an array as an imap arguments array.

Parameters:

array a pointer to an imap arguments array.
element the zero-based index of the object in the imap arguments array to be examined.

Returns:

NULL on failure, or a pointer to the specified object's value as an imap arguments array on success.

Definition at line 104 of file parse.c.

References `ar_length_get()`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_get_type_ar()`, and `log_pedantic`.

Referenced by `imap_fetch_body()`, `imap_id()`, `imap_parse_dataitems()`, `imap_search_messages_inner()`, and `imap_status()`.

uint32_t imap_get_flag (stringer_t * string)

Definition at line 15 of file flags.c.

References `MAIL_STATUS_ANSWERED`, `MAIL_STATUS_DELETED`, `MAIL_STATUS_DRAFT`, `MAIL_STATUS_FLAGGED`, `MAIL_STATUS_RECENT`, `MAIL_STATUS_SEEN`, `PLACER`, and `st_cmp_ci_eq()`.

Referenced by `imap_flag_parse()`.

void* imap_get_ptr (imap_arguments_t * array, size_t element)

Return the value of a specified object in an array as a generic pointer.

Parameters:

array a pointer to an imap arguments array.

element the zero-based index of the object in the imap arguments array to be examined.

Returns:

NULL on failure, or a pointer to the specified object's value on success.

Definition at line 50 of file parse.c.

References `ar_length_get()`, and `log_pedantic`.

Referenced by `imap_append()`, `imap_fetch_body()`, and `imap_store()`.

stringer_t* imap_get_st_ar (imap_arguments_t * array, size_t element)

Return the value of a specified object in an array as a managed string.

Parameters:

array a pointer to an imap arguments array.

element the zero-based index of the object in the imap arguments array to be examined.

Returns:

NULL on failure, or a pointer to the specified object's value as a managed string on success.

Definition at line 75 of file parse.c.

References `ar_length_get()`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_get_type_ar()`, and `log_pedantic`.

Referenced by `imap_append()`, `imap_copy()`, `imap_create()`, `imap_delete()`, `imap_examine()`, `imap_fetch()`, `imap_fetch_body()`, `imap_fetch_body_header()`, `imap_fetch_body_tag()`, `imap_flag_parse()`, `imap_id()`, `imap_list()`, `imap_login()`, `imap_lsub()`, `imap_parse_dataitems()`, `imap_rename()`, `imap_search_messages_inner()`, `imap_select()`, `imap_status()`, `imap_store()`, and `imap_subscribe()`.

int_t imap_get_type_ar (imap_arguments_t * array, size_t element)

TODO: The logic here is just plain ugly. Specifically the logic used to read from the network and advance the buffers.

Get the type code for a specified object in an array.

Note:

Valid return values include `IMAP_ARGUMENT_TYPE_ARRAY`, `IMAP_ARGUMENT_TYPE_ASTRING`, `IMAP_ARGUMENT_TYPE_QSTRING`, `IMAP_ARGUMENT_TYPE_NSTRING`, and `IMAP_ARGUMENT_TYPE_LITERAL`.

Parameters:

array a pointer to an imap arguments array.

element the zero-based index of the object in the imap arguments array to be examined.

Returns:

`IMAP_ARGUMENT_TYPE_EMPTY` on failure, or the element type code of the specified object on success.

Definition at line 24 of file parse.c.

References `ar_length_get()`, `IMAP_ARGUMENT_TYPE_EMPTY`, and `log_pedantic`.

Referenced by `imap_append()`, `imap_copy()`, `imap_create()`, `imap_delete()`, `imap_examine()`, `imap_fetch()`, `imap_fetch_body()`, `imap_fetch_body_header()`, `imap_flag_parse()`, `imap_get_ar_ar()`, `imap_get_st_ar()`,

imap_id(), imap_list(), imap_login(), imap_lsub(), imap_parse_dataitems(), imap_rename(),
imap_search_messages_inner(), imap_select(), imap_status(), imap_store(), and imap_subscribe().

void imap_id (connection_t * con)

Definition at line 1718 of file imap.c.

References ar_length_get(), build_stamp(), build_version(), con_addr_presentation(), con_print(), count,
connection_t::imap, IMAP_ARGUMENT_TYPE_ARRAY, imap_get_ar_ar(), imap_get_st_ar(),
imap_get_type_ar(), log_info, MANAGEDBUF, st_append_opts(), st_char_get(), st_free(), st_length_get(),
st_length_int(), st_sprint(), and time_print_local().

void imap_idle (connection_t * con)

Definition at line 1701 of file imap.c.

References con_print(), connection_t::imap, st_char_get(), and st_length_int().

void imap_init (connection_t * con)

The main imap entry point for all inbound client connections, as dispatched by the generic protocol handler
(display banner greeting).

Parameters:

con a pointer to the connection object of the newly connected client.

Returns:

This function returns no value.

Definition at line 1783 of file imap.c.

References build_version(), con_print(), con_reverse_enqueue(), con_secure(), server_t::domain,
imap_requeue(), connection_t::server, st_char_get(), st_length_get(), and st_length_int().

Referenced by protocol_enqueue().

void imap_invalid (connection_t * con)

Respond to an invalid imap command from a client.

Parameters:

con a pointer to the client connection that issued the bad command.

Returns:

This function returns no value.

Definition at line 62 of file imap.c.

References con_print(), server_t::delay, connection_t::imap, connection_t::protocol, connection_t::server,
st_char_get(), st_length_int(), server_t::violations, and connection_t::violations.

Referenced by imap_process(), and imap_starttls().

void imap_list (connection_t * con)

Definition at line 252 of file imap.c.

References `ar_length_get()`, `con_print()`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_folder_name_escaped()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `imap_narrow_folders()`, `inx_count()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_reset()`, `inx_cursor_value_next()`, `inx_free()`, `log_pedantic`, `magma_folder_name()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_folder_t::name`, `NULLER`, `meta_folder_t::parent`, `PLACER`, `st_char_get()`, `st_cmp_ci_eq()`, `st_free()`, and `st_length_int()`.

void imap_login (connection_t * con)

Attempt to perform a user login on an imap client connection.

Parameters:

con a pointer to the connection object of the remote session.

Returns:

This function returns no value.

LOW: `con->imap.bypass == 0` used to be here.

Definition at line 102 of file imap.c.

References `ar_length_get()`, `credential_t::auth`, `con_addr_presentation()`, `con_print()`, `con_secure()`, `CONTIGUOUS`, `credential_alloc_auth()`, `credential_free()`, `HEAP`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_get_st_ar()`, `imap_get_type_ar()`, `inx_count()`, `log_pedantic`, `MANAGED_T`, `MANAGEDBUF`, `meta_data_update_log()`, `meta_get()`, `META_GET_FOLDERS`, `META_GET_MESSAGES`, `META_PROT_IMAP`, `meta_remove()`, `meta_user_rlock()`, `META_USER_SSL`, `meta_user_unlock()`, `st_char_get()`, `st_cleanup()`, `st_dupe_opts()`, and `st_length_int()`.

void imap_logout (connection_t * con)

Terminate an IMAP session gracefully with a BYE message and destroy the underlying connection.

Parameters:

con the IMAP connection to be terminated.

Returns:

This function returns no value.

Definition at line 76 of file imap.c.

References `con_destroy()`, `con_print()`, `con_status()`, `con_write_bl()`, `connection_t::imap`, `st_char_get()`, and `st_length_int()`.

Referenced by `imap_process()`, and `imap_requeue()`.

void imap_lsub (connection_t * con)

Definition at line 341 of file imap.c.

References `ar_length_get()`, `con_print()`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_folder_name_escaped()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `imap_narrow_folders()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_reset()`, `inx_cursor_value_next()`, `inx_free()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_folder_t::name`, `NULLER`, `meta_folder_t::parent`, `PLACER`, `st_char_get()`, `st_cmp_ci_eq()`, `st_free()`, and `st_length_int()`.

int_t imap_message_copier (connection_t * con, meta_message_t * message, uint64_t target, uint64_t * outnum)

Definition at line 70 of file messages.c.

References meta_message_t::created, connection_t::imap, inx_insert(), log_pedantic, M_TYPE_UINT64, mail_copy_message(), MAIL_STATUS_DELETED, MAIL_STATUS_HIDDEN, MAIL_STATUS_RECENT, meta_message_t::messagenum, meta_message_dupe(), meta_messages_update_sequences(), mm_free(), OBJECT_MESSAGES, serial_get(), serial_increment(), meta_message_t::server, meta_message_t::sigkey, meta_message_t::signum, meta_message_t::size, meta_message_t::status, status, multi_t::type, multi_t::u64, and multi_t::val.

Referenced by imap_copy().

int_t imap_message_expunge (connection_t * con, meta_message_t * message)

Definition at line 58 of file messages.c.

References connection_t::imap, inx_delete(), M_TYPE_UINT64, mail_remove_message(), meta_message_t::messagenum, meta_message_t::server, and meta_message_t::size.

Referenced by imap_close(), and imap_expunge().

inx_t* imap_narrow_folders (inx_t * folders, stringer_t * reference, stringer_t * mailbox)

Definition at line 752 of file folders.c.

References meta_folder_t::foldernum, imap_folder_compare(), inx_alloc(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_insert(), log_pedantic, M_INX_LINKED, M_TYPE_UINT64, meta_folders_name(), mm_dupe(), mm_free(), st_free(), st_length_get(), st_merge, multi_t::u64, and multi_t::val.

Referenced by imap_list(), and imap_lsub().

inx_t* imap_narrow_messages (inx_t * messages, uint64_t selected, stringer_t * range, int_t uid)

Definition at line 1284 of file fetch.c.

References meta_message_t::foldernum, inx_alloc(), inx_cleanup(), inx_count(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_free(), inx_insert(), log_error, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, meta_message_t::messagenum, number, pl_char_get(), pl_empty(), pl_null(), placer_t, meta_message_t::sequencenum, st_char_get(), st_length_int(), tok_get_count_st(), tok_get_st(), multi_t::u64, uint64_conv_st(), and multi_t::val.

Referenced by imap_copy(), imap_fetch(), and imap_store().

uint64_t imap_next_folder_order (inx_t * folders, uint64_t parent)

Get the next order value for an imap folder.

Note:

The next order value is the current highest order value of any child folder in the specified directory, incremented by one.

Parameters:

folder an inx holder containing the collection of imap folders to be scanned.
parent the numerical id of the folder to be queried.

Returns:

0 on failure, or the next order value of the specified folder on success.

Definition at line 72 of file folders.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `meta_folder_t::order`, and `meta_folder_t::parent`.

Referenced by `imap_folder_create()`, and `imap_folder_rename()`.

void imap_noop (connection_t * con)

Definition at line 226 of file imap.c.

References `con_print()`, `connection_t::imap`, `imap_session_update()`, `st_char_get()`, and `st_length_int()`.

stringer_t* imap_parse_address (stringer_t * address)**parse_address.c**

Definition at line 216 of file parse_address.c.

References `imap_build_array()`, `imap_parse_address_breaker()`, `imap_parse_address_part()`, `pl_empty()`, `placer_t`, `st_free()`, `st_merge`, and `imap_fetch_response_t::value`.

Referenced by `imap_fetch_envelope()`.

placer_t imap_parse_address_breaker (stringer_t * address, uint32_t part)

Definition at line 163 of file parse_address.c.

References `length`, `pl_init()`, `pl_null()`, `placer_t`, `st_char_get()`, `st_length_get()`, and `status`.

Referenced by `imap_parse_address()`.

stringer_t* imap_parse_address_part (placer_t input)

Definition at line 32 of file parse_address.c.

References `imap_build_array()`, `imap_parse_address_put()`, `length`, `pl_data_get()`, `pl_length_get()`, `placer_t`, `st_alloc`, `st_char_get()`, `st_free()`, `st_length_get()`, `st_length_set()`, and `tok_get_st()`.

Referenced by `imap_parse_address()`.

void imap_parse_address_put (stringer_t * buffer, chr_t c)

LOW: Do we need an `imap_parse_address_group()` function?

Definition at line 17 of file parse_address.c.

References `st_avail_get()`, `st_char_get()`, `st_length_get()`, and `st_length_set()`.

Referenced by `imap_parse_address_part()`.

int_t imap_parse_arguments (connection_t * con, chr_t ** start, size_t * length)

Parse a chunk of input into an array of IMAP arguments.

See also:

imap_parse_qstring(), imap_parse_literal(), imap_parse_array(), imap_parse_astring()

Note:

This function scans a string for an array of arguments, performing the following logic when a particular character is encountered: `'` : Parse argument as quoted string with **imap_parse_qstring()**. `{` : Parse argument as literal string with **imap_parse_literal()**. `(` or `[` : Parse argument as array with **imap_parse_astring()**. other : Defaults to parsing argument as atomic string with **imap_parse_atomic()**. The supplied start and length pointers will be updated to reflect the input stream if the quoted string is parsed successfully.

Parameters:

start the address of a pointer to the start of the buffer to be parsed, that will be continually updated to point to the next argument in the sequence during the parsing loop.

length a pointer to a size_t variable that contains the length of the string to be parsed, that will be continually updated with the input stream position during the parsing loop.

Returns:

-1 on parsing error or 1 on success.

Definition at line 713 of file parse.c.

References `ar_append()`, `ar_free()`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `IMAP_ARGUMENT_TYPE_ASTRING`, `IMAP_ARGUMENT_TYPE_LITERAL`, `IMAP_ARGUMENT_TYPE_QSTRING`, `imap_parse_array()`, `imap_parse_astring()`, `imap_parse_literal()`, and `imap_parse_qstring()`.

Referenced by `imap_command_parser()`.

int_t imap_parse_array (int_t recursion, connection_t * con, imap_arguments_t ** array, chr_t ** start, size_t * length)

Extract the contents of an array argument and advance the position of the parser stream.

Note:

This function expects as input a string beginning either with `'` or `[`.

Parameters:

length a pointer to a size_t variable that contains the length of the string to be parsed, and that will be updated to reflect the length of the remainder of the input string that follows the parsed literal string.

recursion d

con the client IMAP connection passing the array as input to the server.

array -

start -

length -

Returns:

Definition at line 599 of file parse.c.

References `ar_append()`, `ar_free()`, `IMAP_ARGUMENT_TYPE_ARRAY`, `IMAP_ARGUMENT_TYPE_LITERAL`, and `IMAP_ARGUMENT_TYPE_NSTRING`.

IMAP_ARGUMENT_TYPE_QSTRING, IMAP_ARRAY_RECURSION_LIMIT, imap_parse_array(),
imap_parse_literal(), imap_parse_nstring(), imap_parse_qstring(), log_pedantic, and type().

Referenced by imap_parse_arguments(), and imap_parse_array().

int_t imap_parse_astring (stringer_t ** output, chr_t ** start, size_t * length)

Extract the contents of an atomic string and advance the position of the parser stream.

Note:

This function scans a string, expecting printable ASCII characters until it encounters a space, ,
, (, or [. If any other character is encountered before that point, an error will be returned. The supplied start
and length pointers will be updated to reflect the input stream if the quoted string is parsed successfully.

Parameters:

output the address of a managed string that will receive a copy of the atomic string's contents on success, or
NULL on failure.

start the address of a pointer to the start of the buffer to be parsed, that will also be updated to point to the
next argument in the sequence on success.

length a pointer to a size_t variable that contains the length of the string to be parsed, and which will be
updated to reflect the length of the remainder of the input string that follows the parsed atomic string.

Returns:

-1 on general error or if an invalid character was encountered, or 1 if the supplied atomic string was valid.

Definition at line 158 of file parse.c.

References log_pedantic, and st_import().

Referenced by imap_command_parser(), and imap_parse_arguments().

imap_fetch_dataitems_t* imap_parse_dataitems (imap_arguments_t * arguments)

Definition at line 49 of file fetch.c.

References ar_append(), ar_length_get(), ARRAY_TYPE_ARRAY, ARRAY_TYPE_POINTER,
imap_fetch_dataitems_t::body, imap_fetch_dataitems_t::bodystructure, imap_fetch_dataitems_t::envelope,
imap_fetch_dataitems_t::flags, IMAP_ARGUMENT_TYPE_ARRAY, imap_fetch_free_items(),
imap_get_ar_ar(), imap_get_st_ar(), imap_get_type_ar(), imap_fetch_dataitems_t::internaldate, log_error,
mm_alloc(), imap_fetch_dataitems_t::normal, imap_fetch_dataitems_t::normal_partial, number,
imap_fetch_dataitems_t::peek, imap_fetch_dataitems_t::peek_partial, PLACER,
imap_fetch_dataitems_t::rfc822, imap_fetch_dataitems_t::rfc822_header, imap_fetch_dataitems_t::rfc822_size,
imap_fetch_dataitems_t::rfc822_text, st_cmp_ci_eq(), st_cmp_cs_starts(), type(), and
imap_fetch_dataitems_t::uid.

Referenced by imap_fetch().

**int_t imap_parse_literal (connection_t * con, stringer_t ** output, chr_t ** start, size_t *
length)**

Extract the contents of a literal string and advance the position of the parser stream.

Note:

This function expects as input a string beginning with '{' and followed by a numerical string, an optional '+', and a closing '}'. After reading in the numerical size parameter, it then attempts to read in that many bytes of input from the network stream.

Parameters:

con the client IMAP connection passing the literal string as input to the server.

output the address of a managed string that will receive a copy of the literal string's contents on success, or NULL on failure or if it is zero length.

start the address of a pointer to the start of the buffer to be parsed (beginning with '{'), that will also be updated to point to the next argument in the sequence on success.

length a pointer to a `size_t` variable that contains the length of the string to be parsed, and that will be updated to reflect the length of the remainder of the input string that follows the parsed literal string.

Returns:

-1 on general or parse error or if an enclosing pair of double quotes was not found, or 1 if the supplied quoted string was valid.

Definition at line 385 of file `parse.c`.

References `connection_t::buffer`, `con_read()`, `con_read_line()`, `con_write_bl()`, `connection_t::line`, `line_pl_st()`, `log_pedantic`, `mm_copy()`, `mm_move()`, `connection_t::network`, `number`, `pl_empty()`, `pl_length_get()`, `pl_null()`, `st_alloc`, `st_char_get()`, `st_free()`, `st_length_set()`, and `uint64_conv_bl()`.

Referenced by `imap_parse_arguments()`, and `imap_parse_array()`.

`int_t imap_parse_nstring (stringer_t ** output, chr_t ** start, size_t * length, chr_t type)`

Definition at line 201 of file `parse.c`.

References `log_error`, and `st_import()`.

Referenced by `imap_parse_array()`.

`int_t imap_parse_qstring (stringer_t ** output, chr_t ** start, size_t * length)`

Extract the contents of a quoted string and advance the position of the parser stream.

Note:

This function scans a string beginning with '"' for a terminating '"', returning an error if `or` is encountered first. It is also able to handle escaped characters. The supplied start and length pointers will be updated to reflect the input stream if the quoted string is parsed successfully.

Parameters:

output the address of a managed string that will receive a copy of the quoted string's contents on success, or NULL on failure or if it is zero length.

start the address of a pointer to the start of the buffer to be parsed (beginning with '\'), that will also be updated to point to the next argument in the sequence on success.

length a pointer to a `size_t` variable that contains the length of the string to be parsed, and which will be updated to reflect the length of the remainder of the input string that follows the parsed quoted string.

Returns:

-1 on general error or if an enclosing pair of double quotes was not found, or 1 if the supplied quoted string was valid.

Definition at line 255 of file `parse.c`.

References `log_pedantic`, `st_alloc`, `st_char_get()`, and `st_length_set()`.

Referenced by `imap_parse_arguments()`, and `imap_parse_array()`.

void imap_process (connection_t * con)

Perform client command processing on an established imap session.

Note:

This function will read the next line of user input, parse the command, and then attempt to execute it with the appropriate handler.

Parameters:

con a pointer to the connection object underlying the imap session.

Returns:

This function returns no value.

Definition at line 66 of file commands.c.

References connection_t::command, command, con_print(), con_read_line(), con_write_bl(), server_t::cutoff, enqueue(), command_t::function, connection_t::imap, imap_command_parser(), imap_commands, imap_compare(), imap_invalid(), imap_logout(), imap_process(), imap_requeue(), command_t::length, connection_t::line, connection_t::network, pl_empty(), connection_t::protocol, requeue(), connection_t::server, connection_t::spins, st_char_get(), st_length_get(), st_length_int(), command_t::string, server_t::violations, and connection_t::violations.

Referenced by imap_process(), and imap_requeue().

stringer_t* imap_range_build (size_t length, uint64_t * numbers)

range.c

Definition at line 15 of file range.c.

References count, log_pedantic, st_free(), and st_merge.

Referenced by imap_copy().

void imap_rename (connection_t * con)

Definition at line 528 of file imap.c.

References ar_length_get(), con_print(), FOLDER_LENGTH_LIMIT, connection_t::imap, IMAP_ARGUMENT_TYPE_ARRAY, IMAP_FOLDER_RECURSION_LIMIT, imap_folder_rename(), imap_get_st_ar(), imap_get_type_ar(), meta_user_unlock(), meta_user_wlock(), OBJECT_FOLDERS, serial_get(), serial_increment(), st_char_get(), and st_length_int().

void imap_requeue (connection_t * con)

Requeue an imap connection for processing, or log it out if there was an error or excess of protocol violations.

Parameters:

con a pointer to the connection object of the imap session.

Returns:

This function returns no value.

Definition at line 48 of file commands.c.

References `con_status()`, `server_t::cutoff`, `enqueue()`, `imap_logout()`, `imap_process()`, `connection_t::protocol`, `connection_t::server`, `status`, `server_t::violations`, and `connection_t::violations`.

Referenced by `imap_init()`, and `imap_process()`.

void imap_search (connection_t * con)

Definition at line 1650 of file `imap.c`.

References `ar_length_get()`, `con_print()`, `con_status()`, `con_write_st()`, `HEAP`, `connection_t::imap`, `imap_search_messages()`, `inx_cursor_alloc()`, `inx_cursor_value_next()`, `inx_free()`, `JOINTED`, `MANAGED_T`, `MANAGEDBUF`, `meta_message_t::messagenum`, `meta_message_t::sequencenum`, `st_append_opts()`, `st_aprint_opts()`, `st_char_get()`, `st_free()`, `st_length_get()`, `st_length_int()`, `st_length_set()`, `st_sprint()`, and `status`.

int_t imap_search_flag (uint32_t status, uint32_t flag, int_t has)

`search.c`

Definition at line 271 of file `search.c`.

Referenced by `imap_search_messages_inner()`.

inx_t* imap_search_messages (connection_t * con)

LOW: Is a read lock necessary now that were using index reference counters and thread safe iteration cursors?

Definition at line 569 of file `search.c`.

References `count`, `meta_message_t::foldernum`, `connection_t::imap`, `imap_search_messages_inner()`, `inx_alloc()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_key_active()`, `inx_cursor_value_next()`, `inx_find()`, `inx_insert()`, `M_INX_LINKED`, `M_TYPE_UINT64`, `mail_destroy()`, `mail_destroy_header()`, `meta_message_t::messagenum`, `meta_message_dupe()`, `meta_message_free()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_message_t::sequencenum`, `status`, `multi_t::u64`, and `multi_t::val`.

Referenced by `imap_search()`.

int_t imap_search_messages_body (meta_user_t * user, mail_message_t ** data, meta_message_t * active, stringer_t * value)

Definition at line 202 of file `search.c`.

References `mail_load_message()`, `mail_mime_update()`, `st_free()`, and `st_search_ci()`.

Referenced by `imap_search_messages_inner()`.

int_t imap_search_messages_date (meta_user_t * user, mail_message_t ** data, stringer_t ** header, meta_message_t * active, stringer_t * date, int_t internal, int_t expected)

Definition at line 87 of file `search.c`.

References `meta_message_t::created`, `imap_search_messages_date_compare()`, `mail_header_fetch_cleaned()`, `mail_load_header()`, `ns_length_get()`, `pl_empty()`, `pl_init()`, `pl_null()`, `PLACER`, `placer_t`, `st_char_get()`, `st_free()`, `st_import()`, `st_length_get()`, `st_merge`, `tok_get_count_st()`, and `tok_get_st()`.

Referenced by `imap_search_messages_inner()`.

`int_t imap_search_messages_date_compare (stringer_t * one, stringer_t * two)`

Definition at line 17 of file `search.c`.

References `MONTH_LOOKUP`, `PLACER`, `placer_t`, `st_cmp_ci_eq()`, `tok_get_count_st()`, `tok_get_st()`, and `uint32_conv_st()`.

Referenced by `imap_search_messages_date()`.

`int_t imap_search_messages_header (meta_user_t * user, mail_message_t ** data, stringer_t ** header, meta_message_t * active, stringer_t * field, stringer_t * value)`

Definition at line 166 of file `search.c`.

References `mail_header_fetch_all()`, `mail_load_header()`, `pl_empty()`, `pl_init()`, `pl_null()`, `placer_t`, `st_char_get()`, `st_free()`, `st_length_get()`, and `st_search_ci()`.

Referenced by `imap_search_messages_inner()`.

`int_t imap_search_messages_inner (meta_user_t * user, mail_message_t ** message, stringer_t ** header, meta_message_t * current, imap_arguments_t * array, unsigned recursion)`

Definition at line 353 of file `search.c`.

References `ar_length_get()`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_get_ar_ar()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `imap_search_flag()`, `imap_search_messages_body()`, `imap_search_messages_date()`, `imap_search_messages_header()`, `imap_search_messages_inner()`, `imap_search_messages_range()`, `imap_search_messages_size()`, `imap_search_messages_text()`, `IMAP_SEARCH_RECURSION_LIMIT`, `imap_valid_sequence()`, `log_pedantic`, `MAIL_STATUS_ANSWERED`, `MAIL_STATUS_DELETED`, `MAIL_STATUS_DRAFT`, `MAIL_STATUS_FLAGGED`, `MAIL_STATUS_RECENT`, `MAIL_STATUS_SEEN`, `number`, `PLACER`, `st_cmp_ci_eq()`, and `meta_message_t::status`.

Referenced by `imap_search_messages()`, and `imap_search_messages_inner()`.

`int_t imap_search_messages_range (meta_message_t * active, stringer_t * range, int_t uid)`

Definition at line 281 of file `search.c`.

References `meta_message_t::messagenum`, `number`, `pl_char_get()`, `pl_empty()`, `pl_null()`, `placer_t`, `meta_message_t::sequencenum`, `tok_get_count_st()`, `tok_get_st()`, and `uint64_conv_st()`.

Referenced by `imap_search_messages_inner()`.

`int_t imap_search_messages_size (meta_message_t * active, stringer_t * value, int_t expected)`

Definition at line 250 of file `search.c`.

References `meta_message_t::size`, and `uint64_conv_st()`.

Referenced by `imap_search_messages_inner()`.

**int_t imap_search_messages_text (meta_user_t * user, mail_message_t ** data,
meta_message_t * active, stringer_t * value)**

Definition at line 226 of file search.c.

References mail_load_message(), mail_mime_update(), st_free(), and st_search_ci().

Referenced by imap_search_messages_inner().

void imap_select (connection_t * con)

Definition at line 811 of file imap.c.

References ar_length_get(), con_print(), imap_folder_status_t::first, imap_folder_status_t::foldernum, meta_message_t::foldernum, connection_t::imap, IMAP_ARGUMENT_TYPE_ARRAY, imap_folder_status(), imap_get_st_ar(), imap_get_type_ar(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), MAIL_STATUS_RECENT, imap_folder_status_t::messages, meta_data_flags_remove(), meta_user_unlock(), meta_user_wlock(), imap_folder_status_t::recent, st_char_get(), st_length_int(), meta_message_t::status, status, and imap_folder_status_t::uidnext.

void imap_session_destroy (connection_t * con)

sessions.c

Definition at line 98 of file sessions.c.

References ar_free(), meta_message_t::foldernum, connection_t::imap, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), mail_cache_reset(), MAIL_STATUS_RECENT, META_PROT_IMAP, meta_remove(), meta_user_unlock(), meta_user_wlock(), st_cleanup(), and meta_message_t::status.

Referenced by con_destroy().

int_t imap_session_update (connection_t * con)

Returns 0 if the selected folder wasn't modified, or 1 if things changed and the updated status should be sent to the client. A -1 is used to indicate the update check encountered a problem and should be retried later.

Definition at line 19 of file sessions.c.

References meta_message_t::foldernum, connection_t::imap, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), MAIL_STATUS_RECENT, meta_folders_update(), META_LOCKED, meta_messages_update(), meta_user_unlock(), meta_user_update(), meta_user_wlock(), OBJECT_FOLDERS, OBJECT_MESSAGES, OBJECT_USER, serial_get(), and meta_message_t::status.

Referenced by imap_check(), imap_expunge(), imap_noop(), and imap_store().

void imap_sort (void)

Sort the IMAP command table to be ready for binary searches.

Returns:

This function returns no value.

Definition at line 38 of file commands.c.

References imap_commands, and imap_compare().

Referenced by protocol_init().

void imap_starttls (connection_t * con)

Create a secure connection for an IMAP session.

TODO: Review error messages and update them with the appropriate response code. LOW: When should we check the serial number to see if the local data is stale and needs to be refreshed?

Note:

RFC 2595 / section 3.1 specifies that STARTTLS is only available in a non-authenticated state.

Parameters:

con the connection on top of which the ssl session will be established.

Returns:

This function returns no value.

Definition at line 24 of file imap.c.

References connection_t::buffer, con_print(), con_secure(), connection_t::imap, imap_invalid(), connection_t::line, log_pedantic, M_SSL_BIO_NOCLOSE, connection_t::network, pl_null(), connection_t::server, connection_t::sockd, connection_t::ssl, ssl_alloc(), st_char_get(), st_length_get(), st_length_set(), stats_increment_by_name(), and connection_t::status.

void imap_status (connection_t * con)

Definition at line 594 of file imap.c.

References ar_length_get(), con_print(), imap_folder_status_t::foldernum, connection_t::imap, IMAP_ARGUMENT_TYPE_ARRAY, imap_folder_status(), imap_get_ar_ar(), imap_get_st_ar(), imap_get_type_ar(), imap_folder_status_t::messages, meta_user_rlock(), meta_user_unlock(), NULLER, number, PLACER, imap_folder_status_t::recent, st_append_opts(), st_char_get(), st_cmp_ci_eq(), st_free(), st_length_get(), st_length_int(), status, imap_folder_status_t::uidnext, imap_folder_status_t::unseen, and values.

void imap_store (connection_t * con)

LOW: Shouldn't we be checking for stale status info so the update doesn't make decisions based on incorrect status data? On the other hand the actual IMAP logic is passed all the way through to the DB so even if the server ends up with incorrect status information, the database should remain accurate.

Definition at line 890 of file imap.c.

References ar_length_get(), con_print(), connection_t::imap, IMAP_ARGUMENT_TYPE_ARRAY, imap_duplicate_messages(), imap_flag_action(), imap_flag_parse(), IMAP_FLAG_SILENT, imap_get_ptr(), imap_get_st_ar(), imap_get_type_ar(), imap_narrow_messages(), imap_session_update(), imap_update_flags(), imap_valid_sequence(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_free(), MAIL_STATUS_ANSWERED, MAIL_STATUS_DELETED, MAIL_STATUS_DRAFT, MAIL_STATUS_FLAGGED, MAIL_STATUS_RECENT, MAIL_STATUS_SEEN, meta_message_t::messagenum, meta_user_unlock(), meta_user_wlock(), OBJECT_MESSAGES, meta_message_t::sequencenum, serial_get(), serial_increment(), st_char_get(), st_length_int(), and meta_message_t::status.

void imap_subscribe (connection_t * con)

Definition at line 681 of file imap.c.

References `ar_length_get()`, `con_print()`, `FOLDER_LENGTH_LIMIT`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_folder_create()`, `IMAP_FOLDER_RECURSION_LIMIT`, `imap_get_st_ar()`, `imap_get_type_ar()`, `meta_folders_by_name()`, `meta_user_unlock()`, `meta_user_wlock()`, `st_char_get()`, and `st_length_int()`.

void imap_unsubscribe (connection_t * con)

Definition at line 730 of file imap.c.

References `con_print()`, `connection_t::imap`, `st_char_get()`, and `st_length_int()`.

void imap_update_flags (meta_user_t * user, inx_t * messages, uint64_t foldernum, int_t action, uint32_t flags)

Definition at line 117 of file flags.c.

References `meta_message_t::foldernum`, `IMAP_FLAG_ADD`, `IMAP_FLAG_REMOVE`, `IMAP_FLAG_REPLACE`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `log_error`, `MAIL_STATUS_ANSWERED`, `MAIL_STATUS_DELETED`, `MAIL_STATUS_DRAFT`, `MAIL_STATUS_EMPTY`, `MAIL_STATUS_FLAGGED`, `MAIL_STATUS_RECENT`, `MAIL_STATUS_SEEN`, `meta_data_flags_add()`, `meta_data_flags_remove()`, `meta_data_flags_replace()`, `meta_message_t::status`, and `meta_user_t::usernum`.

Referenced by `imap_store()`.

bool_t imap_valid_folder_name (stringer_t * name)

Check to see if a name is a valid imap folder name.

Note:

The following naming checks are enforced: 1. The name can't be empty or begin with a period. 2. It cannot contain a non-printable character. 3. Trailing periods will be removed. 4. The folder name cannot contain consecutive periods.

Parameters:

name a managed string containing the imap folder name to be validated.

Returns:

true on success or false on failure.

Definition at line 162 of file folders.c.

References `log_pedantic`, `st_empty_out()`, `st_length_get()`, and `st_length_set()`.

Referenced by `contact_folder_create()`, `contact_folder_rename()`, `imap_folder_create()`, `imap_folder_remove()`, `imap_folder_rename()`, and `imap_folder_status()`.

int_t imap_valid_sequence (stringer_t * range)

Definition at line 16 of file fetch.c.

References `length`, and `st_empty_out()`.

Referenced by `imap_copy()`, `imap_fetch()`, `imap_search_messages_inner()`, and `imap_store()`.

magma/network/listeners.c File Reference

Functions used to listen for incoming connections on specific ports. Successful socket connections are then passed up to the controller for routing.

```
#include "magma.h"
```

Functions

- void **net_listen** (void)
 - *The main network handler entry point; poll the listening socket of each configured protocol server, and dispatch the protocol-specific handler for any inbound client connection that is accepted.* **bool_t net_init** (**server_t** *server)
 - *Initialize a server and listen for connections.* void **net_shutdown** (**server_t** *server)
- Close the listening socket associated with a server.*
-

Detailed Description

Functions used to listen for incoming connections on specific ports. Successful socket connections are then passed up to the controller for routing.

Definition in file **listeners.c**.

Function Documentation

bool_t net_init (**server_t** * server)

Initialize a server and listen for connections.

Note:

Each server listens on either an ipv4 or ipv6 address in non-blocking mode, and will be bound and listen on the configured port.

Parameters:

server a pointer to the server object to be initialized.

Returns:

true on successful initialization of the server, or false on failure.

Definition at line 100 of file listeners.c.

References `server_t::ipv6`, `server_t::listen_queue`, `log_critical`, `magma`, `mm_wipe()`, `net_set_buffer_length()`, `net_set_non_blocking()`, `net_set_reuseable_address()`, `server_t::network`, `magma_t::network_buffer`, `server_t::port`, `server_t::sockd`, and `magma_t::system`.

Referenced by `servers_network_start()`.

void net_listen (void)

The main network handler entry point; poll the listening socket of each configured protocol server, and dispatch the protocol-specific handler for any inbound client connection that is accepted.

See also:

protocol_process()

Returns:

This function returns no value.

Definition at line 21 of file listeners.c.

References `buflen`, `bufptr`, `data`, `log_critical`, `log_info`, `magma`, `mm_wipe()`, `server_t::network`, `protocol_process()`, `magma_t::servers`, `servers_get_by_socket()`, `server_t::sockd`, `status`, and `status_set()`.

Referenced by `main()`.

void net_shutdown (server_t * server)

Close the listening socket associated with a server.

Returns:

This function returns no value.

Definition at line 172 of file listeners.c.

References `server_t::network`, and `server_t::sockd`.

Referenced by `servers_network_stop()`.

magma/network/meta.h File Reference

Meta information structures/types for users, folders, messages, etc.

Data Structures

- struct `__attribute__`
- struct `__attribute__`
- struct `meta_message_t`
- struct `meta_folder_t`
- struct `meta_user_t`

Enumerations

- enum `META_GET` { `META_GET_NONE` = 0, `META_GET_MESSAGES` = 1, `META_GET_FOLDERS` = 2, `META_GET_CONTACTS` = 4 }
- enum `META_LOCK_STATUS` { `META_NEED_LOCK` = 0, `META_LOCKED` = 1 }
- enum `META_CHECKPOINT` { `META_CHECKPOINT_MESSAGES` = 0, `META_CHECKPOINT_FOLDERS` = 1, `META_CHECKPOINT_USER` = 2 }
- enum `META_USER_FLAGS` { `META_USER_NONE` = 0, `META_USER_SSL` = 1, `META_USER_ADVERTISING` = 2, `META_USER_OVERQUOTA` = 4, `META_USER_ENCRYPT_DATA` = 8 }
- enum `META_PROT` { `META_PROT_NONE` = 0, `META_PROT_SMTP` = 1, `META_PROT_POP` = 2, `META_PROT_IMAP` = 4, `META_PROT_WEB` = 8, `META_PROT_GENERIC` = 16 }
- enum { `CREDENTIAL_MAIL` = 0, `CREDENTIAL_AUTH` = 1 }

Detailed Description

Meta information structures/types for users, folders, messages, etc.

Definition in file `meta.h`.

Enumeration Type Documentation

anonymous enum

Enumerator:

CREDENTIAL_MAIL
CREDENTIAL_AUTH

Definition at line 52 of file `meta.h`.

enum `META_CHECKPOINT`

Enumerator:

META_CHECKPOINT_MESSAGES
META_CHECKPOINT_FOLDERS

META_CHECKPOINT_USER

Definition at line 29 of file meta.h.

enum META_GET

Enumerator:

META_GET_NONE
META_GET_MESSAGES
META_GET_FOLDERS
META_GET_CONTACTS

Definition at line 17 of file meta.h.

enum META_LOCK_STATUS

Enumerator:

META_NEED_LOCK
META_LOCKED

Definition at line 24 of file meta.h.

enum META_PROT

Enumerator:

META_PROT_NONE
META_PROT_SMTP
META_PROT_POP
META_PROT_IMAP
META_PROT_WEB
META_PROT_GENERIC

Definition at line 43 of file meta.h.

enum META_USER_FLAGS

Enumerator:

META_USER_NONE
META_USER_SSL
META_USER_ADVERTISING
META_USER_OVERQUOTA
META_USER_ENCRYPT_DATA

Definition at line 35 of file meta.h.

magma/network/network.h File Reference

The types and functions for abstracting access to network functionality.

```
#include "meta.h"
#include "sessions.h"
#include "pop.h"
#include "smtp.h"
#include "imap.h"
#include "http.h"
```

Data Structures

- struct **ip_t**
- struct **subnet_t**
- struct **command_t**
- struct **client_t**
- struct **connection_t**

Defines

- #define **MAGMA_NETWORK_H**

Typedefs

- typedef int16_t **octet_t**
- typedef int32_t **segment_t**

Enumerations

- enum { **REVERSE_ERROR** = -1, **REVERSE_EMPTY** = 0, **REVERSE_PENDING** = 1, **REVERSE_COMPLETE** = 2 }

Functions

- **ip_t * con_addr** (**connection_t** *con, **ip_t** *output)
- **addresses.c octet_t con_addr_octet** (**connection_t** *con, **int_t** position)
- *Get a specified 8-bit octet of a connection's peer IP address.* **stringer_t * con_addr_presentation** (**connection_t** *con, **stringer_t** *output)
- *Return a textual representation of the IP address of a specified connection handle.* **stringer_t * con_addr_reversed** (**connection_t** *con, **stringer_t** *output)
- *Get the reversed-IP address string for a client connection.* **segment_t con_addr_segment** (**connection_t** *con, **int_t** position)
- *Extract a specified 16 bit segment from a connection's peer IP address.* **stringer_t * con_addr_standard** (**connection_t** *con, **stringer_t** *output)
- *Get the IP address string for the client connection.* **stringer_t * con_addr_subnet** (**connection_t** *con, **stringer_t** *output)
- *Get the subnet string for a specified connection.* **uint32_t con_addr_word** (**connection_t** *con, **int_t** position)
- *Get a specified 32-bit segment of a connection's peer IP address.* **ip_t * ip_copy** (**ip_t** *dst, **ip_t** *src)
- *Create a copy of an IP address object.* **octet_t ip_octet** (**ip_t** *address, **int_t** position)
- *Extract a specified 8 bit octet from an IPv4 or IPv6 address.* **stringer_t * ip_presentation** (**ip_t** *address, **stringer_t** *output)
- **stringer_t * ip_reversed** (**ip_t** *address, **stringer_t** *output)
- *Get a reversed-IP string, for use in an RBL lookup.* **bool_t ip_address_equal** (**ip_t** *ip1, **ip_t** *ip2)
- *Determine whether two IP addresses are equal.* **segment_t ip_segment** (**ip_t** *address, **int_t** position)

- Extract a specified 16 bit segment from an IPv4 or IPv6 address. **stringer_t * ip_standard (ip_t *address, stringer_t *output)**
- Convert an IP address structure to string representation. **stringer_t * ip_subnet (ip_t *address, stringer_t *output)**
- Display the simple subnet string for an IP address. **uint32_t ip_word (ip_t *address, int_t position)**
- Get a specified 32-bit segment of an IP address. **bool_t ip_str_addr (chr_t *ipstr, ip_t *out)**
- Convert an IP address string to an IP address object. **bool_t ip_str_subnet (chr_t *substr, subnet_t *out)**
- Convert a string to an IP address object. **bool_t ip_matches_subnet (subnet_t *subnet, ip_t *addr)**
- Determines whether a given IP address matches a subnet mask. **uint64_t con_decrement_refs (connection_t *con)**
- **connections.c** void **con_destroy (connection_t *con)**
- Destroy and free a generic connection object after executing its protocol-specific destructor; update any statistics accordingly. **uint64_t con_increment_refs (connection_t *con)**
- Increment a connection object's reference counter. **connection_t * con_init (int cond, server_t *server)**
- Create a new connection object for a client connection. **bool_t con_init_network_buffer (connection_t *con)**
- Allocate and initialize a new network buffer for a connection, if it is not already associated with one. **int_t con_secure (connection_t *con)**
- Return the security level of a specified connection. **int_t con_status (connection_t *con)**
- Return the status of a specified connection. void **client_close (client_t *client)**
- **clients.c** **client_t * client_connect (chr_t *host, uint32_t port)**
- Establish a network client connection to a remote host. **int_t client_secure (client_t *client)**
- Establish an ssl connection with a network client instance. **int_t client_status (client_t *client)**
- Get the status of a network client. **bool_t net_set_buffer_length (int sd, int buffer_recv, int buffer_send)**
- **options.c** **bool_t net_set_keepalive (int sd, bool_t keepalive, int_t idle, int_t interval, int_t tolerance)**
- Set the keepalive flag, the. **bool_t net_set_linger (int sd, bool_t linger, int_t timeout)**
- Set the linger flag for a socket. **bool_t net_set_nodelay (int sd, bool_t nodelay)**
- Set the delay flag for a socket (to disable the Nagle algorithm). **bool_t net_set_non_blocking (int sd, bool_t blocking)**
- Set the blocking flag for a socket. **bool_t net_set_reuseable_address (int sd, bool_t reuse)**
- Set the reusable flag for a socket. **bool_t net_set_timeout (int sd, uint32_t timeout_recv, uint32_t timeout_send)**
- Set the send and receive timeouts for a socket. **int64_t client_read (client_t *client)**
- **read.c** **int64_t client_read_line (client_t *client)**
- Read a line of input from a network client session. **int64_t con_read (connection_t *con)**
- Read data from a connection, and store it in its internal buffer. **int64_t con_read_line (connection_t *con, bool_t block)**
- Read a line of input from a network connection. **stringer_t * con_reverse_check (connection_t *con, uint32_t timeout)**
- **reverse.c** void **con_reverse_domain (connection_t *con, stringer_t *domain, int_t status)**
- Set the domain name and reverse lookup status of a connection. void **con_reverse_enqueue (connection_t *con)**
- Queue a reverse DNS lookup on the specified connection, if one hasn't been performed. void **con_reverse_lookup (connection_t *con)**
- Perform a reverse DNS lookup on the remote end of a connection, and save the hostname. void **con_reverse_status (connection_t *con, int_t status)**
- **int net_accept (int sd)**
- **listeners.c** **bool_t net_init (server_t *server)**
- Initialize a server and listen for connections. void **net_listen (void)**
- The main network handler entry point; poll the listening socket of each configured protocol server, and dispatch the protocol-specific handler for any inbound client connection that is accepted. void **net_shutdown (server_t *server)**
- Close the listening socket associated with a server. **int64_t client_print (client_t *client, chr_t *format,...)**
- **write.c** **int64_t client_write (client_t *client, stringer_t *s)**

- Write data to a network client. `int64_t con_print (connection_t *con, chr_t *format,...)`
 - Write a formatted string to a network connection. `int64_t con_write_bl (connection_t *con, char *block, size_t length)`
 - Write data to a network connection. `int64_t con_write_ns (connection_t *con, char *string)`
 - Write a null-terminated string to a network connection. `int64_t con_write_pl (connection_t *con, placer_t string)`
 - Write a placer to a network connection. `int64_t con_write_st (connection_t *con, stringer_t *string)`
- Write a managed string to a network connection.
-

Detailed Description

The types and functions for abstracting access to network functionality.

Definition in file **network.h**.

Define Documentation

#define MAGMA_NETWORK_H

Definition at line 36 of file network.h.

Typedef Documentation

typedef int16_t octet_t

Definition at line 45 of file network.h.

typedef int32_t segment_t

Definition at line 46 of file network.h.

Enumeration Type Documentation

anonymous enum

Enumerator:

REVERSE_ERROR
REVERSE_EMPTY
REVERSE_PENDING
REVERSE_COMPLETE

Definition at line 48 of file network.h.

Function Documentation

void client_close (client_t * *client*)

clients.c

clients.c

Note:

If ssl is in use, this will also destroy the overlying ssl session.

Returns:

This function returns no value.

Definition at line 137 of file clients.c.

References client_t::buffer, mm_free(), client_t::sockd, client_t::ssl, ssl_free(), and st_cleanup().

Referenced by smtp_client_close(), and smtp_client_connect().

client_t* client_connect (chr_t * *host*, uint32_t *port*)

Establish a network client connection to a remote host.

Parameters:

host a pointer to a null-terminated string containing the hostname of the remote server.

port the port number of the server to which the connection will be established.

Returns:

NULL on failure or a pointer to a newly initialized network client object for the connection upon success.

Definition at line 62 of file clients.c.

References client_t::buffer, FOREIGNDATA, JOINTED, client_t::line, log_pedantic, MEMORYBUF, mm_alloc(), mm_free(), PLACER_T, client_t::sockd, st_alloc, STACK, and client_t::status.

Referenced by smtp_client_connect().

int64_t client_print (client_t * *client*, chr_t * *format*, ...)

write.c

write.c

See also:

client_write()

Parameters:

client the network client connection to which the supplied data will be written.

format the format string for the data to be written to the network client connection.

... a va_arg style collection of variables to be expanded by the passed format string.

Returns:

-1 on general network failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 279 of file write.c.

References client_write(), client_t::sockd, st_free(), and st_vaprint.

Referenced by `smtp_client_send_helo()`, `smtp_client_send_mailfrom()`, `smtp_client_send_nullfrom()`, and `smtp_client_send_rcptto()`.

int64_t client_read (client_t * *client*)

read.c

Definition at line 308 of file `read.c`.

References `client_t::buffer`, `bytes`, `client_t::line`, `mm_move()`, `pl_length_get()`, `pl_null()`, `client_t::sockd`, `client_t::ssl`, `SSL_get_error_d`, `ssl_read()`, `st_avail_get()`, `st_char_get()`, `st_data_get()`, `st_length_get()`, `st_length_set()`, `status`, and `client_t::status`.

int64_t client_read_line (client_t * *client*)

Read a line of input from a network client session.

Returns:

Definition at line 220 of file `read.c`.

References `client_t::buffer`, `bytes`, `client_t::line`, `line_pl_st()`, `mm_move()`, `pl_empty()`, `pl_length_get()`, `pl_null()`, `client_t::sockd`, `client_t::ssl`, `SSL_get_error_d`, `ssl_read()`, `st_avail_get()`, `st_char_get()`, `st_data_get()`, `st_empty()`, `st_length_get()`, `st_length_set()`, `status`, and `client_t::status`.

Referenced by `smtp_client_close()`, `smtp_client_connect()`, `smtp_client_send_data()`, `smtp_client_send_helo()`, `smtp_client_send_mailfrom()`, `smtp_client_send_nullfrom()`, and `smtp_client_send_rcptto()`.

int_t client_secure (client_t * *client*)

Establish an ssl connection with a network client instance.

Parameters:

client a pointer to the network client object to have its transport security upgraded.

Returns:

-1 on failure or 0 on success.

Definition at line 37 of file `clients.c`.

References `client_t::sockd`, `client_t::ssl`, `ssl_client_create()`, and `client_t::status`.

Referenced by `smtp_client_connect()`.

int_t client_status (client_t * *client*)

Get the status of a network client.

Parameters:

client a pointer to the network client object to be queried.

Returns:

-1 on error state, 0 for unknown status, or 1 if connected.

Definition at line 21 of file clients.c.

References client_t::sockd, and client_t::status.

int64_t client_write (client_t * *client*, stringer_t * *s*)

Write data to a network client.

Note:

This function works regardless of whether or not the client connection is ssl-enabled. If the network write requires multiple system calls, then this code will loop until all the data has been transmitted.

Parameters:

client the network client connection to which the supplied data will be written.

s a managed string containing the data to be written to the network client connection.

Returns:

-1 on general network failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 199 of file write.c.

References buflen, bufptr, length, log_pedantic, client_t::sockd, client_t::ssl, SSL_get_error_d, ssl_write(), st_empty_out(), and client_t::status.

Referenced by client_print(), smtp_client_close(), and smtp_client_send_data().

ip_t* con_addr (connection_t * *con*, ip_t * *output*)

addresses.c

addresses.c

Parameters:

con the input client connection.

output a pointer to an **ip_t** structure to receive the remote IP address, which is allocated for the caller if output is NULL.

Returns:

NULL on error, or a pointer to the IP address of the remote host.

Definition at line 395 of file addresses.c.

References ip_t::family, ip_t::ip4, ip_t::ip6, log_pedantic, mm_alloc(), mm_copy(), mm_free(), connection_t::network, and connection_t::sockd.

Referenced by con_addr_octet(), con_addr_presentation(), con_addr_reversed(), con_addr_segment(), con_addr_standard(), con_addr_subnet(), con_addr_word(), sess_create(), sess_get(), smtp_bypass_check(), and smtp_rcpt_to().

octet_t con_addr_octet (connection_t * *con*, int_t *position*)

Get a specified 8-bit octet of a connection's peer IP address.

Parameters:

con a pointer to the connection object to be examined.

position a zero-based index into the 8-bit octets that comprise the target IP address.

Returns:

-1 on error, or the specified 8-bit octet of the passed address on success.

Definition at line 525 of file addresses.c.

References `con_addr()`, and `ip_octet()`.

stringer_t* con_addr_presentation (connection_t * con, stringer_t * output)

Return a textual representation of the IP address of a specified connection handle.

Parameters:

con the remote connection to be queried.

output a managed string to store the result, which will be allocated for the caller if output is NULL.

Returns:

NULL on failure, or a pointer to a managed string containing a textual representation of the IP address.

Definition at line 435 of file addresses.c.

References `con_addr()`, and `ip_presentation()`.

Referenced by `contact_abuse_checks()`, `contact_abuse_increment_history()`, `contact_business()`, `imap_id()`, `imap_login()`, `mail_add_inbound_headers()`, `mail_add_outbound_headers()`, `pop_pass()`, `portal_meta()`, `register_abuse_checks()`, `register_data_insert_user()`, `register_session_cache()`, `register_session_get()`, and `smtp_rcpt_to()`.

stringer_t* con_addr_reversed (connection_t * con, stringer_t * output)

Get the reversed-IP address string for a client connection.

Parameters:

con the client connection to be queried.

output a managed string to receive the output, which will be allocated for the caller if passed as NULL.

Returns:

NULL on failure, or a pointer to a managed string containing a textual representation of the IP address on success.

Definition at line 471 of file addresses.c.

References `con_addr()`, and `ip_reversed()`.

Referenced by `smtp_check_greylist()`, and `smtp_check_rbl()`.

segment_t con_addr_segment (connection_t * con, int_t position)

Extract a specified 16 bit segment from a connection's peer IP address.

Parameters:

con a pointer to the connection object to be examined.

position the zero-indexed (starting at least-significant word) 16-bit segment of the IP address to be evaluated.

Returns:

-1 on failure, or the 32 bit-widened segment extracted from the supplied IP address.

Definition at line 543 of file addresses.c.

References `con_addr()`, and `ip_segment()`.

`stringer_t* con_addr_standard (connection_t * con, stringer_t * output)`

Get the IP address string for the client connection.

Parameters:

con the client connection to be queried.

output a managed string to receive the output, which will be allocated for the caller if output is NULL.

Returns:

NULL on failure, or a pointer to a managed string containing a textual representation of the IP address.

Definition at line 453 of file addresses.c.

References `con_addr()`, and `ip_standard()`.

Referenced by `register_abuse_check_blocklist()`, `register_abuse_checks()`, and `register_abuse_increment_history()`.

`stringer_t* con_addr_subnet (connection_t * con, stringer_t * output)`

Get the subnet string for a specified connection.

Parameters:

con the client connection to be queried.

output a managed string that will store the output of the subnet string lookup.

Returns:

NULL on failure, or a managed string containing a textual representation of a subnet address on success.

Definition at line 489 of file addresses.c.

References `con_addr()`, and `ip_subnet()`.

Referenced by `smtp_check_receive_quota()`, and `smtp_update_receive_stats()`.

`uint32_t con_addr_word (connection_t * con, int_t position)`

Get a specified 32-bit segment of a connection's peer IP address.

Parameters:

con a pointer to the connection object to be examined.

position a zero-based index into the 32-bit word(s) that comprise the target IP address.

Returns:

-1 if the connecting address can't be looked up, 0 on general error, or the specified 32-bit segment of the passed address on success.

Definition at line 507 of file addresses.c.

References `con_addr()`, and `ip_word()`.

Referenced by `portal_endpoint()`, `portal_process()`, and `portal_upload()`.

uint64_t con_decrement_refs (connection_t * con)

connections.c

connections.c

Parameters:

con the connection object being referenced.

Returns:

an integer containing the new number of references to the specified connection.

Definition at line 149 of file connections.c.

References connection_t::lock, mutex_lock(), mutex_unlock(), and connection_t::refs.

Referenced by con_destroy().

void con_destroy (connection_t * con)

Destroy and free a generic connection object after executing its protocol-specific destructor; update any statistics accordingly.

Parameters:

con a pointer to the connection to be destroyed.

Returns:

This function returns no value.

Definition at line 53 of file connections.c.

References connection_t::buffer, con_decrement_refs(), HTTP, http_session_destroy(), IMAP, imap_session_destroy(), connection_t::lock, mm_free(), MOLTEN, molten_session_destroy(), mutex_destroy(), connection_t::network, POP, pop_session_destroy(), server_t::protocol, connection_t::reverse, connection_t::server, SMTP, smtp_session_destroy(), connection_t::sockd, connection_t::ssl, ssl_free(), st_cleanup(), stats_decrement_by_name(), and SUBMISSION.

Referenced by con_reverse_lookup(), http_close(), imap_logout(), molten_quit(), pop_quit(), protocol_enqueue(), and smtp_quit().

uint64_t con_increment_refs (connection_t * con)

Increment a connection object's reference counter.

Parameters:

con the connection object being referenced.

Returns:

an integer containing the new number of references to the specified connection.

Definition at line 133 of file connections.c.

References connection_t::lock, mutex_lock(), mutex_unlock(), and connection_t::refs.

Referenced by con_init().

connection_t* con_init (int cond, server_t * server)

Create a new connection object for a client connection.

Parameters:

cond the socket descriptor of the inbound client connection that was just accepted.

server the handle of the server that serviced the inbound request.

Definition at line 189 of file connections.c.

References `con_increment_refs()`, `connection_t::lock`, `mm_alloc()`, `mm_free()`, `mutex_init()`, `connection_t::network`, `connection_t::server`, and `connection_t::sockd`.

Referenced by `protocol_process()`.

bool_t con_init_network_buffer (connection_t * con)

Allocate and initialize a new network buffer for a connection, if it is not already associated with one.

Note:

A network buffer of size `magma.system.network_buffer` bytes will be allocated for the connection if one does not exist.

Returns:

true if the connection object has been associated with a network buffer or false on failure.

Definition at line 165 of file connections.c.

References `connection_t::buffer`, `connection_t::line`, `log_info`, `magma`, `connection_t::network`, `magma_t::network_buffer`, `pl_null()`, `st_alloc`, and `magma_t::system`.

Referenced by `con_read()`, and `con_read_line()`.

int64_t con_print (connection_t * con, chr_t * format, ...)

Write a formatted string to a network connection.

See also:

`con_write_bl()`

Parameters:

con the connection across which the supplied data will be written.

format a format string for the output string to be written to the connection's remote client.

... a variable argument list of parameters for the specified format string.

Returns:

-1 on general failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 142 of file write.c.

References `buflen`, `bufptr`, `con_write_bl()`, `length`, `mm_alloc()`, `mm_free()`, `connection_t::network`, `connection_t::sockd`, and `connection_t::status`.

Referenced by `http_parse_header()`, `http_print_301()`, `http_response_header()`, `http_response_options()`, `imap_append()`, `imap_capability()`, `imap_check()`, `imap_close()`, `imap_copy()`, `imap_create()`, `imap_delete()`, `imap_examine()`, `imap_expunge()`, `imap_fetch()`, `imap_id()`, `imap_idle()`, `imap_init()`, `imap_invalid()`, `imap_list()`, `imap_login()`, `imap_logout()`, `imap_lsub()`, `imap_noop()`, `imap_process()`, `imap_rename()`, `imap_search()`, `imap_select()`, `imap_starttls()`, `imap_status()`, `imap_store()`, `imap_subscribe()`, `imap_unsubscribe()`, `molten_stats()`, `pop_capa()`, `pop_last()`, `pop_list()`, `pop_retr()`, `pop_stat()`, `pop_top()`,

pop_uidl(), register_print_captcha(), smtp_data(), smtp_data_inbound(), smtp_data_outbound(), smtp_disabled(), smtp_ehlo(), smtp_helo(), smtp_init(), smtp_rcpt_to(), and teacher_print_message().

int64_t con_read (connection_t * con)

Read data from a connection, and store it in its internal buffer.

Note:

This function handles reading data from both regular and ssl connections. If the connection's network buffer hasn't been allocated, it will be initialized.

Parameters:

con a pointer to the connection object from which the data will be read.

Returns:

-1 on internal error, or on a read error.

Definition at line 128 of file read.c.

References connection_t::buffer, bytes, con_init_network_buffer(), connection_t::line, mm_move(), connection_t::network, pl_length_get(), pl_null(), connection_t::sockd, connection_t::ssl, ssl_read(), ssl_shutdown_get(), st_avail_get(), st_char_get(), st_data_get(), st_length_get(), st_length_set(), status, and connection_t::status.

Referenced by http_body(), imap_parse_literal(), smtp_data_finish(), and smtp_data_read().

int64_t con_read_line (connection_t * con, bool_t block)

Read a line of input from a network connection.

Note:

This function handles reading data from both regular and ssl connections. This function continually attempts to read incoming data from the specified connection until a terminated line of input is received. If a new line is read, the length of that line is returned to the caller, including the trailing

. If the read returns -1 and wasn't caused by a syscall interruption or blocking error, -1 is returned, and the connection status is set to -1. If the read returns 0 and wasn't caused by a syscall interruption or blocking error, -2 is returned, and the connection status is set to 2. Once a character is reached, the length of the current line of input is returned to the user, and the connection status is set to 1.

Parameters:

con the network connection across which the line of data will be read.

Returns:

-1 on general failure, -2 if the connection was reset, or the length of the current line of input, including the trailing

Definition at line 27 of file read.c.

References connection_t::buffer, bytes, con_init_network_buffer(), connection_t::line, line_pl_st(), mm_move(), connection_t::network, pl_empty(), pl_length_get(), pl_null(), connection_t::sockd, connection_t::ssl, ssl_read(), st_avail_get(), st_char_get(), st_data_get(), st_empty(), st_length_get(), st_length_set(), status, and connection_t::status.

Referenced by `http_process()`, `imap_parse_literal()`, `imap_process()`, `molten_parse()`, `pop_process()`, `smtp_auth_login()`, `smtp_auth_plain()`, and `smtp_process()`.

`stringer_t* con_reverse_check (connection_t * con, uint32_t timeout)`

reverse.c

reverse.c

Note:

Polling will occur every 100ms, for as many seconds as specified by the user.

Parameters:

con the specified connection object to be polled that is the target of the DNS lookup.

timeout the number of seconds to wait for the lookup operation to complete.

Returns:

NULL on failure, or a pointer to a managed string containing the connection's peer hostname on success.

Definition at line 80 of file `reverse.c`.

References `connection_t::lock`, `mutex_lock()`, `mutex_unlock()`, `connection_t::network`, `connection_t::reverse`, `REVERSE_COMPLETE`, `REVERSE_PENDING`, and `status`.

Referenced by `mail_add_inbound_headers()`, and `mail_add_outbound_headers()`.

`void con_reverse_domain (connection_t * con, stringer_t * domain, int_t status)`

Set the domain name and reverse lookup status of a connection.

Note:

Possible values for status include `REVERSE_ERROR`, `REVERSE_EMPTY`, `REVERSE_PENDING`, and `REVERSE_COMPLETE`.

Parameters:

domain the new value of the connection's hostname.

status the new value of the connection's reverse lookup status.

Returns:

This function returns no value.

Definition at line 47 of file `reverse.c`.

References `connection_t::lock`, `mutex_lock()`, `mutex_unlock()`, `connection_t::network`, and `connection_t::reverse`.

Referenced by `con_reverse_lookup()`.

`void con_reverse_enqueue (connection_t * con)`

Queue a reverse DNS lookup on the specified connection, if one hasn't been performed.

Parameters:

con the connection object to be examined.

Returns:

This function returns no value.

Definition at line 20 of file reverse.c.

References `con_reverse_lookup()`, `enqueue()`, `connection_t::lock`, `mutex_lock()`, `mutex_unlock()`, `connection_t::network`, `connection_t::refs`, `connection_t::reverse`, `REVERSE_EMPTY`, and `REVERSE_PENDING`.

Referenced by `http_init()`, `imap_init()`, and `smtp_init()`.

void con_reverse_lookup (connection_t * con)

Perform a reverse DNS lookup on the remote end of a connection, and save the hostname.

Parameters:

con the connection object to be queried.

Returns:

This function returns no value.

Definition at line 112 of file reverse.c.

References `buflen`, `bufptr`, `con_destroy()`, `con_reverse_domain()`, `con_reverse_status()`, `connection_t::network`, `ns_length_get()`, `REVERSE_COMPLETE`, `REVERSE_ERROR`, `connection_t::sockd`, and `st_import()`.

Referenced by `con_reverse_enqueue()`.

void con_reverse_status (connection_t * con, int_t status)

Definition at line 64 of file reverse.c.

References `connection_t::lock`, `mutex_lock()`, `mutex_unlock()`, `connection_t::network`, and `connection_t::reverse`.

Referenced by `con_reverse_lookup()`.

int_t con_secure (connection_t * con)

Return the security level of a specified connection.

Parameters:

con the input client connection.

Returns:

1 for a secure connection, 0 for insecure, and -1 if the server does not support SSL/TLS.

Definition at line 20 of file connections.c.

References `server_t::context`, `connection_t::network`, `connection_t::server`, `server_t::ssl`, and `connection_t::ssl`.

Referenced by `contact_process()`, `http_print_301()`, `http_response_cookie()`, `imap_capability()`, `imap_init()`, `imap_login()`, `imap_starttls()`, `pop_capa()`, `pop_pass()`, `pop_starttls()`, `portal_endpoint()`, `portal_endpoint_auth()`, `portal_process()`, `portal_upload()`, `protocol_enqueue()`, `register_process()`, `sess_create()`, `sess_get()`, `smtp_ehlo()`, `smtp_rcpt_to()`, `smtp_starttls()`, and `teacher_process()`.

int_t con_status (connection_t * con)

Return the status of a specified connection.

Parameters:

con the input client connection.

Returns:

-1 on error, 0 for unknown status, 1 for connected, or 2 if the socket has been shutting down.

Definition at line 37 of file connections.c.

References `connection_t::network`, `connection_t::sockd`, and `connection_t::status`.

Referenced by `http_requeue()`, `imap_fetch()`, `imap_logout()`, `imap_requeue()`, `imap_search()`, `pop_quit()`, `pop_requeue()`, `smtp_quit()`, and `smtp_requeue()`.

int64_t con_write_bl (connection_t * *con*, char * *block*, size_t *length*)

Write data to a network connection.

HIGH: Create a simpler method of triggering a queue event following a connection write operation, and audit the code to ensure write calls do not accidentally orphan a connection by not queuing the connection upon completion. In other words, always ensure that **enqueue()** is being called on a connection after all processing is performed, so that it is not lost (whether it is to be kept or not).

Note:

This function works regardless of whether or not the connection is ssl-enabled. If the network write requires multiple system calls, then this code will loop until all the data has been transmitted.

Parameters:

con the connection across which the supplied data will be written.

block a pointer to a data buffer containing the data to be written to the connection's remote client.

length the length, in bytes, of the data buffer to be written.

Returns:

-1 on general network failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 28 of file write.c.

References `buflen`, `bufptr`, `log_pedantic`, `connection_t::network`, `connection_t::sockd`, `connection_t::ssl`, `SSL_get_error_d`, `ssl_write()`, and `connection_t::status`.

Referenced by `con_print()`, `con_write_ns()`, `con_write_pl()`, `con_write_st()`, `imap_fetch()`, `imap_logout()`, `imap_parse_literal()`, `imap_process()`, `molten_invalid()`, `molten_stats()`, `pop_delete()`, `pop_init()`, `pop_invalid()`, `pop_list()`, `pop_noop()`, `pop_pass()`, `pop_quit()`, `pop_retr()`, `pop_rset()`, `pop_starttls()`, `pop_top()`, `pop_uidl()`, `pop_user()`, `smtp_auth_login()`, `smtp_auth_plain()`, `smtp_data()`, `smtp_data_inbound()`, `smtp_data_outbound()`, `smtp_ehlo()`, `smtp_helo()`, `smtp_invalid()`, `smtp_mail_from()`, `smtp_noop()`, `smtp_quit()`, `smtp_rcpt_to()`, `smtp_rset()`, and `smtp_starttls()`.

int64_t con_write_ns (connection_t * *con*, char * *string*)

Write a null-terminated string to a network connection.

See also:

`con_write_bl()`

Parameters:

con the connection across which the supplied data will be written.

string a pointer to a null-terminated string containing the data to be written to the connection's remote client.

Returns:

-1 on general network failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 117 of file write.c.

References `con_write_bl()`, and `ns_length_get()`.

int64_t con_write_pl (connection_t * con, placer_t string)

Write a placer to a network connection.

See also:

`con_write_bl()`

Parameters:

con the connection across which the supplied data will be written.

string a placer pointing to the data to be written to the connection's remote client.

Returns:

-1 on general network failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 129 of file write.c.

References `con_write_bl()`, `pl_char_get()`, and `pl_length_get()`.

int64_t con_write_st (connection_t * con, stringer_t * string)

Write a managed string to a network connection.

See also:

`con_write_bl()`

Parameters:

con the connection across which the supplied data will be written.

string a managed string containing the data to be written to the connection's remote client.

Returns:

-1 on general network failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 105 of file write.c.

References `con_write_bl()`, `st_char_get()`, and `st_length_get()`.

Referenced by `contact_print_form()`, `contact_print_message()`, `http_print_400()`, `http_print_403()`, `http_print_404()`, `http_print_405()`, `http_print_500()`, `http_print_500_log()`, `http_print_501()`, `http_response()`, `imap_fetch()`, `imap_search()`, `pop_retr()`, `pop_top()`, `portal_endpoint_attachments_progress()`, `portal_endpoint_error()`, `portal_endpoint_response()`, `portal_endpoint_search()`, `portal_print_login()`, `register_print_captcha()`, `register_print_message()`, `register_print_step1()`, `register_print_step2()`, `register_print_step3()`, `smtp_data_outbound()`, `statistics_process()`, `teacher_print_form()`, and `teacher_print_message()`.

bool_t ip_address_equal (ip_t * ip1, ip_t * ip2)

Determine whether two IP addresses are equal.

Parameters:

ip1 a pointer to the first ip address object in the comparison.

ip2 a pointer to the second ip address object in the comparison.

Returns:

true if the two addresses are equal, or false if they are not.

Definition at line 373 of file addresses.c.

References ip_t::family, ip_t::ip4, and ip_t::ip6.

Referenced by sess_get().

ip_t* ip_copy (ip_t * dst, ip_t * src)

Create a copy of an IP address object.

LOW: Create a set of sock_addr_* functions and update the con_addr_* functions to use use them. Then add a matching set of sock_peer_* and con_peer_* functions based around the getpeername() function. Add functions for getting a socket's port number as well. Also note that peer_should_ mean the remote address, and addr_should_ mean the local address.

```
ip_t ip; char working[64]; struct stat64 info; struct sockaddr_in6 saddr; socklen_t len = sizeof(struct sockaddr_in6);
```

```
if (!fstat64(connection, &info) && S_ISSOCK(info.st_mode) && !(ret = getpeername(connection, &saddr, &len))) { if (len == sizeof(struct sockaddr_in6) && saddr.sin6_family == AF_INET6) { mm_copy(&(ip.ip6), &(saddr.sin6_addr), sizeof(struct in6_addr)); ip.family = AF_INET6; log_info("%s:%u accepted.", st_char_get(ip_presentation(&ip, PLACER(working, 64))), ntohs(saddr.sin6_port)); } else if (len == sizeof(struct sockaddr_in) && ((struct sockaddr_in *)&saddr)->sin_family == AF_INET) { mm_copy(&(ip.ip4), &(((struct sockaddr_in *)&saddr)->sin_addr), sizeof(struct in_addr)); ip.family = AF_INET; log_info("%s:%u accepted.", st_char_get(ip_presentation(&ip, PLACER(working, 64))), ntohs(((struct sockaddr_in *)&saddr)->sin_port)); } }
```

Parameters:

dst a pointer to the destination IP address object to receive the copy.

src a pointer to the source IP address object to be copied.

NULL on failure, or a pointer to the destination IP address buffer on success.

Definition at line 45 of file addresses.c.

References mm_copy().

Referenced by ip_str_subnet(), and sess_create().

bool_t ip_matches_subnet (subnet_t * subnet, ip_t * addr)

Determines whether a given IP address matches a subnet mask.

Parameters:

subnet a pointer to a subnet that the address will be compared against.

addr a pointer to the IP address of interest.

Returns:

true if the specified IP address falls into the subnet, or false if it does not.

Definition at line 671 of file addresses.c.

References subnet_t::address, ip_t::family, ip_t::ip, and subnet_t::mask.

Referenced by smtp_bypass_check().

octet_t ip_octet (ip_t * *address*, int_t *position*)

Extract a specified 8 bit octet from an IPv4 or IPv6 address.

Parameters:

address the IP address object to be examined.

position the zero-indexed (starting at least-significant byte) byte number of the IP address to be evaluated.

Returns:

-1 on failure, or the 16 bit-widened octet extracted from the supplied IP address.

Definition at line 60 of file addresses.c.

References ip_t::family, ip_t::ip4, and ip_t::ip6.

Referenced by con_addr_octet().

stringer_t* ip_presentation (ip_t * *address*, stringer_t * *output*)

Definition at line 184 of file addresses.c.

References buflen, bufptr, ip_t::family, ip_t::ip4, ip_t::ip6, log_pedantic, ns_length_get(), st_alloc, st_avail_get(), st_char_get(), st_free(), st_length_set(), st_valid_destination(), and st_valid_tracked().

Referenced by con_addr_presentation(), and signal_shutdown().

stringer_t* ip_reversed (ip_t * *address*, stringer_t * *output*)

Get a reversed-IP string, for use in an RBL lookup.

Parameters:

address a pointer to the IP address to be displayed as a string.

output a pointer to the managed string to receive the reversed-IP string, which will be allocated for the caller if output is NULL.

Returns:

NULL on failure, or a pointer to a managed string containing the reversed-IP string on success.

Definition at line 301 of file addresses.c.

References ip_t::family, ip_t::ip4, ip_t::ip6, log_pedantic, st_alloc, st_avail_get(), st_char_get(), st_length_set(), st_sprint(), st_valid_destination(), and st_valid_tracked().

Referenced by con_addr_reversed().

segment_t ip_segment (ip_t * *address*, int_t *position*)

Extract a specified 16 bit segment from an IPv4 or IPv6 address.

Parameters:

address the IP address object to be examined.

position the 0-indexed (starting at least-significant word) 16-bit segment of the IP address to be evaluated.

Returns:

-1 on failure, or the 32 bit-widened segment extracted from the supplied IP address.

Definition at line 85 of file addresses.c.

References `ip_t::family`, `ip_t::ip4`, and `ip_t::ip6`.

Referenced by `con_addr_segment()`.

`stringer_t* ip_standard (ip_t * address, stringer_t * output)`

Convert an IP address structure to string representation.

Parameters:

address a pointer to the IP address to be converted.

output a pointer to the managed string to receive the converted value, which will be allocated for the caller if output is NULL.

Returns:

NULL on failure, or a pointer to a managed string containing a textual representation of the IP address.

Definition at line 226 of file addresses.c.

References `ip_t::family`, `ip_t::ip4`, `ip_t::ip6`, `log_pedantic`, `st_alloc`, `st_avail_get()`, `st_char_get()`, `st_free()`, `st_length_set()`, `st_sprint()`, `st_valid_destination()`, `st_valid_tracked()`, and `st_wipe()`.

Referenced by `con_addr_standard()`.

`bool_t ip_str_addr (chr_t * ipstr, ip_t * out)`

Convert an IP address string to an IP address object.

Parameters:

ipstr a pointer to a null-terminated string containing the IP string to be parsed.

out a pointer to an IP address object that will receive result of the operation.

Returns:

true if the input string was a valid IP address or false if it was unable to be parsed.

Definition at line 561 of file addresses.c.

References `ip_t::family`, `ip_t::ip4`, `ip_t::ip6`, and `ns_length_get()`.

Referenced by `ip_str_subnet()`.

`bool_t ip_str_subnet (chr_t * substr, subnet_t * out)`

Convert a string to an IP address object.

Parameters:

ipstr a pointer to a null-terminated string containing the IP string to be parsed.
out a pointer to an IP address object that will receive result of the operation.

Returns:

true if the input string was a valid IP address or false if it was unable to be parsed.

Definition at line 611 of file addresses.c.

References subnet_t::address, ip_t::family, ip_copy(), ip_str_addr(), subnet_t::mask, ns_free(), ns_import(), ns_length_get(), NULLER, pl_char_get(), pl_length_get(), placer_t, tok_get_count_st(), tok_get_ns(), and uint8_conv_bl().

Referenced by smtp_add_bypass_entry().

stringer_t* ip_subnet (ip_t * address, stringer_t * output)

Display the simple subnet string for an IP address.

Note:

IPv4 addresses will yield a /24 subnet address, and IPv6 addresses will result in a /48 subnet address.
 Neither address will be displayed with the trailing zero-octet(s).

Parameters:

address a pointer to the ip address to be examined.
output a pointer to a managed string to receive the subnet string, or if passed as NULL, it will be allocated for the caller.

Returns:

NULL on failure, or a pointer to a managed string containing the result on success.

Definition at line 135 of file addresses.c.

References ip_t::family, ip_t::ip4, ip_t::ip6, log_pedantic, st_alloc, st_avail_get(), st_free(), st_length_get(), st_sprint(), st_valid_avail(), and st_valid_destination().

Referenced by con_addr_subnet().

uint32_t ip_word (ip_t * address, int_t position)

Get a specified 32-bit segment of an IP address.

Note:

This function operates on both ipv4 and ipv6 addresses.

Parameters:

address a pointer to the IP address object to be examined.
position a zero-based index into the 32-bit word(s) that comprise the target IP address.

Returns:

0 on error, or the specified 32-bit segment of the passed address.

Definition at line 113 of file addresses.c.

References ip_t::family, ip_t::ip4, and ip_t::ip6.

Referenced by con_addr_word().

int net_accept (int *sd*)

listeners.c

bool_t net_init (server_t * *server*)

Initialize a server and listen for connections.

Note:

Each server listens on either an ipv4 or ipv6 address in non-blocking mode, and will be bound and listen on the configured port.

Parameters:

server a pointer to the server object to be initialized.

Returns:

true on successful initialization of the server, or false on failure.

Definition at line 100 of file listeners.c.

References `server_t::ipv6`, `server_t::listen_queue`, `log_critical`, `magma`, `mm_wipe()`, `net_set_buffer_length()`, `net_set_non_blocking()`, `net_set_reuseable_address()`, `server_t::network`, `magma_t::network_buffer`, `server_t::port`, `server_t::sockd`, and `magma_t::system`.

Referenced by `servers_network_start()`.

void net_listen (void)

The main network handler entry point; poll the listening socket of each configured protocol server, and dispatch the protocol-specific handler for any inbound client connection that is accepted.

See also:

protocol_process()

Returns:

This function returns no value.

Definition at line 21 of file listeners.c.

References `buflen`, `bufptr`, `data`, `log_critical`, `log_info`, `magma`, `mm_wipe()`, `server_t::network`, `protocol_process()`, `magma_t::servers`, `servers_get_by_socket()`, `server_t::sockd`, `status`, and `status_set()`.

Referenced by `main()`.

bool_t net_set_buffer_length (int *sd*, int *buffer_recv*, int *buffer_send*)

options.c

options.c

Parameters:

sd the socket descriptor to be configured.

buffer_recv the length, in bytes, of the socket receive buffer.

buffer_send the length, in bytes, of the socket send buffer.

Returns:

true if both buffer values were set successfully, or false on failure.

Definition at line 164 of file options.c.

References buflen, bufptr, and log_pedantic.

Referenced by net_init().

bool_t net_set_keepalive (int sd, bool_t keepalive, int_t idle, int_t interval, int_t tolerance)

Set the keepalive flag, the.

Controls whether to send periodic messages along idle socket connections to ensure the connected peer is still present. If the peer fails to respond the connection can be broken.

Parameters:

sd The socket being configured.

keepalive Whether to enable the keepalive process.

idle Controls how long a connection must be idle (in seconds) before the system will start sending keep alive probes.

interval They delay (in seconds) between each keep alive probe.

tolerance The maximum number of failed probes that must be sent before dropping a connection.

Returns:

Returns true if all the parameters are configured properly, otherwise false to indicate an error.

Definition at line 28 of file options.c.

References buflen, bufptr, and log_pedantic.

bool_t net_set_linger (int sd, bool_t linger, int_t timeout)

Set the linger flag for a socket.

Parameters:

sd the socket descriptor to be adjusted.

linger a boolean variable specifying whether the socket will linger before closing until all queued messages for the socket have been sent.

timeout a value specifying the number of seconds to linger, if linger is true.

Returns:

true if the flag was successfully set or false on failure.

Definition at line 68 of file options.c.

References buflen, bufptr, and log_pedantic.

bool_t net_set_nodelay (int sd, bool_t nodelay)

Set the delay flag for a socket (to disable the Nagle algorithm).

Parameters:

sd the socket descriptor to be adjusted.

nodelay a boolean variable specifying whether the Nagle algorithm should be disabled (true) or not (false).

Returns:

true if the flag was successfully set or false on failure.

Definition at line 49 of file options.c.

References buflen, bufptr, and log_pedantic.

bool_t net_set_non_blocking (int *sd*, bool_t *blocking*)

Set the blocking flag for a socket.

Parameters:

sd the socket descriptor to be adjusted.

blocking a boolean variable specifying whether blocking should be set.

Returns:

true if the flag was successfully set or false on failure.

Definition at line 107 of file options.c.

References buflen, bufptr, log_error, and log_pedantic.

Referenced by net_init().

bool_t net_set_reuseable_address (int *sd*, bool_t *reuse*)

Set the reusable flag for a socket.

Parameters:

sd the socket descriptor to be adjusted.

reuse a boolean variable specifying whether the listening socket should be reusable or not.

Returns:

true if the flag was successfully set or false on failure.

Definition at line 89 of file options.c.

References buflen, bufptr, and log_pedantic.

Referenced by net_init().

bool_t net_set_timeout (int *sd*, uint32_t *timeout_recv*, uint32_t *timeout_send*)

Set the send and receive timeouts for a socket.

Parameters:

sd the socket descriptor to be configured.

timeout_recv the receive timeout value for the socket, in seconds.

timeout_send the send timeout value for the socket, in seconds.

Returns:

true if both timeout values were set successfully, or false on failure.

Definition at line 132 of file options.c.

References buflen, bufptr, log_pedantic, and mm_wipe().

Referenced by protocol_process(), and smtp_client_connect().

void net_shutdown (server_t * server)

Close the listening socket associated with a server.

Returns:

This function returns no value.

Definition at line 172 of file listeners.c.

References `server_t::network`, and `server_t::sockd`.

Referenced by `servers_network_stop()`.

magma/network/options.c File Reference

Functions used to set network socket options.

```
#include "magma.h"
```

Functions

- **bool_t net_set_keepalive** (int sd, **bool_t** keepalive, **int_t** idle, **int_t** interval, **int_t** tolerance)
- *Set the keepalive flag, the.* **bool_t net_set_nodelay** (int sd, **bool_t** nodelay)
- *Set the delay flag for a socket (to disable the Nagle algorithm).* **bool_t net_set_linger** (int sd, **bool_t** linger, **int_t** timeout)
- *Set the linger flag for a socket.* **bool_t net_set_reuseable_address** (int sd, **bool_t** reuse)
- *Set the reusable flag for a socket.* **bool_t net_set_non_blocking** (int sd, **bool_t** blocking)
- *Set the blocking flag for a socket.* **bool_t net_set_timeout** (int sd, uint32_t timeout_rcv, uint32_t timeout_snd)
- *Set the send and receive timeouts for a socket.* **bool_t net_set_buffer_length** (int sd, int buffer_rcv, int buffer_snd)

Set the send and receive buffers for a socket at the system level.

Detailed Description

Functions used to set network socket options.

Definition in file **options.c**.

Function Documentation

bool_t net_set_buffer_length (int sd, int buffer_rcv, int buffer_snd)

Set the send and receive buffers for a socket at the system level.

options.c

Parameters:

sd the socket descriptor to be configured.

buffer_rcv the length, in bytes, of the socket receive buffer.

buffer_snd the length, in bytes, of the socket send buffer.

Returns:

true if both buffer values were set successfully, or false on failure.

Definition at line 164 of file options.c.

References buflen, bufptr, and log_pedantic.

Referenced by net_init().

bool_t net_set_keepalive (int sd, **bool_t** keepalive, **int_t** idle, **int_t** interval, **int_t** tolerance)

Set the keepalive flag, the.

Controls whether to send periodic messages along idle socket connections to ensure the connected peer is still present. If the peer fails to respond the connection can be broken.

Parameters:

sd The socket being configured.

keepalive Whether to enable the keepalive process.

idle Controls how long a connection must be idle (in seconds) before the system will start sending keep alive probes.

interval They delay (in seconds) between each keep alive probe.

tolerance The maximum number of failed probes that must be sent before dropping a connection.

Returns:

Returns true if all the parameters are configured properly, otherwise false to indicate an error.

Definition at line 28 of file options.c.

References buflen, bufptr, and log_pedantic.

bool_t net_set_linger (int *sd*, bool_t *linger*, int_t *timeout*)

Set the linger flag for a socket.

Parameters:

sd the socket descriptor to be adjusted.

linger a boolean variable specifying whether the socket will linger before closing until all queued messages for the socket have been sent.

timeout a value specifying the number of seconds to linger, if *linger* is true.

Returns:

true if the flag was successfully set or false on failure.

Definition at line 68 of file options.c.

References buflen, bufptr, and log_pedantic.

bool_t net_set_nodelay (int *sd*, bool_t *nodelay*)

Set the delay flag for a socket (to disable the Nagle algorithm).

Parameters:

sd the socket descriptor to be adjusted.

nodelay a boolean variable specifying whether the Nagle algorithm should be disabled (true) or not (false).

Returns:

true if the flag was successfully set or false on failure.

Definition at line 49 of file options.c.

References buflen, bufptr, and log_pedantic.

bool_t net_set_non_blocking (int *sd*, bool_t *blocking*)

Set the blocking flag for a socket.

Parameters:

sd the socket descriptor to be adjusted.

blocking a boolean variable specifying whether blocking should be set.

Returns:

true if the flag was successfully set or false on failure.

Definition at line 107 of file options.c.

References buflen, bufptr, log_error, and log_pedantic.

Referenced by net_init().

bool_t net_set_reuseable_address (int *sd*, bool_t *reuse*)

Set the reusable flag for a socket.

Parameters:

sd the socket descriptor to be adjusted.

reuse a boolean variable specifying whether the listening socket should be reusable or not.

Returns:

true if the flag was successfully set or false on failure.

Definition at line 89 of file options.c.

References buflen, bufptr, and log_pedantic.

Referenced by net_init().

bool_t net_set_timeout (int *sd*, uint32_t *timeout_recv*, uint32_t *timeout_send*)

Set the send and receive timeouts for a socket.

Parameters:

sd the socket descriptor to be configured.

timeout_recv the receive timeout value for the socket, in seconds.

timeout_send the send timeout value for the socket, in seconds.

Returns:

true if both timeout values were set successfully, or false on failure.

Definition at line 132 of file options.c.

References buflen, bufptr, log_pedantic, and mm_wipe().

Referenced by protocol_process(), and smtp_client_connect().

magma/network/pop.h File Reference

The POP control structures.

Data Structures

- struct `__attribute__`
-

Detailed Description

The POP control structures.

Definition in file **pop.h**.

magma/servers/pop/pop.h File Reference

The entry point for the POP server module.

Functions

- **int_t pop_compare** (const void *compare, const void *command)
- *commands.c* void **pop_process** (connection_t *con)
- void **pop_requeue** (connection_t *con)
- void **pop_sort** (void)
- *Sort the POP3 command table to be ready for binary searches.* uint64_t **pop_get_last** (inx_t *messages)
- *mailbox.c* meta_message_t * **pop_get_message** (inx_t *messages, uint64_t get)
- *Get a message by its pop sequence number.* uint64_t **pop_total_messages** (inx_t *messages)
- *Get the number of messages available to a POP3 user.* uint64_t **pop_total_size** (inx_t *messages)
- *Get the total size of all messages available to a POP3 user.* bool_t **pop_num_parse** (connection_t *con, uint64_t *outnum, bool_t required)
- *parse.c* stringer_t * **pop_pass_parse** (connection_t *con)
- *Parse a POP3 PASS command.* bool_t **pop_top_parse** (connection_t *con, uint64_t *number, uint64_t *lines)
- *Parse the arguments to a POP3 TOP command.* stringer_t * **pop_user_parse** (connection_t *con)
- *Parse a POP3 USER command.* void **pop_capa** (connection_t *con)
- *pop.c* void **pop_dele** (connection_t *con)
- *Get a message, in response to a POP3 DELE command.* void **pop_init** (connection_t *con)
- *Initialize a new POP3 connection.* void **pop_invalid** (connection_t *con)
- *A function handler for invalid POP3 commands.* void **pop_last** (connection_t *con)
- *Get the sequence number of the last read message, in response to a POP3 LAST command.* void **pop_list** (connection_t *con)
- *Get the list of a user's messages, in response to a POP3 LIST command.* void **pop_noop** (connection_t *con)
- *Execute a POP3 no-operation command.* void **pop_pass** (connection_t *con)
- *Accept and verify a password for POP3 authentication.* void **pop_quit** (connection_t *con)
- *Gracefully destroy a POP3 session, whether because of an error or in response to a user QUIT command.* void **pop_retr** (connection_t *con)
- *Retrieve a user's message, in response to a POP3 RETR command.* void **pop_rset** (connection_t *con)
- *Reset the user's mailbox, in response to a POP3 RSET command.* void **pop_starttls** (connection_t *con)
- *TODO: Review error messages and update them with the appropriate response code.* void **pop_stat** (connection_t *con)
- *Display a user's message statistics, in response to a POP3 STAT command.* void **pop_top** (connection_t *con)
- *Get the top lines of a message or collection of messages, in response to a POP3 TOP command.* void **pop_uidl** (connection_t *con)
- *Get the UIDL for a message or collection of messages, in response to a POP3 UIDL command.* void **pop_user** (connection_t *con)
- *Accept a username for POP3 authentication.* void **pop_session_destroy** (connection_t *con)
- *sessions.c* int_t **pop_session_reset** (connection_t *con)

Reset a POP3 session by guaranteeing that no messages are flagged to be deleted.

Detailed Description

The entry point for the POP server module.

Definition in file **pop.h**.

Function Documentation

void pop_capa (connection_t * *con*)

pop.c

pop.c

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 303 of file pop.c.

References build_version(), con_print(), con_secure(), and connection_t::pop.

int_t pop_compare (const void * *compare*, const void * *command*)

commands.c

Definition at line 16 of file commands.c.

References command_t::length, PLACER, st_cmp_ci_eq(), st_cmp_ci_starts(), and command_t::string.

Referenced by pop_process(), and pop_sort().

void pop_dele (connection_t * *con*)

Get a message, in response to a POP3 DELE command.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 439 of file pop.c.

References con_write_bl(), MAIL_STATUS_HIDDEN, meta_user_unlock(), meta_user_wlock(), number, connection_t::pop, pop_get_message(), pop_invalid(), pop_num_parse(), and meta_message_t::status.

uint64_t pop_get_last (inx_t * *messages*)

mailbox.c

mailbox.c

Note:

This function only counts messages that aren't deleted or hidden, and weren't created by the IMAP APPEND command.

Parameters:

messages an inx holder containing a collection of messages to be analyzed.

Returns:

the sequence number of the last message that isn't flagged as recent, or 0 on failure.

Definition at line 79 of file mailbox.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_APPENDED`, `MAIL_STATUS_HIDDEN`, `MAIL_STATUS_RECENT`, and `meta_message_t::status`.

Referenced by `pop_last()`.

`meta_message_t* pop_get_message (inx_t * messages, uint64_t get)`

Get a message by its pop sequence number.

Parameters:

messages an `inx` holder containing the collection of the user's messages to be traversed.

number the zero-based pop sequence number of the message to be retrieved.

Returns:

NULL on failure or the meta message object of the message if it was found.

Definition at line 118 of file mailbox.c.

References `count`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_APPENDED`, and `meta_message_t::status`.

Referenced by `pop_delete()`, `pop_list()`, `pop_retr()`, `pop_top()`, and `pop_uidl()`.

`void pop_init (connection_t * con)`

Initialize a new POP3 connection.

Parameters:

con the newly connected POP3 client connection.

Returns:

This function returns no value.

Definition at line 684 of file pop.c.

References `con_write_bl()`, and `pop_requeue()`.

Referenced by `protocol_enqueue()`.

`void pop_invalid (connection_t * con)`

A function handler for invalid POP3 commands.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 72 of file pop.c.

References `con_write_bl()`, `server_t::delay`, `connection_t::protocol`, `connection_t::server`, `server_t::violations`, and `connection_t::violations`.

Referenced by `pop_delete()`, `pop_last()`, `pop_list()`, `pop_pass()`, `pop_process()`, `pop_retr()`, `pop_rset()`, `pop_starttls()`, `pop_stat()`, `pop_top()`, `pop_uidl()`, and `pop_user()`.

void pop_last (connection_t * con)

Get the sequence number of the last read message, in response to a POP3 LAST command.

See also:

pop_get_last()

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 351 of file pop.c.

References `con_print()`, `meta_user_rlock()`, `meta_user_unlock()`, `number`, `connection_t::pop`, `pop_get_last()`, and `pop_invalid()`.

void pop_list (connection_t * con)

Get the list of a user's messages, in response to a POP3 LIST command.

See also:

pop_total_messages(), **pop_get_message()**

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 376 of file pop.c.

References `con_print()`, `con_write_bl()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_APPENDED`, `MAIL_STATUS_HIDDEN`, `meta_user_rlock()`, `meta_user_unlock()`, `number`, `connection_t::pop`, `pop_get_message()`, `pop_invalid()`, `pop_num_parse()`, `pop_total_messages()`, `meta_message_t::size`, and `meta_message_t::status`.

void pop_noop (connection_t * con)

Execute a POP3 no-operation command.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 61 of file pop.c.

References `con_write_bl()`.

bool_t pop_num_parse (connection_t * con, uint64_t * outnum, bool_t required)

parse.c

parse.c

Parameters:

con the connection across which the pop command was issued.

outnum a pointer to an unsigned 64-bit integer to receive the numerical argument of the pop command on success.

required specifies whether or not a parameter is required.

Returns:

true if the pop command contained a valid unsigned number or false on failure.

Definition at line 22 of file parse.c.

References `length`, `connection_t::line`, `log_pedantic`, `connection_t::network`, `st_data_get()`, `st_length_get()`, and `uint64_conv_bl()`.

Referenced by `pop_dele()`, `pop_list()`, `pop_retr()`, and `pop_uidl()`.

void pop_pass (connection_t * con)

Accept and verify a password for POP3 authentication.

Note:

This command is only allowed for sessions which have not yet been authenticated, but which have already supplied a username. If the username/password combo was validated, the account information is retrieved and checked to see if it is locked. After successful authentication, this function will prohibit insecure connections for any user configured to use SSL only, and enforce the existence of only one POP3 session at a time. Finally, the database Log table for this user's POP3 access is updated, and all the user's messages are retrieved.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 175 of file pop.c.

References `credential_t::auth`, `con_addr_presentation()`, `con_secure()`, `con_write_bl()`, `credential_alloc_auth()`, `credential_free()`, `credential_username()`, `inx_count()`, `log_pedantic`, `MANAGEDBUF`, `meta_data_update_log()`, `meta_get()`, `META_GET_MESSAGES`, `META_LOCKED`, `meta_messages_login_update()`, `META_PROT_POP`, `meta_remove()`, `META_USER_SSL`, `meta_user_unlock()`, `meta_user_wlock()`, `connection_t::pop`, `pop_invalid()`, `pop_pass_parse()`, `st_char_get()`, `st_empty()`, `st_free()`, `st_length_int()`, and `st_wipe()`.

stringer_t* pop_pass_parse (connection_t * con)

Parse a POP3 PASS command.

Note:

This function will stop reading in password characters when an invalid character is encountered.

Parameters:

con the POP3 client connection that issued the PASS command.

Returns:

a managed string containing the user supplied password, or NULL on failure.

Definition at line 196 of file parse.c.

References `length`, `connection_t::line`, `log_pedantic`, `connection_t::network`, `st_data_get()`, `st_import()`, and `st_length_get()`.

Referenced by `pop_pass()`.

void pop_process (connection_t * con)

Definition at line 48 of file commands.c.

References `connection_t::command`, `command`, `con_read_line()`, `server_t::cutoff`, `enqueue()`, `command_t::function`, `command_t::length`, `connection_t::line`, `connection_t::network`, `pl_char_get()`, `pl_empty()`, `pl_length_get()`, `connection_t::pop`, `pop_commands`, `pop_compare()`, `pop_invalid()`, `pop_process()`, `pop_quit()`, `pop_requeue()`, `connection_t::protocol`, `requeue()`, `connection_t::server`, `connection_t::spins`, `command_t::string`, `server_t::violations`, and `connection_t::violations`.

Referenced by `pop_process()`, and `pop_requeue()`.

void pop_quit (connection_t * con)

Gracefully destroy a POP3 session, whether because of an error or in response to a user QUIT command.

Parameters:

con the POP3 client connection to be shut down. This function returns no value.

Definition at line 112 of file pop.c.

References `con_destroy()`, `con_status()`, and `con_write_bl()`.

Referenced by `pop_process()`, and `pop_requeue()`.

void pop_requeue (connection_t * con)

Definition at line 36 of file commands.c.

References `con_status()`, `server_t::cutoff`, `enqueue()`, `pop_process()`, `pop_quit()`, `connection_t::protocol`, `connection_t::server`, `status`, `server_t::violations`, and `connection_t::violations`.

Referenced by `pop_init()`, and `pop_process()`.

void pop_retr (connection_t * con)

Retrieve a user's message, in response to a POP3 RETR command.

Note:

This function will fail if a deleted message was specified by the user.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 614 of file pop.c.

References `con_print()`, `con_write_bl()`, `con_write_st()`, `mail_destroy()`, `mail_load_message()`, `MAIL_STATUS_HIDDEN`, `meta_user_rlock()`, `meta_user_unlock()`, `number`, `PLACER`, `connection_t::pop`, `pop_get_message()`, `pop_invalid()`, `pop_num_parse()`, `connection_t::server`, `st_char_get()`, `st_length_get()`, `st_replace()`, `meta_message_t::status`, and `mail_message_t::text`.

void pop_rset (connection_t * con)

Reset the user's mailbox, in response to a POP3 RSET command.

See also:

`pop_session_reset()`

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 87 of file `pop.c`.

References `con_write_bl()`, `connection_t::pop`, `pop_invalid()`, and `pop_session_reset()`.

void pop_session_destroy (connection_t * con)

`sessions.c`

`sessions.c`

Parameters:

con the POP3 client connection issuing the command. This function returns no value.

Definition at line 55 of file `sessions.c`.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_delete()`, `M_TYPE_UINT64`, `mail_cache_reset()`, `mail_remove_message()`, `MAIL_STATUS_HIDDEN`, `meta_message_t::messagenum`, `meta_messages_update_sequences()`, `META_PROT_POP`, `meta_remove()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_MESSAGES`, `connection_t::pop`, `serial_increment()`, `meta_message_t::server`, `meta_message_t::size`, `st_cleanup()`, `meta_message_t::status`, `multi_t::u64`, and `multi_t::val`.

Referenced by `con_destroy()`.

int_t pop_session_reset (connection_t * con)

Reset a POP3 session by guaranteeing that no messages are flagged to be deleted.

Parameters:

con the POP3 client connection issuing the command.

Returns:

-1 on general error, 0 if the connection hasn't been authenticated, or 1 if all messages have had their statuses successfully reset.

Definition at line 20 of file `sessions.c`.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_HIDDEN`, `meta_user_unlock()`, `meta_user_wlock()`, `connection_t::pop`, and `meta_message_t::status`.

Referenced by `pop_rset()`, and `pop_starttls()`.

void pop_sort (void)

Sort the POP3 command table to be ready for binary searches.

Returns:

This function returns no value.

Definition at line 31 of file commands.c.

References pop_commands, and pop_compare().

Referenced by protocol_init().

void pop_starttls (connection_t * con)

TODO: Review error messages and update them with the appropriate response code.

Initialize a TLS session for an unauthenticated POP3 session.

Note:

RFC 2595 / section 4 dictates that the STLS/STARTTLS command should only be available in the authorization state.

Parameters:

con the connection of the POP3 client requesting the transport layer security upgrade.

Returns:

This function returns no value (all error messages are written directly to the requesting client).

Definition at line 23 of file pop.c.

References connection_t::buffer, con_secure(), con_write_bl(), connection_t::line, log_pedantic, M_SSL_BIO_NOCLOSE, connection_t::network, pl_null(), connection_t::pop, pop_invalid(), pop_session_reset(), connection_t::server, connection_t::sockd, connection_t::ssl, ssl_alloc(), st_length_set(), stats_increment_by_name(), and connection_t::status.

void pop_stat (connection_t * con)

Display a user's message statistics, in response to a POP3 STAT command.

See also:

pop_total_messages(), pop_total_size()

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 325 of file pop.c.

References con_print(), count, meta_user_rlock(), meta_user_unlock(), connection_t::pop, pop_invalid(), pop_total_messages(), and pop_total_size().

void pop_top (connection_t * con)

Get the top lines of a message or collection of messages, in response to a POP3 TOP command.

Note:

This function will fail if a deleted message was specified by the user.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 543 of file pop.c.

References `con_print()`, `con_write_bl()`, `con_write_st()`, `mail_destroy()`, `mail_load_message_top()`, `MAIL_STATUS_HIDDEN`, `meta_user_rlock()`, `meta_user_unlock()`, `number`, `PLACER`, `connection_t::pop`, `pop_get_message()`, `pop_invalid()`, `pop_top_parse()`, `connection_t::server`, `st_char_get()`, `st_length_get()`, `st_replace()`, `meta_message_t::status`, and `mail_message_t::text`.

bool_t pop_top_parse (connection_t * *con*, uint64_t * *number*, uint64_t * *lines*)

Parse the arguments to a POP3 TOP command.

Parameters:

con the POP3 client connection that issued the TOP command.

number a pointer to an unsigned 64 bit integer to receive the value of the TOP command's message sequence number.

lines a pointer to an unsigned 64 bit integer to receive the value of the TOP command's line number count.

Returns:

true on success or false if an error was encountered.

Definition at line 94 of file parse.c.

References `length`, `connection_t::line`, `log_pedantic`, `connection_t::network`, `st_data_get()`, `st_length_get()`, and `uint64_conv_bl()`.

Referenced by `pop_top()`.

uint64_t pop_total_messages (inx_t * *messages*)

Get the number of messages available to a POP3 user.

Note:

This function only counts messages that aren't deleted or hidden, and weren't created by the IMAP APPEND command.

Parameters:

messages an inx holder containing a collection of messages to be analyzed.

Returns:

the total number of messages available to the POP3 user, or 0 on failure.

Definition at line 21 of file mailbox.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_APPENDED`, `MAIL_STATUS_HIDDEN`, and `meta_message_t::status`.

Referenced by `pop_list()`, `pop_stat()`, and `pop_uidl()`.

uint64_t pop_total_size (inx_t * messages)

Get the total size of all messages available to a POP3 user.

Note:

This function only counts messages that aren't deleted or hidden, and weren't created by the IMAP APPEND command.

Parameters:

messages an inx holder containing a collection of messages to be analyzed.

Returns:

the total size, in bytes, of all messages available to the POP3 user, or 0 on failure.

Definition at line 50 of file mailbox.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_APPENDED`, `MAIL_STATUS_HIDDEN`, `meta_message_t::size`, and `meta_message_t::status`.

Referenced by `pop_stat()`.

void pop_uidl (connection_t * con)

Get the UIDL for a message or collection of messages, in response to a POP3 UIDL command.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 478 of file pop.c.

References `con_print()`, `con_write_bl()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_APPENDED`, `MAIL_STATUS_HIDDEN`, `meta_message_t::messagenum`, `meta_user_rlock()`, `meta_user_unlock()`, `number`, `connection_t::pop`, `pop_get_message()`, `pop_invalid()`, `pop_num_parse()`, `pop_total_messages()`, and `meta_message_t::status`.

void pop_user (connection_t * con)

Accept a username for POP3 authentication.

Note:

This command is only allowed for sessions which have not yet been authenticated. If the username has already been supplied pre-authentication, the old value will be overwritten with the new one.

Parameters:

con the POP3 client connection issuing the command. This function returns no value.

Definition at line 136 of file pop.c.

References `con_write_bl()`, `credential_address()`, `connection_t::pop`, `pop_invalid()`, `pop_user_parse()`, `st_cleanup()`, and `st_free()`.

stringer_t* pop_user_parse (connection_t * con)

Parse a POP3 USER command.

Note:

The username can be at most 255 characters and will be returned in lowercase. This function will stop reading in username characters when an invalid character is encountered.

Parameters:

con the POP3 client connection that issued the USER command.

Returns:

a managed string containing the validated username, or NULL on failure.

Definition at line 257 of file parse.c.

References `length`, `connection_t::line`, `log_pedantic`, `lower_chr()`, `connection_t::network`, `st_data_get()`, `st_import()`, and `st_length_get()`.

Referenced by `pop_user()`.

magma/network/read.c File Reference

Functions used to read in data via the network.

```
#include "magma.h"
```

Functions

- `int64_t con_read_line (connection_t *con, bool_t block)`
- *Read a line of input from a network connection.* `int64_t con_read (connection_t *con)`
- *Read data from a connection, and store it in its internal buffer.* `int64_t client_read_line (client_t *client)`
- *Read a line of input from a network client session.* `int64_t client_read (client_t *client)`

read.c

Detailed Description

Functions used to read in data via the network.

Definition in file **read.c**.

Function Documentation

`int64_t client_read (client_t * client)`

read.c

Definition at line 308 of file `read.c`.

References `client_t::buffer`, `bytes`, `client_t::line`, `mm_move()`, `pl_length_get()`, `pl_null()`, `client_t::sockd`, `client_t::ssl`, `SSL_get_error_d`, `ssl_read()`, `st_avail_get()`, `st_char_get()`, `st_data_get()`, `st_length_get()`, `st_length_set()`, `status`, and `client_t::status`.

`int64_t client_read_line (client_t * client)`

Read a line of input from a network client session.

Returns:

Definition at line 220 of file `read.c`.

References `client_t::buffer`, `bytes`, `client_t::line`, `line_pl_st()`, `mm_move()`, `pl_empty()`, `pl_length_get()`, `pl_null()`, `client_t::sockd`, `client_t::ssl`, `SSL_get_error_d`, `ssl_read()`, `st_avail_get()`, `st_char_get()`, `st_data_get()`, `st_empty()`, `st_length_get()`, `st_length_set()`, `status`, and `client_t::status`.

Referenced by `smtp_client_close()`, `smtp_client_connect()`, `smtp_client_send_data()`, `smtp_client_send_helo()`, `smtp_client_send_mailfrom()`, `smtp_client_send_nullfrom()`, and `smtp_client_send_rcptto()`.

`int64_t con_read (connection_t * con)`

Read data from a connection, and store it in its internal buffer.

Note:

This function handles reading data from both regular and ssl connections. If the connection's network buffer hasn't been allocated, it will be initialized.

Parameters:

con a pointer to the connection object from which the data will be read.

Returns:

-1 on internal error, or on a read error.

Definition at line 128 of file read.c.

References `connection_t::buffer`, `bytes`, `con_init_network_buffer()`, `connection_t::line`, `mm_move()`, `connection_t::network`, `pl_length_get()`, `pl_null()`, `connection_t::sockd`, `connection_t::ssl`, `ssl_read()`, `ssl_shutdown_get()`, `st_avail_get()`, `st_char_get()`, `st_data_get()`, `st_length_get()`, `st_length_set()`, `status`, and `connection_t::status`.

Referenced by `http_body()`, `imap_parse_literal()`, `smtp_data_finish()`, and `smtp_data_read()`.

int64_t con_read_line (connection_t * *con*, bool_t *block*)

Read a line of input from a network connection.

Note:

This function handles reading data from both regular and ssl connections. This function continually attempts to read incoming data from the specified connection until a terminated line of input is received. If a new line is read, the length of that line is returned to the caller, including the trailing

. If the read returns -1 and wasn't caused by a syscall interruption or blocking error, -1 is returned, and the connection status is set to -1. If the read returns 0 and wasn't caused by a syscall interruption or blocking error, -2 is returned, and the connection status is set to 2. Once a character is reached, the length of the current line of input is returned to the user, and the connection status is set to 1.

Parameters:

con the network connection across which the line of data will be read.

Returns:

-1 on general failure, -2 if the connection was reset, or the length of the current line of input, including the trailing

.

Definition at line 27 of file read.c.

References `connection_t::buffer`, `bytes`, `con_init_network_buffer()`, `connection_t::line`, `line_pl_st()`, `mm_move()`, `connection_t::network`, `pl_empty()`, `pl_length_get()`, `pl_null()`, `connection_t::sockd`, `connection_t::ssl`, `ssl_read()`, `st_avail_get()`, `st_char_get()`, `st_data_get()`, `st_empty()`, `st_length_get()`, `st_length_set()`, `status`, and `connection_t::status`.

Referenced by `http_process()`, `imap_parse_literal()`, `imap_process()`, `molten_parse()`, `pop_process()`, `smtp_auth_login()`, `smtp_auth_plain()`, and `smtp_process()`.

magma/network/reverse.c File Reference

Function to perform reverse DNS lookups.

```
#include "magma.h"
```

Functions

- void **con_reverse_enqueue** (**connection_t** *con)
- *Queue a reverse DNS lookup on the specified connection, if one hasn't been performed.* void **con_reverse_domain** (**connection_t** *con, **stringer_t** *domain, **int_t** status)
- *Set the domain name and reverse lookup status of a connection.* void **con_reverse_status** (**connection_t** *con, **int_t** status)
- **stringer_t** * **con_reverse_check** (**connection_t** *con, **uint32_t** timeout)
- *Poll a connection periodically to see if a reverse DNS lookup operation has completed.* void **con_reverse_lookup** (**connection_t** *con)

Perform a reverse DNS lookup on the remote end of a connection, and save the hostname.

Detailed Description

Function to perform reverse DNS lookups.

Definition in file **reverse.c**.

Function Documentation

stringer_t * **con_reverse_check** (**connection_t** * con, **uint32_t** timeout)

Poll a connection periodically to see if a reverse DNS lookup operation has completed.

reverse.c

Note:

Polling will occur every 100ms, for as many seconds as specified by the user.

Parameters:

con the specified connection object to be polled that is the target of the DNS lookup.

timeout the number of seconds to wait for the lookup operation to complete.

Returns:

NULL on failure, or a pointer to a managed string containing the connection's peer hostname on success.

Definition at line 80 of file reverse.c.

References `connection_t::lock`, `mutex_lock()`, `mutex_unlock()`, `connection_t::network`, `connection_t::reverse`, `REVERSE_COMPLETE`, `REVERSE_PENDING`, and `status`.

Referenced by `mail_add_inbound_headers()`, and `mail_add_outbound_headers()`.

void **con_reverse_domain** (**connection_t** * con, **stringer_t** * domain, **int_t** status)

Set the domain name and reverse lookup status of a connection.

Note:

Possible values for status include REVERSE_ERROR, REVERSE_EMPTY, REVERSE_PENDING, and REVERSE_COMPLETE.

Parameters:

domain the new value of the connection's hostname.

status the new value of the connection's reverse lookup status.

Returns:

This function returns no value.

Definition at line 47 of file reverse.c.

References connection_t::lock, mutex_lock(), mutex_unlock(), connection_t::network, and connection_t::reverse.

Referenced by con_reverse_lookup().

void con_reverse_enqueue (connection_t * con)

Queue a reverse DNS lookup on the specified connection, if one hasn't been performed.

Parameters:

con the connection object to be examined.

Returns:

This function returns no value.

Definition at line 20 of file reverse.c.

References con_reverse_lookup(), enqueue(), connection_t::lock, mutex_lock(), mutex_unlock(), connection_t::network, connection_t::refs, connection_t::reverse, REVERSE_EMPTY, and REVERSE_PENDING.

Referenced by http_init(), imap_init(), and smtp_init().

void con_reverse_lookup (connection_t * con)

Perform a reverse DNS lookup on the remote end of a connection, and save the hostname.

Parameters:

con the connection object to be queried.

Returns:

This function returns no value.

Definition at line 112 of file reverse.c.

References buflen, bufptr, con_destroy(), con_reverse_domain(), con_reverse_status(), connection_t::network, ns_length_get(), REVERSE_COMPLETE, REVERSE_ERROR, connection_t::sockd, and st_import().

Referenced by con_reverse_enqueue().

void con_reverse_status (connection_t * con, int_t status)

Definition at line 64 of file reverse.c.

References connection_t::lock, mutex_lock(), mutex_unlock(), connection_t::network, and connection_t::reverse.

Referenced by `con_reverse_lookup()`.

magma/network/sessions.h File Reference

Structures for handling web session data.

Data Structures

- struct **attachment_t**
- struct **composition_t**
- struct **session_t**

Enumerations

- enum { **SESSION_STATE_TERMINATED** = -1, **SESSION_STATE_NEUTRAL** = 0, **SESSION_STATE_AUTHENTICATED** = 1 }

Detailed Description

Structures for handling web session data.

Definition in file **sessions.h**.

Enumeration Type Documentation

anonymous enum

Enumerator:

SESSION_STATE_TERMINATED
SESSION_STATE_NEUTRAL
SESSION_STATE_AUTHENTICATED

Definition at line 16 of file sessions.h.

magma/objects/sessions/sessions.h File Reference

Functions for handling web sessions.

Functions

- **session_t * sess_create** (connection_t *con, stringer_t *path, stringer_t *application)
- *sessions.c* void **sess_destroy** (session_t *sess)
- *Destroy a web session and its associated data.* int_t **sess_get** (connection_t *con, stringer_t *application, stringer_t *path, stringer_t *token)
- *Try to retrieve the session associated with a client connection's supplied cookie.* uint64_t **sess_key** (void)
- *Generate a random 12 digit 64-bit web session key.* uint64_t **sess_number** (void)
- *Reserve a unique web session identifier.* void **sess_ref_add** (session_t *sess)
- *Increment the web session's reference counter and update its timestamp.* void **sess_ref_dec** (session_t *sess)
- *Decrement the web session's reference counter and update its timestamp.* time_t **sess_ref_stamp** (session_t *sess)
- *Get the timestamp for a web session's reference counter.* uint64_t **sess_ref_total** (session_t *sess)
- *Get the reference count of a web session.* bool_t **sess_refresh_check** (session_t *sess)
- *Check to see if a web session is ready to be refreshed, and if so, disarm its trigger and update its refresh stamp.* void **sess_refresh_flush** (session_t *sess)
- *Update a web session's refresh stamp and prevent redundant refreshing of a web session.* time_t **sess_refresh_stamp** (session_t *sess)
- *Get the timestamp for the last time the session was refreshed.* void **sess_release** (session_t *sess)
- *Release a web session.* void **sess_serial_check** (session_t *sess, uint64_t object)
- *Queue a web session refresh if there is stale data.* stringer_t * **sess_token** (session_t *sess)
- *Securely generate a unique zbase32-encoded token for a session.* void **sess_trigger** (session_t *sess)
- *Set a web session's trigger so it will be refreshed as soon as possible.* void **sess_update** (session_t *sess)
- *Update a session's underlying data and its refresh timestamp.* void **sess_release_attachment** (attachment_t *attachment)
- *Free an attachment object.* void **sess_release_composition** (composition_t *comp)

Free a composition object.

Detailed Description

Functions for handling web sessions.

Definition in file **sessions.h**.

Function Documentation

session_t* sess_create (connection_t * con, stringer_t * path, stringer_t * application)

sessions.c

sessions.c

Note:

The session stores the following data points: remote IP address, request path, application name, the specified http hostname, the remote client's user agent string, the server's host number, a unique session id,

the server's current timestamp, a randomly- generated session key for authentication, and an encrypted token for the session returned to the user as a cookie.

Parameters:

con a pointer to the connection underlying the web session.

path a pointer to a managed string containing the pathname of the generating request (should be "/portal/camel").

application a pointer to a managed string containing the name of the parent application of the session (should be "portal").

Returns:

NULL on failure or a pointer to a newly allocated session object for the specified connection.

Definition at line 384 of file sessions.c.

References session_t::agent, session_t::application, session_t::compositions, con_addr(), con_secure(), CONTIGUOUS, HEAP, magma_t::host, session_t::host, connection_t::http, session_t::httponly, inx_alloc(), inx_insert(), session_t::ip, ip_copy(), session_t::key, session_t::lock, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, magma, MANAGED_T, MEMORYBUF, mm_alloc(), mm_free(), session_t::number, magma_t::number, objects, session_t::path, session_t::request, session_t::secure, sess_destroy(), sess_key(), sess_number(), sess_ref_add(), sess_ref_dec(), sess_release_composition(), sess_token(), object_cache_t::sessions, st_dupe_opts(), session_t::stamp, session_t::token, multi_t::u64, multi_t::val, and session_t::warden.

Referenced by portal_endpoint().

void sess_destroy(session_t * sess)

Destroy a web session and its associated data.

Parameters:

sess a pointer to the web session to be destroyed.

Returns:

This function returns no value.

Definition at line 64 of file sessions.c.

References session_t::agent, session_t::application, session_t::compositions, session_t::cred, credential_free(), session_t::host, inx_cleanup(), session_t::lock, META_PROT_WEB, meta_remove(), mm_free(), mutex_destroy(), session_t::path, session_t::request, st_cleanup(), session_t::token, session_t::user, meta_user_t::username, and session_t::warden.

Referenced by obj_cache_start(), and sess_create().

int_t sess_get(connection_t * con, stringer_t * application, stringer_t * path, stringer_t * token)

Try to retrieve the session associated with a client connection's supplied cookie.

Parameters:

con a pointer to the connection object sending the cookie.

application a managed string containing the application associated with the session.

path a managed string containing the path associated with the session.

token the encrypted user token retrieved from the supplied http cookie.

Returns:

1 if the cookie was found and valid, or one of the following values on failure: 0 = Session not found. -1 = Server error. -2 = Invalid token. -3 = Security violation / incorrect user-agent. -4 = Security violation / incorrect session key. -5 = Security violation / incorrect source address. -6 = Session terminated by logout. -7 = Session timed out.

Most session attributes need simple equality comparison, except for timeout checking. Make sure not to validate against a stale session that should have already timed out (which will have to be determined dynamically).

Definition at line 446 of file sessions.c.

References `con_addr()`, `con_secure()`, `magma_t::http`, `connection_t::http`, `inx_delete()`, `inx_find()`, `inx_lock_read()`, `inx_unlock()`, `ip_address_equal()`, `log_error`, `log_pedantic`, `M_TYPE_UINT64`, `magma`, `MEMORYBUF`, `objects`, `scramble_decrypt()`, `scramble_import()`, `magma_t::secure`, `sess_ref_add()`, `sess_ref_dec()`, `sess_refresh_stamp()`, `sess_update()`, `object_cache_t::sessions`, `magma_t::sessions`, `st_char_get()`, `st_cleanup()`, `st_cmp_cs_eq()`, `st_data_get()`, `st_free()`, `multi_t::u64`, `multi_t::val`, and `zbase32_decode()`.

Referenced by `http_parse_context()`.

uint64_t sess_key (void)

Generate a random 12 digit 64-bit web session key.

Returns:

the randomly generated web session key as an unsigned 64 bit integer.

Definition at line 27 of file sessions.c.

References `log_pedantic`, `rand_get_uint64()`, and `uint64_digits()`.

Referenced by `sess_create()`.

uint64_t sess_number (void)

Reserve a unique web session identifier.

Returns:

a number containing a unique web session identifier.

Definition at line 48 of file sessions.c.

References `mutex_lock()`, `mutex_unlock()`, and `sessions`.

Referenced by `sess_create()`.

void sess_ref_add (session_t * sess)

Increment the web session's reference counter and update its timestamp.

Parameters:

sess a pointer to the web session to be updated.

Returns:

This function returns no value.

Definition at line 98 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refs, session_t::stamp, and session_t::web.

Referenced by sess_create(), and sess_get().

void sess_ref_dec (session_t * sess)

Decrement the web session's reference counter and update its timestamp.

Parameters:

sess a pointer to the web session to be updated.

Returns:

This function returns no value.

Definition at line 123 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refs, session_t::stamp, and session_t::web.

Referenced by sess_create(), sess_get(), and sess_release().

time_t sess_ref_stamp (session_t * sess)

Get the timestamp for a web session's reference counter.

Parameters:

sess a pointer to the web session to be queried.

Returns:

the last-modified UTC timestamp value of the specified web session.

Definition at line 173 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refs, and session_t::stamp.

Referenced by obj_cache_prune().

uint64_t sess_ref_total (session_t * sess)

Get the reference count of a web session.

Parameters:

sess a pointer to the web session to be queried.

Returns:

the total number of references to the specified web session.

Definition at line 148 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refs, and session_t::web.

Referenced by obj_cache_prune().

bool_t sess_refresh_check (session_t * sess)

Check to see if a web session is ready to be refreshed, and if so, disarm its trigger and update its refresh stamp.

Note:

The caller needs to make immediate use of this function's return value because the refresh trigger will be cleared if it was previously set. Any session longer than 2 minutes will be marked as ready for refresh.

Parameters:

a pointer to the web session to be queried.

Returns:

true if the web session is ready to be refreshed, or false if it is not.

Definition at line 228 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refresh, session_t::stamp, and session_t::trigger.

Referenced by sess_release().

void sess_refresh_flush (session_t * sess)

Update a web session's refresh stamp and prevent redundant refreshing of a web session.

Parameters:

sess a pointer to the web session to be touched.

Returns:

This function returns no value.

Definition at line 191 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refresh, session_t::stamp, and session_t::trigger.

Referenced by sess_update().

time_t sess_refresh_stamp (session_t * sess)

Get the timestamp for the last time the session was refreshed.

Parameters:

sess a pointer to the web session to be queried.

Returns:

the last-refreshed UTC timestamp value of the specified web session.

Definition at line 208 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refresh, and session_t::stamp.

Referenced by sess_get().

void sess_release (session_t * sess)

Release a web session.

Note:

If the web session should have been refreshed, it will be refreshed before the reference counter is decremented.

Parameters:

sess a pointer to the web session to be released.

Returns:

This function returns no value.

Definition at line 315 of file sessions.c.

References `requeue()`, `sess_ref_dec()`, `sess_refresh_check()`, and `sess_update()`.

Referenced by `http_session_reset()`.

void sess_release_attachment (attachment_t * *attachment*)

Free an attachment object.

Note:

This is an `inx` helper function.

Parameters:

attachment a pointer to the attachment object to be destroyed.

Returns:

This function returns no value.

Definition at line 334 of file sessions.c.

References `attachment_t::filedata`, `attachment_t::filename`, `mm_free()`, and `st_cleanup()`.

Referenced by `portal_endpoint_attachments_add()`, and `portal_endpoint_messages_compose()`.

void sess_release_composition (composition_t * *comp*)

Free a composition object.

Note:

This is an `inx` helper function.

Parameters:

comp a pointer to the composition object to be destroyed.

Returns:

This function returns no value.

Definition at line 350 of file sessions.c.

References `composition_t::attachments`, `inx_cleanup()`, and `mm_free()`.

Referenced by `portal_endpoint_messages_compose()`, and `sess_create()`.

void sess_serial_check (session_t * *sess*, uint64_t *object*)

Queue a web session refresh if there is stale data.

Parameters:

sess a pointer to the web session to be queried.

object the value of the object in the cache to be checked (can include OBJECT_CONTACTS and OBJECT_FOLDERS).

Definition at line 365 of file sessions.c.

References meta_user_serial_check(), sess_trigger(), and session_t::user.

Referenced by portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(), portal_endpoint_contacts_move(), portal_endpoint_contacts_remove(), portal_folder_contacts_add(), portal_folder_contacts_remove(), portal_folder_mail_add(), and portal_folder_mail_remove().

stringer_t* sess_token (session_t * sess)

Securely generate a unique zbase32-encoded token for a session.

Parameters:

sess a pointer to the input web session.

Returns:

a managed string containing the generated web session token.

Definition at line 251 of file sessions.c.

References session_t::host, session_t::key, log_pedantic, magma, session_t::number, PLACER, scramble_encrypt(), scramble_free(), scramble_total_length(), magma_t::secure, magma_t::sessions, st_cleanup(), st_merge, session_t::stamp, session_t::warden, and zbase32_encode().

Referenced by sess_create().

void sess_trigger (session_t * sess)

Set a web session's trigger so it will be refreshed as soon as possible.

Parameters:

sess a pointer to the web session to be refreshed.

Returns:

This function returns no value.

Definition at line 298 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refresh, and session_t::trigger.

Referenced by portal_endpoint_folders_rename(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(), and sess_serial_check().

void sess_update (session_t * sess)

Update a session's underlying data and its refresh timestamp.

Note:

This function ensures the session's associated meta user data, mail folders and messages, and contact folders and contact entries are all current.

Parameters:

sess a pointer to the session object to be updated.

Returns:

This function returns no value.

Definition at line 278 of file sessions.c.

References `meta_contacts_update()`, `meta_folders_update()`, `meta_messages_update()`, `META_NEED_LOCK`, `meta_user_update()`, `sess_refresh_flush()`, and `session_t::user`.

Referenced by `sess_get()`, and `sess_release()`.

magma/network/smtp.h File Reference

Structures used to control SMTP connections/sessions.

Data Structures

- struct `smtp_message_t`
- struct `smtp_recipients_t`
- struct `smtp_inbound_filter_t`
- struct `smtp_inbound_prefs_t`
- struct `smtp_outbound_prefs_t`
- struct `smtp_session_t`

Enumerations

- enum { `SMTP_ACTION_ERROR` = -1, `SMTP_ACTION_UNDEFINED` = 0, `SMTP_ACTION_DELETE` = 1, `SMTP_ACTION_REJECT` = 2, `SMTP_ACTION_BOUNCE` = 3, `SMTP_ACTION_MARK` = 4, `SMTP_ACTION_MARK_READ` = 5, `SMTP_MARK_NONE` = 0, `SMTP_MARK_READ` = 1, `SMTP_MARK_PHISH` = 2, `SMTP_MARK_VIRUS` = 4, `SMTP_MARK_SPAM` = 8, `SMTP_MARK_RBL` = 16, `SMTP_MARK_SPOOF` = 32, `SMTP_MARK_FILTERED` = 64, `SMTP_FILTER_TYPE_EXACT` = 0, `SMTP_FILTER_TYPE_CONTAINS` = 1, `SMTP_FILTER_TYPE_STARTS` = 2, `SMTP_FILTER_TYPE_ENDS` = 4, `SMTP_FILTER_TYPE_REGEX` = 8, `SMTP_FILTER_LOCATION_HEADER` = 1, `SMTP_FILTER_LOCATION_BODY` = 2, `SMTP_FILTER_LOCATION_FIELD` = 4, `SMTP_FILTER_LOCATION_ENTIRE` = 8, `SMTP_FILTER_ACTION_NONE` = 0, `SMTP_FILTER_ACTION_MOVE` = 1, `SMTP_FILTER_ACTION_LABEL` = 2, `SMTP_FILTER_ACTION_DELETE` = 4, `SMTP_FILTER_ACTION_MARK_READ` = 8, `SMTP_OUTCOME_SUCCESS` = 0, `SMTP_OUTCOME_PERM_FAILURE` = 1, `SMTP_OUTCOME_TEMP_SERVER` = 2, `SMTP_OUTCOME_TEMP_OVERQUOTA` = 4, `SMTP_OUTCOME_TEMP_LOCKED` = 8, `SMTP_OUTCOME_BOUNCE_SPF` = 16, `SMTP_OUTCOME_BOUNCE_DKIM` = 32, `SMTP_OUTCOME_BOUNCE_VIRUS` = 64, `SMTP_OUTCOME_BOUNCE_PHISH` = 128, `SMTP_OUTCOME_BOUNCE_SPAM` = 256, `SMTP_OUTCOME_BOUNCE_RBL` = 512 }

Detailed Description

Structures used to control SMTP connections/sessions.

Definition in file `smtp.h`.

Enumeration Type Documentation

anonymous enum

Enumerator:

SMTP_ACTION_ERROR
SMTP_ACTION_UNDEFINED
SMTP_ACTION_DELETE
SMTP_ACTION_REJECT

SMTP_ACTION_BOUNCE
SMTP_ACTION_MARK
SMTP_ACTION_MARK_READ
SMTP_MARK_NONE
SMTP_MARK_READ
SMTP_MARK_PHISH
SMTP_MARK_VIRUS
SMTP_MARK_SPAM
SMTP_MARK_RBL
SMTP_MARK_SPOOF
SMTP_MARK_FILTERED
SMTP_FILTER_TYPE_EXACT
SMTP_FILTER_TYPE_CONTAINS
SMTP_FILTER_TYPE_STARTS
SMTP_FILTER_TYPE_ENDS
SMTP_FILTER_TYPE_REGEX
SMTP_FILTER_LOCATION_HEADER
SMTP_FILTER_LOCATION_BODY
SMTP_FILTER_LOCATION_FIELD
SMTP_FILTER_LOCATION_ENTIRE
SMTP_FILTER_ACTION_NONE
SMTP_FILTER_ACTION_MOVE
SMTP_FILTER_ACTION_LABEL
SMTP_FILTER_ACTION_DELETE
SMTP_FILTER_ACTION_MARK_READ
SMTP_OUTCOME_SUCESS
SMTP_OUTCOME_PERM_FAILURE
SMTP_OUTCOME_TEMP_SERVER
SMTP_OUTCOME_TEMP_OVERQUOTA
SMTP_OUTCOME_TEMP_LOCKED
SMTP_OUTCOME_BOUNCE_SPF
SMTP_OUTCOME_BOUNCE_DKIM
SMTP_OUTCOME_BOUNCE_VIRUS
SMTP_OUTCOME_BOUNCE_PHISH
SMTP_OUTCOME_BOUNCE_SPAM
SMTP_OUTCOME_BOUNCE_RBL

Definition at line 16 of file smtp.h.

magma/servers/smtp/smtp.h File Reference

The entry point for the SMTP server module.

Functions

- **int_t smtp_accept_message** (**connection_t** *con, **smtp_inbound_prefs_t** *prefs)
- *accept.c* **int_t smtp_rollout** (**smtp_inbound_prefs_t** *prefs)
- *Delete the oldest mail message owned by a user until their storage usage falls below their storage quota.* **int_t smtp_store_message** (**smtp_inbound_prefs_t** *prefs, **stringer_t** **local)
- *Store a received SMTP message as a generic mail message, both on disk and in the database.* **bool_t smtp_store_spamsig** (**smtp_inbound_prefs_t** *prefs, **int_t** spam)
- *Generate a random key for a spam signature and store it in the database.* **int_t smtp_check_filters** (**smtp_inbound_prefs_t** *prefs, **stringer_t** **local)
- *checkers.c* **int_t smtp_check_greylist** (**connection_t** *con, **smtp_inbound_prefs_t** *prefs)
- *Check to see if a transmitting address is in a user's greylist.* **int_t smtp_check_rbl** (**connection_t** *con)
- *Check the SMTP connection's remote address against a collection of real-time blacklists.* **bool_t smtp_add_bypass_entry** (**stringer_t** *subnet)
- *Add an entry to the SMTP subnet bypass list.* **bool_t smtp_bypass_check** (**connection_t** *con)
- *Check if a connection should bypass certain SMTP checks.* **void smtp_requeue** (**connection_t** *con)
- *commands.c* **int_t smtp_compare** (const void *compare, const void *command)
- **void smtp_process** (**connection_t** *con)
- *The main entry point in the smtp server for processing commands issued by clients.* **void smtp_sort** (void)
- *Sort the SMTP command table to be ready for binary searches.* **int_t smtp_check_authorized_from** (uint64_t usernum, **stringer_t** *address)
- *datatier.c* **int_t smtp_check_receive_quota** (**connection_t** *con, **smtp_inbound_prefs_t** *prefs)
- *Check the user's received statistics from the database to see if receiving a message would result in a quota overage.* **int_t smtp_check_transmit_quota** (uint64_t usernum, size_t num_recipients, **smtp_outbound_prefs_t** *prefs)
- *Check to see if a user's current mail send request would push them over their daily transmission quota.* **int_t smtp_fetch_authorization** (**credential_t** *cred, **smtp_outbound_prefs_t** **output)
- *Check if a user is authorized to send messages, and retrieve the user's outbound smtp preferences.* **stringer_t** *smtp_fetch_autoreply (uint64_t autoreply, uint64_t usernum)
- *Fetch a specified auto-reply message for a user.* **int_t smtp_fetch_inbound** (**credential_t** *cred, **stringer_t** *address, **smtp_inbound_prefs_t** **output)
- *Fetch a user's smtp inbound preferences from the database.* **table_t** *smtp_fetch_rollmessages (uint64_t usernum)
- *Retrieve a list, at a maximum of 20 entries, of the oldest messages owned by a user.* **int_t smtp_get_action** (**chr_t** *string, size_t length)
- *Parse an smtp event action string from the Dispatch table.* uint64_t smtp_insert_spamsig (**smtp_inbound_prefs_t** *prefs, uint64_t key, **int_t** code)
- *Store a spam signature in the database.* **void smtp_update_receive_stats** (**connection_t** *con, **smtp_inbound_prefs_t** *prefs)
- *Update the receiving statistics and per-user log tables in the database for a successfully received smtp message.* **void smtp_update_transmission_stats** (**connection_t** *con)
- *Update the transmission and per-user log tables in the database for a successfully sent smtp message.* **void smtp_auth_login** (**connection_t** *con)
- *smtp.c* **void smtp_auth_plain** (**connection_t** *con)
- **void smtp_data** (**connection_t** *con)
- **void smtp_disabled** (**connection_t** *con)
- *A stub function for an SMTP command that has not been implemented.* **void smtp_ehlo** (**connection_t** *con)
- *Process an SMTP EHLO command.* **void smtp_helo** (**connection_t** *con)
- *Process an SMTP HELO command.* **void smtp_init** (**connection_t** *con)

- The start of the protocol handler for the SMTP server. void **smtp_invalid** (**connection_t** *con)
- A function that is executed when an invalid SMTP command is executed. void **smtp_mail_from** (**connection_t** *con)
- Specify the identity of a message's sender, in response to an SMTP MAIL FROM command. void **smtp_noop** (**connection_t** *con)
- Perform an SMTP NOOP (no-operation) command. void **smtp_quit** (**connection_t** *con)
- Gracefully terminate an SMTP session, especially in response to an SMTP QUIT command. void **smtp_rcpt_to** (**connection_t** *con)
- void **smtp_rset** (**connection_t** *con)
- Reset the SMTP session, in response to an SMTP RSET command. void **smtp_starttls** (**connection_t** *con)
- TODO: Review error messages and update them with the appropriate response code. void **submission_init** (**connection_t** *con)
- **stringer_t** * **smtp_parse_auth** (**stringer_t** *data)
- *parse.c* **stringer_t** * **smtp_parse_helo_domain** (**connection_t** *con)
- Parse the domain specified as the parameter to an SMTP HELO or EHLO command. **stringer_t** * **smtp_parse_mail_from_path** (**connection_t** *con)
- **stringer_t** * **smtp_parse_rcpt_to** (**connection_t** *con)
- void **smtp_client_close** (**client_t** *client)
- *relay.c* **client_t** * **smtp_client_connect** (**int_t** premium)
- Connect to a randomly selected mail relay server, and wait for a successful banner message. **int_t** **smtp_client_send_data** (**client_t** *client, **stringer_t** *message)
- Issue a DATA command to an smtp server, and wait for a successful response. **int_t** **smtp_client_send_helo** (**client_t** *client)
- Issue a EHLO command to an smtp server, or fall back to HELO, and wait for a successful response. **int_t** **smtp_client_send_mailfrom** (**client_t** *client, **stringer_t** *mailfrom, **size_t** send_size)
- Issue a MAIL FROM command to an smtp server, and wait for a successful response. **int_t** **smtp_client_send_nullfrom** (**client_t** *client)
- Issue a MAIL FROM command to an smtp server with a null sender, and wait for a successful response. **int_t** **smtp_client_send_rcptto** (**client_t** *client, **stringer_t** *rcptto)
- Issue a RCPT TO command to an smtp server, and wait for a successful response. void **smtp_add_inbound** (**connection_t** *con, **smtp_inbound_prefs_t** *inbound)
- *session.c* void **smtp_add_outbound** (**connection_t** *con, **smtp_outbound_prefs_t** *outbound)
- Attach a set of SMTP outbound mail preferences to an SMTP client connection. **bool_t** **smtp_add_recipient** (**connection_t** *con, **stringer_t** *address)
- Add an entry to an SMTP session's recipients list for outbound/relayed mail. **bool_t** **smtp_check_duplicate_recipient** (**connection_t** *con, **uint64_t** usernum)
- void **smtp_free_inbound** (**smtp_inbound_prefs_t** *inbound)
- Free a list of SMTP inbound mail preferences and its underlying data. void **smtp_free_outbound** (**smtp_outbound_prefs_t** *outbound)
- Free a set of SMTP outbound mail preferences and its underlying data. void **smtp_free_recipients** (**smtp_recipients_t** *recipients)
- Free a list of SMTP recipients and its underlying data. void **smtp_list_free_filter** (**smtp_inbound_filter_t** *filter)
- Free an SMTP inbound filter and its underlying data. void **smtp_session_destroy** (**connection_t** *con)
- Destroy the data associated with an SMTP session. void **smtp_session_reset** (**connection_t** *con)
- Reset an SMTP session to its initialized state. **int_t** **smtp_bounce** (**connection_t** *con)
- *transmit.c* **int_t** **smtp_forward_message** (**server_t** *server, **stringer_t** *address, **stringer_t** *message, **stringer_t** *id, **int_t** mark, **uint64_t** signum, **uint64_t** sigkey)
- **int_t** **smtp_relay_message** (**connection_t** *con, **stringer_t** **result)
- Relay an outbound smtp message for a user. **int_t** **smtp_reply** (**stringer_t** *from, **stringer_t** *to, **uint64_t** usernum, **uint64_t** autoreply, **int_t** spf, **int_t** dkim)
- **int_t** **smtp_send_message** (**stringer_t** *to, **stringer_t** *from, **stringer_t** *message)

Relay an outbound smtp message for the user.

Detailed Description

The entry point for the SMTP server module.

Definition in file **smtp.h**.

Function Documentation

int_t smtp_accept_message (connection_t * con, smtp_inbound_prefs_t * prefs)

accept.c

Definition at line 201 of file accept.c.

References magma_t::abuse, magma_t::admin, smtp_inbound_prefs_t::autoreply, smtp_inbound_prefs_t::bounces, smtp_session_t::bypass, smtp_session_t::checked, magma_t::contact, smtp_session_t::dkim, smtp_inbound_prefs_t::dkim, dkim_check(), smtp_inbound_prefs_t::dkimaction, dspam_check(), smtp_inbound_prefs_t::filters, smtp_inbound_prefs_t::foldernum, smtp_inbound_prefs_t::forwarded, smtp_message_t::id, smtp_inbound_prefs_t::inbox, log_error, log_pedantic, magma, mail_add_inbound_headers(), smtp_session_t::mailfrom, smtp_inbound_prefs_t::mark, smtp_session_t::message, smtp_inbound_prefs_t::overquota, smtp_inbound_prefs_t::phish, smtp_inbound_prefs_t::phishaction, PLACER, smtp_session_t::rbl, smtp_inbound_prefs_t::rbl, smtp_inbound_prefs_t::rblaction, smtp_inbound_prefs_t::rcptto, smtp_inbound_prefs_t::recv_size_limit, smtp_inbound_prefs_t::rollout, connection_t::server, smtp_inbound_prefs_t::signum, connection_t::smtp, SMTP_ACTION_BOUNCE, SMTP_ACTION_DELETE, SMTP_ACTION_MARK, SMTP_ACTION_MARK_READ, smtp_check_filters(), smtp_forward_message(), SMTP_MARK_NONE, SMTP_MARK_PHISH, SMTP_MARK_RBL, SMTP_MARK_READ, SMTP_MARK_SPAM, SMTP_MARK_SPOOF, SMTP_MARK_VIRUS, SMTP_OUTCOME_BOUNCE_DKIM, SMTP_OUTCOME_BOUNCE_PHISH, SMTP_OUTCOME_BOUNCE_RBL, SMTP_OUTCOME_BOUNCE_SPAM, SMTP_OUTCOME_BOUNCE_SPF, SMTP_OUTCOME_BOUNCE_VIRUS, SMTP_OUTCOME_PERM_FAILURE, SMTP_OUTCOME_SUCESS, SMTP_OUTCOME_TEMP_LOCKED, SMTP_OUTCOME_TEMP_OVERQUOTA, SMTP_OUTCOME_TEMP_SERVER, smtp_reply(), smtp_rollout(), smtp_store_message(), smtp_store_spamsig(), smtp_update_receive_stats(), smtp_inbound_prefs_t::spam, smtp_inbound_prefs_t::spam_checked, smtp_inbound_prefs_t::spamaction, smtp_inbound_prefs_t::spamkey, smtp_inbound_prefs_t::spamsig, smtp_session_t::spf, smtp_inbound_prefs_t::spf, smtp_inbound_prefs_t::spfaction, st_cmp_ci_eq(), st_cmp_cs_eq(), st_free(), st_length_get(), smtp_message_t::text, smtp_inbound_prefs_t::usernum, smtp_session_t::virus, smtp_inbound_prefs_t::virus, virus_check(), and smtp_inbound_prefs_t::virusaction.

Referenced by smtp_data_inbound().

bool_t smtp_add_bypass_entry (stringer_t * subnet)

Add an entry to the SMTP subnet bypass list.

Parameters:

subnet a pointer to a managed string containing the IP address or subnet address to be bypassed for checks.

Returns:

true if the entry was valid and was added, or false on failure.

Definition at line 265 of file checkers.c.

References magma_t::bypass_subnets, inx_alloc(), inx_insert(), ip_str_subnet(), log_error, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, magma, mm_alloc(), mm_free(), magma_t::smtp, st_char_get(), multi_t::type, multi_t::u64, and multi_t::val.

Referenced by config_value_set().

void smtp_add_inbound (connection_t * con, smtp_inbound_prefs_t * inbound)

session.c

session.c

Parameters:

con the SMTP client connection specifying the added local recipient.

inbound a pointer to the SMTP inbound preferences object to be added to the SMTP session's inbound preferences list.

Returns:

true if the requested inbound preferences object was added successfully to the inbound preferences list, or false on failure.

Definition at line 157 of file session.c.

References smtp_session_t::in_prefs, smtp_inbound_prefs_t::next, smtp_session_t::num_recipients, and connection_t::smtp.

Referenced by smtp_rcpt_to().

void smtp_add_outbound (connection_t * con, smtp_outbound_prefs_t * outbound)

Attach a set of SMTP outbound mail preferences to an SMTP client connection.

Parameters:

con the SMTP client connection to which the outbound mail preferences should be attached.

outbound a pointer to the SMTP outbound mail preferences to be set for the specified connection.

Returns:

This function returns no value.

Definition at line 186 of file session.c.

References smtp_session_t::out_prefs, and connection_t::smtp.

Referenced by smtp_auth_login(), and smtp_auth_plain().

bool_t smtp_add_recipient (connection_t * con, stringer_t * address)

Add an entry to an SMTP session's recipients list for outbound/relayed mail.

Note:

If the recipients list does not exist, it will be created automatically.

Parameters:

con the SMTP client connection specifying the added recipient.

address a managed string containing the recipient's email address.

Returns:

true if the requested recipient was added successfully to the recipients list, or false on failure.

Definition at line 114 of file session.c.

References smtp_recipients_t::address, log_pedantic, mm_alloc(), mm_free(), smtp_recipients_t::next, smtp_session_t::num_recipients, smtp_session_t::out_prefs, smtp_outbound_prefs_t::recipients, connection_t::smtp, and st_dupe().

Referenced by smtp_rcpt_to().

void smtp_auth_login (connection_t * con)**smtp.c**

BUG: The code should be able to differentiate between invalid usernames which trigger a NULL return and allocation (or similar) errors which are temporary. We approximate this functionality by not rejecting "@domain.com" as a username via the credentials function, but instead check for leading at symbols explicitly below.

Definition at line 360 of file smtp.c.

References credential_t::auth, smtp_session_t::authenticated, base64_decode(), con_read_line(), con_write_bl(), credential_alloc_auth(), credential_free(), connection_t::line, smtp_session_t::max_length, connection_t::network, pl_char_get(), pl_length_get(), PLACER, smtp_outbound_prefs_t::send_size_limit, connection_t::smtp, smtp_add_outbound(), smtp_fetch_authorization(), smtp_parse_auth(), smtp_session_reset(), st_cleanup(), st_cmp_ci_eq(), st_cmp_cs_starts(), st_empty(), and st_free().

void smtp_auth_plain (connection_t * con)

BUG: The SMTP server is not handling AUTH requests in accordance with RFC 2554 <<http://tools.ietf.org/html/rfc2554>> and RFC 4954 <<http://tools.ietf.org/html/rfc4954>>, according to a user.

In this particular case it's erroring out when users present an AUTH PLAIN request without the optional initial response. This could affect other protocols as well.

According to the RFC, a client can send either the following: 1. "AUTH mechanism" (e.g. "AUTH PLAIN"), which the server acknowledges with a "334 " (note the space). The client then sends the base64-encoded authentication response on a separate line.

2. "AUTH mechanism [initial response]" (e.g. "AUTH PLAIN

dGVzdAB0ZXN0ADEyMzQ="), where user transmits the auth mechanism and the base64 authentication response in a single line, so as to minimize the back-and-forth traffic.

BUG: The code should be able to differentiate between invalid usernames which trigger a NULL return and allocation (or similar) errors which are temporary. We approximate this functionality by not rejecting "@domain.com" as a username via the credentials function, but instead check for leading at symbols explicitly below.

Definition at line 246 of file smtp.c.

References credential_t::auth, smtp_session_t::authenticated, base64_decode(), con_read_line(), con_write_bl(), credential_alloc_auth(), credential_free(), FOREIGNDATA, JOINTED, connection_t::line, smtp_session_t::max_length, mm_copy(), connection_t::network, pl_char_get(), pl_length_get(), PLACER, PLACER_T, placer_t, smtp_outbound_prefs_t::send_size_limit, connection_t::smtp, smtp_add_outbound(),

smtp_fetch_authorization(), smtp_parse_auth(), smtp_session_reset(), st_cleanup(), st_cmp_ci_eq(), st_cmp_cs_starts(), st_empty(), st_free(), st_length_get(), STACK, and tok_get_st().

int_t smtp_bounce (connection_t * con)

transmit.c

Definition at line 179 of file transmit.c.

References smtp_session_t::checked, smtp_session_t::dkim, dkim_create(), magma_t::domain, hash_crc64(), smtp_session_t::in_prefs, log_pedantic, magma, smtp_session_t::mailfrom, MANAGEDBUF, smtp_session_t::message, smtp_inbound_prefs_t::next, number, smtp_inbound_prefs_t::outcome, rand_choices(), smtp_inbound_prefs_t::rcptto, connection_t::smtp, smtp_client_close(), smtp_client_connect(), smtp_client_send_data(), smtp_client_send_helo(), smtp_client_send_nullfrom(), smtp_client_send_rcptto(), SMTP_OUTCOME_BOUNCE_DKIM, SMTP_OUTCOME_BOUNCE_PHISH, SMTP_OUTCOME_BOUNCE_RBL, SMTP_OUTCOME_BOUNCE_SPAM, SMTP_OUTCOME_BOUNCE_SPF, SMTP_OUTCOME_BOUNCE_VIRUS, SMTP_OUTCOME_PERM_FAILURE, SMTP_OUTCOME_SUCESS, SMTP_OUTCOME_TEMP_LOCKED, SMTP_OUTCOME_TEMP_OVERQUOTA, SMTP_OUTCOME_TEMP_SERVER, smtp_session_t::spf, st_char_get(), st_cleanup(), st_cmp_cs_eq(), st_free(), st_length_int(), st_merge, st_sprint(), magma_t::system, and smtp_message_t::text.

Referenced by smtp_data_inbound().

bool_t smtp_bypass_check (connection_t * con)

Check if a connection should bypass certain SMTP checks.

Note:

This check is run against host and/or subnet masks configured in the magma.smtp.bypass_addr option.

Parameters:

con a pointer to the connection object to be checked.

Returns:

true if the specified connection meets the SMTP bypass check or false on failure or if it does not.

Definition at line 305 of file checkers.c.

References magma_t::bypass_subnets, con_addr(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), ip_matches_subnet(), magma, and magma_t::smtp.

Referenced by smtp_init().

int_t smtp_check_authorized_from (uint64_t usernum, stringer_t * address)

datatier.c

datatier.c

Note:

The authorization attempt first checks against the specified email address, and if unsuccessful, upon the domain component of the email address if wildcards are enabled for that domain.

Parameters:

usernum the numerical id of the user attempting to send the mail message.

address a pointer to a managed string containing the From address value of the mail message to be sent.

Returns:

1 if the user is authorized to send email from the specified address, or 0 otherwise.

Definition at line 760 of file `datatier.c`.

References `domain_wildcard()`, `mail_domain_get()`, `mm_wipe()`, `placer_t`, `res_row_count()`, `res_table_free()`, `st_char_get()`, `st_length_get()`, and `stmt_get_result()`.

Referenced by `portal_outbound_checks()`, and `smtp_data_outbound()`.

bool_t smtp_check_duplicate_recipient (connection_t * con, uint64_t usernum)

Definition at line 87 of file `session.c`.

References `smtp_session_t::in_prefs`, `smtp_inbound_prefs_t::next`, `connection_t::smtp`, and `smtp_inbound_prefs_t::usernum`.

Referenced by `smtp_rcpt_to()`.

int_t smtp_check_filters (smtp_inbound_prefs_t * prefs, stringer_t ** local)**checkers.c**

Apply any user specific filters. Return -1 for errors, and -2 to delete a message. Return 1 if no action was taken, and 2 if the message was moved to a different folder, 3 if the message content was modified, and 4 if the message was marked read.

Definition at line 144 of file `checkers.c`.

References `data`, `smtp_inbound_prefs_t::filters`, `smtp_inbound_prefs_t::foldernum`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `length`, `log_pedantic`, `mail_header_end()`, `mail_header_fetch_all()`, `mail_mod_subject()`, `smtp_inbound_prefs_t::mark`, `mm_wipe()`, `pl_data_get()`, `pl_init()`, `pl_length_get()`, `pl_null()`, `PLACER`, `placer_t`, `SMTP_FILTER_ACTION_DELETE`, `SMTP_FILTER_ACTION_LABEL`, `SMTP_FILTER_ACTION_MARK_READ`, `SMTP_FILTER_ACTION_MOVE`, `SMTP_FILTER_LOCATION_BODY`, `SMTP_FILTER_LOCATION_ENTIRE`, `SMTP_FILTER_LOCATION_FIELD`, `SMTP_FILTER_LOCATION_HEADER`, `SMTP_MARK_NONE`, `SMTP_MARK_PHISH`, `SMTP_MARK_RBL`, `SMTP_MARK_READ`, `SMTP_MARK_SPAM`, `SMTP_MARK_SPOOF`, `SMTP_MARK_VIRUS`, `st_char_get()`, `st_cleanup()`, `st_cmp_ci_eq()`, `st_length_get()`, `st_length_int()`, and `smtp_inbound_prefs_t::usernum`.

Referenced by `smtp_accept_message()`.

int_t smtp_check_greylist (connection_t * con, smtp_inbound_prefs_t * prefs)

Check to see if a transmitting address is in a user's greylist.

Note:

The greylist is configured in the Dispatch table and specifies the minimum time, in minutes, that a transmitting smtp relay server must wait in order to be able to send more messages to the same recipient address again.

Parameters:

con the connection to have its remote address checked against the user's greylist.

prefs the smtp inbound preferences of the user

Returns:

-1 on general error, -2 if the check failed, and 1 if the check was passed.

Definition at line 27 of file checkers.c.

References BLOCK_T, smtp_session_t::bypass, cache_get(), cache_set(), con_addr_reversed(), CONTIGUOUS, smtp_inbound_prefs_t::greytime, HEAP, imap_fetch_response_t::key, log_pedantic, MANAGEDBUF, connection_t::smtp, st_alloc_opts(), st_char_get(), st_cleanup(), st_data_get(), st_length_get(), st_length_int(), st_sprint(), smtp_inbound_prefs_t::usernum, and imap_fetch_response_t::value.

Referenced by smtp_rcpt_to().

int_t smtp_check_rbl (connection_t * con)

Check the SMTP connection's remote address against a collection of real-time blacklists.

Note:

The connection's IP address will be checked against each of the servers configured in magma.smtp.blacklists.domain.

Parameters:

con the connection to have its address examined against the RBLs.

Returns:

-1 on general error, -2 if the address was blacklisted, or 1 if it passed the check.

Definition at line 95 of file checkers.c.

References magma_t::blacklists, con_addr_reversed(), log_pedantic, magma, MANAGEDBUF, mm_wipe(), magma_t::smtp, st_char_get(), and st_length_int().

Referenced by smtp_rcpt_to().

int_t smtp_check_receive_quota (connection_t * con, smtp_inbound_prefs_t * prefs)

Check the user's received statistics from the database to see if receiving a message would result in a quota overage.

Note:

The sum total of all emails received by the user over the past 24 hours is calculated from the database, and these checks are made: 1. The amount of mail messages received in the past 24 hours by the user does not exceed their daily mail received quota. 2. The messages received in the past 24 hours by the user from this subnet does not exceed the user's daily per-subnet received quota.

Parameters:

con a pointer to the connection object over which the smtp message was received.

prefs a pointer to the user's smtp inbound mail preferences.

Returns:

0 if message receipt is permitted, 1 if the general daily receiving limit was exceeded, 2 if the sending subnet's transmission limit was exceeded, or -1 if there was a general error.

Definition at line 695 of file datatier.c.

References con_addr_subnet(), smtp_inbound_prefs_t::daily_rcv_limit, smtp_inbound_prefs_t::daily_rcv_limit_ip, log_pedantic, mm_wipe(), number, res_field_block(), res_field_length(), res_field_uint64(), res_row_next(), res_table_free(), st_char_get(), st_free(), st_length_get(), stmt_get_result(), uint64_conv_bl(), and smtp_inbound_prefs_t::usernum.

Referenced by smtp_rcpt_to().

**int_t smtp_check_transmit_quota (uint64_t *usernum*, size_t *num_recipients*,
smtp_outbound_prefs_t * *prefs*)**

Check to see if a user's current mail send request would push them over their daily transmission quota.

Note:

This check is performed by querying the database to see how many messages a user has sent in the past 24 hour period, and by adding the current number of recipients of the pending email request to that number to see if their quota would be exceeded.

Parameters:

con a pointer to the connection object of the user attempting to send mail.

Returns:

-1 on error, 0 if the send operation is permitted, or 1 if the send operation would result in a daily send quota overage.

Definition at line 525 of file *datatier.c*.

References *smtp_outbound_prefs_t::daily_send_limit*, *log_pedantic*, *mm_wipe()*, *res_field_uint64()*, *res_row_next()*, *res_table_free()*, *smtp_outbound_prefs_t::sent_today*, and *stmt_get_result()*.

Referenced by *portal_outbound_checks()*, and *smtp_data_outbound()*.

void smtp_client_close (client_t * *client*)

relay.c

relay.c

Parameters:

client a pointer to the smtp client session to be closed.

Returns:

This function returns no value.

Definition at line 20 of file *relay.c*.

References *client_close()*, *client_read_line()*, *client_write()*, and *PLACER*.

Referenced by *portal_smtp_relay_message()*, *smtp_bounce()*, *smtp_forward_message()*, *smtp_relay_message()*, *smtp_reply()*, and *smtp_send_message()*.

client_t* smtp_client_connect (int_t *premium*)

Connect to a randomly selected mail relay server, and wait for a successful banner message.

Parameters:

premium if set, a premium relay will be selected instead of a standard one.

Returns:

NULL on failure or a pointer to the newly established network client object connected to a mail relay on success.

Definition at line 36 of file *relay.c*.

References *client_t::buffer*, *client_close()*, *client_connect()*, *client_read_line()*, *client_secure()*, *magma_t::count*, *magma_t::host*, *client_t::line*, *log_pedantic*, *magma*, *MAGMA_RELAY_INSTANCES*, *relay_t::name*,

net_set_timeout(), relay_t::port, magma_t::premium, rand_get_uint32(), magma_t::relay, relay_t::secure, client_t::sockd, st_char_get(), st_length_get(), st_length_int(), and magma_t::timeout.

Referenced by portal_smtp_relay_message(), smtp_bounce(), smtp_forward_message(), smtp_relay_message(), smtp_reply(), and smtp_send_message().

int_t smtp_client_send_data (client_t * *client*, stringer_t * *message*)

Issue a DATA command to an smtp server, and wait for a successful response.

Parameters:

client a pointer to the network client to issue the DATA command.

message a pointer to a managed string containing the body of the message to be sent.

Returns:

-2 if the remote server rejected the command, -1 on general network failure, or 1 on success.

Definition at line 238 of file relay.c.

References client_t::buffer, client_read_line(), client_write(), length, client_t::line, log_pedantic, PLACER, st_char_get(), st_length_get(), and st_length_int().

Referenced by portal_smtp_relay_message(), smtp_bounce(), smtp_forward_message(), smtp_relay_message(), smtp_reply(), and smtp_send_message().

int_t smtp_client_send_helo (client_t * *client*)

Issue a EHLO command to an smtp server, or fall back to HELO, and wait for a successful response.

Parameters:

client a pointer to the network client to issue the remote command.

Returns:

-1 on failure or 1 on success.

Definition at line 115 of file relay.c.

References client_t::buffer, client_print(), client_read_line(), magma_t::host, client_t::line, log_pedantic, magma, magma_t::name, st_char_get(), and st_length_get().

Referenced by portal_smtp_relay_message(), smtp_bounce(), smtp_forward_message(), smtp_relay_message(), smtp_reply(), and smtp_send_message().

int_t smtp_client_send_mailfrom (client_t * *client*, stringer_t * *mailfrom*, size_t *send_size*)

Issue a MAIL FROM command to an smtp server, and wait for a successful response.

Parameters:

client a pointer to the network client to issue the MAIL FROM command.

mailfrom a pointer to a managed string containing the address parameter for the MAIL FROM command.

send_size if greater than 0, specify the optional SIZE parameter to the MAIL FROM command.

Returns:

-2 if the remote server rejected the command, -1 on general network failure, or 1 on success.

Definition at line 165 of file relay.c.

References `client_t::buffer`, `client_print()`, `client_read_line()`, `client_t::line`, `log_pedantic`, `st_char_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by `portal_smtp_relay_message()`, `smtp_relay_message()`, and `smtp_send_message()`.

`int_t smtp_client_send_nullfrom (client_t * client)`

Issue a MAIL FROM command to an smtp server with a null sender, and wait for a successful response.

Note:

Null senders are used when the sender is not concerned about being notified about bounced messages.

Parameters:

client a pointer to the network client to issue the MAIL FROM command.

Returns:

-2 if the remote server rejected the command, -1 on general network failure, or 1 on success.

Definition at line 192 of file `relay.c`.

References `client_t::buffer`, `client_print()`, `client_read_line()`, `client_t::line`, `log_pedantic`, `st_char_get()`, and `st_length_int()`.

Referenced by `smtp_bounce()`, `smtp_forward_message()`, and `smtp_reply()`.

`int_t smtp_client_send_rcptto (client_t * client, stringer_t * rcptto)`

Issue a RCPT TO command to an smtp server, and wait for a successful response.

Parameters:

client a pointer to the network client to issue the RCPT TO command.

rcptto a pointer to a managed string containing the recipient address parameter for the RCPT TO command.

Returns:

-2 if the remote server rejected the command, -1 on general network failure, or 1 on success.

Definition at line 215 of file `relay.c`.

References `client_t::buffer`, `client_print()`, `client_read_line()`, `client_t::line`, `log_pedantic`, `st_char_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by `portal_smtp_relay_message()`, `smtp_bounce()`, `smtp_forward_message()`, `smtp_relay_message()`, `smtp_reply()`, and `smtp_send_message()`.

`int_t smtp_compare (const void * compare, const void * command)`

Definition at line 16 of file `commands.c`.

References `command_t::length`, `PLACER`, `st_cmp_ci_eq()`, `st_cmp_ci_starts()`, and `command_t::string`.

Referenced by `smtp_process()`, and `smtp_sort()`.

`void smtp_data (connection_t * con)`

LOW: Why are -2 and -3 identical?

Definition at line 1067 of file smtp.c.

References smtp_session_t::authenticated, con_print(), con_write_bl(), smtp_session_t::helo, smtp_session_t::in_prefs, magma, mail_add_required_headers(), mail_count_received(), mail_create_message(), mail_destroy_message(), mail_message_cleanup(), smtp_session_t::mailfrom, smtp_session_t::max_length, smtp_session_t::message, smtp_session_t::out_prefs, smtp_outbound_prefs_t::recipients, magma_t::relay_limit, requeue(), magma_t::smtp, connection_t::smtp, smtp_data_inbound(), smtp_data_outbound(), smtp_data_read(), smtp_quit(), smtp_requeue(), and st_free().

Referenced by smtp_process().

void smtp_disabled (connection_t * con)

A stub function for an SMTP command that has not been implemented.

Note:

Executing a disabled command while result in a small delay and the protocol violation counter being incremented.

Returns:

This function returns no value.

Definition at line 168 of file smtp.c.

References connection_t::command, con_print(), server_t::delay, command_t::length, connection_t::protocol, connection_t::server, command_t::string, server_t::violations, and connection_t::violations.

void smtp_ehlo (connection_t * con)

Process an SMTP EHLO command.

See also:

smtp_parse_helo_domain()

Note:

Any prior domain specified by a HELO/EHLO command will be overwritten.

Parameters:

con the SMTP client connection issuing the command.

Returns:

This function returns no value.

Definition at line 103 of file smtp.c.

References con_print(), con_secure(), con_write_bl(), server_t::domain, smtp_session_t::esmtp, smtp_session_t::helo, magma, magma_t::message_length_limit, connection_t::server, magma_t::smtp, connection_t::smtp, smtp_parse_helo_domain(), st_char_get(), st_cleanup(), and st_length_int().

int_t smtp_fetch_authorization (credential_t * cred, smtp_outbound_prefs_t ** output)

Check if a user is authorized to send messages, and retrieve the user's outbound smtp preferences.

Note:

This function first checks if the account is locked in the Users table; next it populates the user's outbound mail preferences with a combination of data from the Users and Dispatch tables. The number of messages the user has sent in the past 24 hours is also computed from the Transmitting table.

Parameters:

cred a pointer to a credential object for the user, which must be of type CREDENTIAL_AUTH.
output a pointer to the address of an outbound smtp preferences object to receive the value of the lookup.

Returns:

1 on success or <= 0 on failure. 0: authentication failure (invalid username/password combination). -1: general or database failure. -2: the account is subject to an administrative lock. -3: the account is locked due to suspicion of abuse violations. -4: the account has been locked at the request of the user.

Definition at line 577 of file datatier.c.

References credential_t::auth, CREDENTIAL_AUTH, smtp_outbound_prefs_t::daily_send_limit, smtp_outbound_prefs_t::domain, smtp_outbound_prefs_t::importance, log_error, log_pedantic, meta_data_update_lock(), mm_alloc(), mm_free(), mm_wipe(), res_field_int8(), res_field_string(), res_field_uint32(), res_field_uint64(), res_row_next(), res_table_free(), smtp_outbound_prefs_t::send_size_limit, smtp_outbound_prefs_t::sent_today, smtp_outbound_prefs_t::ssl, st_char_get(), st_length_get(), st_length_int(), stmt_get_result(), credential_t::type, and smtp_outbound_prefs_t::username.

Referenced by portal_outbound_checks(), smtp_auth_login(), and smtp_auth_plain().

stringer_t* smtp_fetch_autoreply (uint64_t autoreply, uint64_t username)

Fetch a specified auto-reply message for a user.

Note:

This function first checks the cache, and falls back to the database.

Parameters:

autoreply the numerical id of the auto-reply message in the database.
username the numerical id of the user to whom the auto-reply message belongs.

Returns:

NULL on failure, or a pointer to a managed string containing the user's auto-reply on success.

Definition at line 56 of file datatier.c.

References cache_get(), cache_set(), log_pedantic, mm_wipe(), PLACER, res_field_string(), res_row_next(), res_table_free(), and stmt_get_result().

Referenced by smtp_reply().

int_t smtp_fetch_inbound (credential_t * cred, stringer_t * address, smtp_inbound_prefs_t ** output)

Fetch a user's smtp inbound preferences from the database.

Parameters:

cred a pointer to the credential object of a user with
address Returns -1 for errors, -2 for an admin lock, -3 for an inactivity lock, -4 for a user lock, -5 for an abuse lock, -6 if the domain isn't local and 0 if the domain is local but the address wasn't found. If everything works, return 1 to indicate success.

Definition at line 116 of file `datatier.c`.

References `smtp_inbound_filter_t::action`, `smtp_inbound_prefs_t::address`, `smtp_inbound_prefs_t::autoreply`, `smtp_inbound_prefs_t::bounces`, `CONTIGUOUS`, `CREDENTIAL_MAIL`, `smtp_inbound_prefs_t::daily_recv_limit`, `smtp_inbound_prefs_t::daily_recv_limit_ip`, `smtp_inbound_prefs_t::dkim`, `smtp_inbound_prefs_t::dkimaction`, `smtp_inbound_prefs_t::domain`, `domain_wildcard()`, `smtp_inbound_filter_t::expression`, `smtp_inbound_filter_t::field`, `smtp_inbound_prefs_t::filters`, `smtp_inbound_filter_t::foldernum`, `smtp_inbound_prefs_t::forwarded`, `smtp_inbound_prefs_t::greylist`, `smtp_inbound_prefs_t::greytime`, `HEAP`, `smtp_inbound_prefs_t::inbox`, `inx_alloc()`, `inx_insert()`, `smtp_inbound_filter_t::label`, `smtp_inbound_filter_t::location`, `log_error`, `log_pedantic`, `M_INX_LINKED`, `M_TYPE_UINT64`, `credential_t::mail`, `MANAGED_T`, `mm_alloc()`, `mm_free()`, `mm_wipe()`, `smtp_inbound_prefs_t::overquota`, `smtp_inbound_prefs_t::phish`, `smtp_inbound_prefs_t::phishaction`, `smtp_inbound_prefs_t::quota`, `smtp_inbound_prefs_t::rbl`, `smtp_inbound_prefs_t::rblaction`, `smtp_inbound_prefs_t::rcptto`, `smtp_inbound_prefs_t::rcv_size_limit`, `res_field_block()`, `res_field_int8()`, `res_field_length()`, `res_field_string()`, `res_field_uint32()`, `res_field_uint64()`, `res_row_count()`, `res_row_next()`, `res_table_free()`, `smtp_inbound_prefs_t::rollout`, `smtp_inbound_filter_t::rulenum`, `smtp_inbound_prefs_t::secure`, `SMTP_FILTER_ACTION_LABEL`, `SMTP_FILTER_ACTION_MOVE`, `SMTP_FILTER_LOCATION_FIELD`, `smtp_free_inbound()`, `smtp_get_action()`, `smtp_list_free_filter()`, `smtp_inbound_prefs_t::spam`, `smtp_inbound_prefs_t::spamaction`, `smtp_inbound_prefs_t::spf`, `smtp_inbound_prefs_t::spfaction`, `st_char_get()`, `st_dupe()`, `st_dupe_opts()`, `st_length_get()`, `st_length_int()`, `stmt_get_result()`, `smtp_inbound_prefs_t::stor_size`, `smtp_inbound_filter_t::type`, `credential_t::type`, `multi_t::u64`, `smtp_inbound_prefs_t::usernum`, `multi_t::val`, `smtp_inbound_prefs_t::virus`, and `smtp_inbound_prefs_t::virusaction`.

Referenced by `smtp_rcpt_to()`.

table_t* smtp_fetch_rollmessages (uint64_t usernum)

Retrieve a list, at a maximum of 20 entries, of the oldest messages owned by a user.

Parameters:

usernum the numerical id of the user whose messages are to be queried.

Returns:

NULL on failure, or a pointer to a sql results set containing the user's oldest messages on success.

Definition at line 327 of file `datatier.c`.

References `log_pedantic`, `mm_wipe()`, and `stmt_get_result()`.

Referenced by `smtp_rollout()`.

int_t smtp_forward_message (server_t * server, stringer_t * address, stringer_t * message, stringer_t * id, int_t mark, uint64_t signum, uint64_t sigkey)

Definition at line 115 of file `transmit.c`.

References `log_pedantic`, `mail_add_forward_headers()`, `smtp_client_close()`, `smtp_client_connect()`, `smtp_client_send_data()`, `smtp_client_send_helo()`, `smtp_client_send_nullfrom()`, `smtp_client_send_rcptto()`, `st_dupe()`, and `st_free()`.

Referenced by `smtp_accept_message()`.

void smtp_free_inbound (smtp_inbound_prefs_t * inbound)

Free a list of SMTP inbound mail preferences and its underlying data.

Parameters:

inbound a pointer to the head of the SMTP inbound mail preferences list to be destroyed.

Returns:

This function returns no value.

Definition at line 235 of file session.c.

References smtp_inbound_prefs_t::address, smtp_inbound_prefs_t::domain, smtp_inbound_prefs_t::filters, smtp_inbound_prefs_t::forwarded, inx_cleanup(), mm_free(), smtp_inbound_prefs_t::next, smtp_inbound_prefs_t::rcptto, smtp_inbound_prefs_t::spamsig, and st_cleanup().

Referenced by smtp_fetch_inbound(), smtp_rcpt_to(), smtp_session_destroy(), and smtp_session_reset().

void smtp_free_outbound (smtp_outbound_prefs_t * *outbound*)

Free a set of SMTP outbound mail preferences and its underlying data.

Parameters:

outbound a pointer to the SMTP outbound mail preferences set to be destroyed.

Returns:

This function returns no value.

Definition at line 217 of file session.c.

References smtp_outbound_prefs_t::domain, mm_free(), smtp_outbound_prefs_t::recipients, smtp_free_recipients(), and st_cleanup().

Referenced by smtp_session_destroy().

void smtp_free_recipients (smtp_recipients_t * *recipients*)

Free a list of SMTP recipients and its underlying data.

Parameters:

recipients a pointer to the head of the recipients list to be destroyed.

Returns:

This function returns no value.

Definition at line 198 of file session.c.

References smtp_recipients_t::address, mm_free(), smtp_recipients_t::next, and st_cleanup().

Referenced by smtp_free_outbound(), and smtp_session_reset().

int_t smtp_get_action (chr_t * *string*, size_t *length*)

Parse an smtp event action string from the Dispatch table.

Note:

These values describe user-specified actions for events related to the spam filter, virus scanner, phishing detection, SPF, DKIM, and RBL checks.

Parameters:

string a pointer to a null-terminated string containing the name of the action.
length the length, in bytes, of the action string.

Returns:

the code of the corresponding smtp event action, SMTP_ACTION_UNDEFINED if the action is not known, or SMTP_ACTION_ERROR on failure.

Definition at line 24 of file datatier.c.

References `log_error`, `PLACER`, `SMTP_ACTION_BOUNCE`, `SMTP_ACTION_DELETE`, `SMTP_ACTION_ERROR`, `SMTP_ACTION_MARK`, `SMTP_ACTION_MARK_READ`, `SMTP_ACTION_REJECT`, `SMTP_ACTION_UNDEFINED`, and `st_cmp_cs_eq()`.

Referenced by `smtp_fetch_inbound()`.

void smtp_helo (connection_t * con)

Process an SMTP HELO command.

See also:

`smtp_parse_helo_domain()`

Note:

Any prior domain specified by a HELO/EHLO command will be overwritten.

Parameters:

con the SMTP client connection issuing the command.

Returns:

This function returns no value.

Definition at line 132 of file smtp.c.

References `con_print()`, `con_write_bl()`, `server_t::domain`, `smtp_session_t::esmtplib`, `smtp_session_t::helo`, `connection_t::server`, `connection_t::smtp`, `smtp_parse_helo_domain()`, `st_char_get()`, `st_cleanup()`, and `st_length_int()`.

void smtp_init (connection_t * con)

The start of the protocol handler for the SMTP server.

Parameters:

con the new inbound SMTP client connection.

Returns:

This function returns no value.

Definition at line 1177 of file smtp.c.

References `smtp_session_t::bypass`, `con_print()`, `con_reverse_enqueue()`, `server_t::domain`, `connection_t::server`, `connection_t::smtp`, `smtp_bypass_check()`, `smtp_requeue()`, `st_char_get()`, and `st_length_int()`.

Referenced by `protocol_enqueue()`, and `submission_init()`.

uint64_t smtp_insert_spamsig (smtp_inbound_prefs_t * prefs, uint64_t key, int_t code)

Store a spam signature in the database.

See also:

dspam_process()

Parameters:

prefs a pointer to the specified user's inbound mail preferences data.

key a randomly chosen authentication key for the signature.

code the dspam return code for the message from dspam_process()

Returns:

0 on failure, or the number of the newly inserted spam signature on success.

Definition at line 428 of file datatier.c.

References log_pedantic, mm_wipe(), smtp_inbound_prefs_t::spamsig, st_char_get(), st_length_get(), stmt_insert(), and smtp_inbound_prefs_t::usernum.

Referenced by smtp_store_spamsig().

void smtp_invalid (connection_t * con)

A function that is executed when an invalid SMTP command is executed.

Returns:

This function returns no value.

Definition at line 181 of file smtp.c.

References con_write_bl(), server_t::delay, connection_t::protocol, connection_t::server, server_t::violations, and connection_t::violations.

Referenced by smtp_process().

void smtp_list_free_filter (smtp_inbound_filter_t * filter)

Free an SMTP inbound filter and its underlying data.

Parameters:

filter a pointer to the SMTP inbound filter to be destroyed.

Returns:

This function returns no value.

Definition at line 259 of file session.c.

References smtp_inbound_filter_t::expression, smtp_inbound_filter_t::field, smtp_inbound_filter_t::label, mm_free(), and st_cleanup().

Referenced by smtp_fetch_inbound().

void smtp_mail_from (connection_t * con)

Specify the identity of a message's sender, in response to an SMTP MAIL FROM command.

See also:**smtp_parse_mail_from_path()****Note:**

This command must be preceded by a HELO command and successful authentication. Any prior email address specified by a MAIL FROM command will be overwritten. If the SIZE parameter was specified with the MAIL FROM command, its value will be compared to the maximum value specified in the smtp.message_length_limit configuration option.

Parameters:

con the SMTP client connection issuing the command.

Returns:

This function returns no value.

Definition at line 62 of file smtp.c.

References smtp_session_t::authenticated, con_write_bl(), smtp_session_t::helo, magma, smtp_session_t::mailfrom, magma_t::message_length_limit, magma_t::smtp, connection_t::smtp, smtp_parse_mail_from_path(), smtp_session_reset(), st_free(), and smtp_session_t::suggested_length.

void smtp_noop (connection_t * *con*)

Perform an SMTP NOOP (no-operation) command.

Note:

This command does essentially nothing and is mostly a way to keep connections alive without timing out due to inactivity.

Returns:

This function returns no value.

Definition at line 157 of file smtp.c.

References con_write_bl().

stringer_t* smtp_parse_auth (stringer_t * *data*)

parse.c

parse.c

Note:

This function will stop reading input when an invalid base64-encoding character is encountered.

Parameters:

con the SMTP client connection issuing the AUTH command.

Returns:

a managed string containing the domain specified by the AUTH command, or NULL on failure.

Definition at line 118 of file parse.c.

References length, log_pedantic, st_empty_out(), and st_import().

Referenced by smtp_auth_login(), and smtp_auth_plain().

stringer_t* smtp_parse_helo_domain (connection_t * *con*)

Parse the domain specified as the parameter to an SMTP HELO or EHLO command.

Note:

Valid domain names may only contain letters, numbers, periods and hyphens. This function will return a lowercase domain name, and stop reading input when an invalid character is encountered.

Parameters:

con the SMTP client connection issuing the command.

Returns:

a managed string containing the domain specified by the HELO command, or NULL on failure.

Definition at line 315 of file parse.c.

References connection_t::command, magma_t::helo_length_limit, length, connection_t::line, log_pedantic, lower_chr(), magma, connection_t::network, pl_char_get(), pl_empty(), pl_length_get(), pl_length_int(), pl_trim_end(), magma_t::smtp, st_import(), and command_t::string.

Referenced by smtp_ehlo(), and smtp_helo().

stringer_t* smtp_parse_mail_from_path (connection_t * con)

Extract the provided path from the command line Reverse-path = Path Forward-path = Path Path = "<" [A-d-l ":"] Mailbox ">" A-d-l = At-domain *("," A-d-l) At-domain = @ domain

Parameters:

con A connection structure which presumably contains a the MAIL FROM value inside the line buffer.

Returns:

Returns a stringer with the path that was extracted or NULL to indicate a problem.

Definition at line 173 of file parse.c.

References magma_t::address_length_limit, connection_t::command, CONSTANT, length, connection_t::line, log_pedantic, lower_chr(), magma, connection_t::network, pl_char_get(), pl_empty(), pl_length_get(), pl_length_int(), pl_trim(), pl_trim_end(), PLACER, placer_t, connection_t::smtp, magma_t::smtp, st_cmp_ci_starts(), st_cmp_cs_eq(), st_import(), command_t::string, smtp_session_t::suggested_eight_bit, smtp_session_t::suggested_length, tok_get_bl(), tok_get_count_bl(), tok_get_pl(), and uint64_conv_pl().

Referenced by smtp_mail_from().

stringer_t* smtp_parse_rcpt_to (connection_t * con)

LOW: The list of characters allowed in an email address needs to be verified against the RFC's.

LOW: The parser should use different lists of valid characters for the the local and domain portions of an email address. Extract the provided path from the command line Reverse-path = Path Forward-path = Path Path = "<" [A-d-l ":"] Mailbox ">" A-d-l = At-domain *("," A-d-l) At-domain = @ domain

Parameters:

con A connection structure which presumably contains a the RCPT TO value inside the line buffer.

Returns:

Returns a stringer with the path that was extracted or NULL to indicate a problem.

Definition at line 29 of file parse.c.

References magma_t::address_length_limit, connection_t::command, length, connection_t::line, log_pedantic, lower_chr(), magma, connection_t::network, pl_char_get(), pl_empty(), pl_length_get(), pl_length_int(), pl_trim_end(), magma_t::smtp, st_import(), and command_t::string.

Referenced by smtp_rcpt_to().

void smtp_process (connection_t * con)

The main entry point in the smtp server for processing commands issued by clients.

Parameters:

con a pointer to the connection object of the client issuing the smtp command.

Returns:

This function returns no value.

Definition at line 53 of file commands.c.

References `connection_t::command`, `command`, `con_read_line()`, `server_t::cutoff`, `enqueue()`, `command_t::function`, `command_t::length`, `connection_t::line`, `connection_t::network`, `pl_char_get()`, `pl_empty()`, `pl_length_get()`, `connection_t::protocol`, `requeue()`, `connection_t::server`, `smtp_commands`, `smtp_compare()`, `smtp_data()`, `smtp_invalid()`, `smtp_process()`, `smtp_quit()`, `smtp_requeue()`, `connection_t::spins`, `command_t::string`, `server_t::violations`, and `connection_t::violations`.

Referenced by `smtp_process()`, and `smtp_requeue()`.

void smtp_quit (connection_t * con)

Gracefully terminate an SMTP session, especially in response to an SMTP QUIT command.

Note:

The standards specify that the receiver MUST send an OK reply, and then close the transmission channel.

Parameters:

the SMTP client connection to be terminated.

Returns:

This function returns no value.

Definition at line 196 of file smtp.c.

References `con_destroy()`, `con_status()`, and `con_write_bl()`.

Referenced by `smtp_data()`, `smtp_process()`, and `smtp_requeue()`.

void smtp_rcpt_to (connection_t * con)

BUG: The code should be able to differentiate between invalid addresses which trigger a NULL return and allocation (or similar) errors which are temporary.

BUG: Detect messages 'from' a local user/domain and tell them to authenticate first.

Definition at line 465 of file smtp.c.

References `magma_t::abuse`, `smtp_inbound_prefs_t::address`, `magma_t::admin`, `smtp_session_t::authenticated`, `smtp_session_t::bypass`, `smtp_session_t::checked`, `con_addr()`, `con_addr_presentation()`, `con_print()`, `con_secure()`, `con_write_bl()`, `magma_t::contact`, `credential_alloc_mail()`, `credential_free()`, `smtp_inbound_prefs_t::daily_rcv_limit`, `smtp_inbound_prefs_t::daily_rcv_limit_ip`, `smtp_outbound_prefs_t::daily_send_limit`, `smtp_inbound_prefs_t::forwarded`, `smtp_inbound_prefs_t::greylist`, `smtp_inbound_prefs_t::greytime`, `smtp_session_t::helo`, `lower_st()`, `magma`, `credential_t::mail`, `smtp_session_t::mailfrom`, `MANAGEDBUF`, `smtp_session_t::max_length`, `MEMORYBUF`, `smtp_session_t::num_recipients`, `smtp_session_t::out_prefs`, and `smtp_inbound_prefs_t::overquota`.

smtp_session_t::rbl, smtp_inbound_prefs_t::rbl, smtp_inbound_prefs_t::rblaction,
 smtp_inbound_prefs_t::rcptto, magma_t::recipient_limit, smtp_inbound_prefs_t::recv_size_limit,
 smtp_inbound_prefs_t::rollout, smtp_outbound_prefs_t::send_size_limit, smtp_outbound_prefs_t::sent_today,
 magma_t::smtp, connection_t::smtp, SMTP_ACTION_REJECT, smtp_add_inbound(), smtp_add_recipient(),
 smtp_check_duplicate_recipient(), smtp_check_greylist(), smtp_check_rbl(), smtp_check_receive_quota(),
 smtp_fetch_inbound(), smtp_free_inbound(), smtp_parse_rcpt_to(), smtp_session_t::spf,
 smtp_inbound_prefs_t::spf, spf_check(), smtp_inbound_prefs_t::spfaction, smtp_outbound_prefs_t::ssl,
 st_char_get(), st_cmp_ci_eq(), st_free(), st_length_int(), smtp_session_t::suggested_length, and
 smtp_inbound_prefs_t::usernum.

int_t smtp_relay_message (connection_t * con, stringer_t ** result)

Relay an outbound smtp message for a user.

Note:

The following process occurs before the message will be sent: 1. Necessary outbound headers are attached to the message.* 2. An outbound connection to a mail relay server is established (with a premium or normal server pool). 3. Once the connection is negotiated, an RCPT TO command is issued for each of the message's recipients. 4. The mail message data is sent and the client connection is closed.

Parameters:

con a pointer to the connection object across which the outbound mail was attempted to be sent.
result a pointer to the address of a managed string that will receive the server's last response to the mail send attempt, regardless of whether or not it was successful.

Returns:

1 if the message was successfully sent or -1 on failure.

Definition at line 27 of file transmit.c.

References smtp_recipients_t::address, CONTIGUOUS, HEAP, smtp_outbound_prefs_t::importance,
 client_t::line, log_pedantic, mail_add_outbound_headers(), smtp_session_t::mailfrom, MANAGED_T,
 smtp_session_t::message, smtp_recipients_t::next, smtp_session_t::out_prefs,
 smtp_outbound_prefs_t::recipients, connection_t::smtp, smtp_client_close(), smtp_client_connect(),
 smtp_client_send_data(), smtp_client_send_helo(), smtp_client_send_mailfrom(), smtp_client_send_rcptto(),
 st_dupe_opts(), and smtp_message_t::text.

Referenced by smtp_data_outbound().

**int_t smtp_reply (stringer_t * from, stringer_t * to, uint64_t usernum, uint64_t autoreply,
 int_t spf, int_t dkim)**

Definition at line 363 of file transmit.c.

References cache_get_u64(), cache_set_u64(), dkim_create(), hash_crc64(), lock_get(), lock_release(),
 log_pedantic, MANAGEDBUF, rand_choices(), smtp_client_close(), smtp_client_connect(),
 smtp_client_send_data(), smtp_client_send_helo(), smtp_client_send_nullfrom(), smtp_client_send_rcptto(),
 smtp_fetch_autoreply(), st_char_get(), st_cleanup(), st_free(), st_length_int(), st_merge, and st_sprint().

Referenced by smtp_accept_message().

void smtp_requeue (connection_t * con)

commands.c

Definition at line 36 of file commands.c.

References `con_status()`, `server_t::cutoff`, `enqueue()`, `connection_t::protocol`, `connection_t::server`, `smtp_process()`, `smtp_quit()`, `status`, `server_t::violations`, and `connection_t::violations`.

Referenced by `smtp_data()`, `smtp_init()`, and `smtp_process()`.

int_t smtp_rollout (smtp_inbound_prefs_t * prefs)

Delete the oldest mail message owned by a user until their storage usage falls below their storage quota.

Parameters:

prefs a pointer to the specified user's inbound mail preferences data.

Returns:

1 on success or < 0 on failure, where -1: An error occurred retrieving the rollout message list from the database. -2: The user lock could not be acquired.

Definition at line 95 of file `accept.c`.

References `log_pedantic`, `mail_remove_message()`, `OBJECT_MESSAGES`, `smtp_inbound_prefs_t::quota`, `res_field_string()`, `res_field_uint32()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `serial_increment()`, `smtp_fetch_rollmessages()`, `st_char_get()`, `st_free()`, `smtp_inbound_prefs_t::stor_size`, `user_lock()`, `user_unlock()`, and `smtp_inbound_prefs_t::usernum`.

Referenced by `smtp_accept_message()`.

void smtp_rset (connection_t * con)

Reset the SMTP session, in response to an SMTP RSET command.

Note:

This command clears any sender, recipient, and mail data, along with all buffers and state tables. In other words, return to the state immediately after HELO.

Parameters:

con the SMTP client connection issuing the command.

Returns:

This function returns no value.

Definition at line 220 of file `smtp.c`.

References `con_write_bl()`, and `smtp_session_reset()`.

int_t smtp_send_message (stringer_t * to, stringer_t * from, stringer_t * message)

Relay an outbound smtp message for the user.

Parameters:

to a managed string containing the name of the mail recipient.

from a managed string containing the address from which the email is being sent.

message a managed string containing the raw body of the mail message.

Returns:

-1 on error or 1 on success.

Definition at line 503 of file `transmit.c`.

References `log_pedantic`, `smtp_client_close()`, `smtp_client_connect()`, `smtp_client_send_data()`, `smtp_client_send_helo()`, `smtp_client_send_mailfrom()`, and `smtp_client_send_rcptto()`.

Referenced by `contact_business()`, and `register_business_step2()`.

void smtp_session_destroy (connection_t * con)

Destroy the data associated with an SMTP session.

Parameters:

con the SMTP client connection to be destroyed.

Returns:

This function returns no value.

Definition at line 62 of file `session.c`.

References `smtp_session_t::helo`, `smtp_session_t::in_prefs`, `mail_destroy_message()`, `smtp_session_t::mailfrom`, `smtp_session_t::message`, `smtp_session_t::out_prefs`, `connection_t::smtp`, `smtp_free_inbound()`, `smtp_free_outbound()`, and `st_cleanup()`.

Referenced by `con_destroy()`.

void smtp_session_reset (connection_t * con)

Reset an SMTP session to its initialized state.

Parameters:

con the SMTP client connection to be reset.

Returns:

This function returns no value.

Definition at line 20 of file `session.c`.

References `smtp_session_t::checked`, `smtp_session_t::dkim`, `smtp_session_t::in_prefs`, `mail_destroy_message()`, `smtp_session_t::mailfrom`, `smtp_session_t::max_length`, `smtp_session_t::message`, `smtp_session_t::num_recipients`, `smtp_session_t::out_prefs`, `smtp_session_t::rbl`, `smtp_outbound_prefs_t::recipients`, `connection_t::smtp`, `smtp_free_inbound()`, `smtp_free_recipients()`, `smtp_session_t::spf`, `st_cleanup()`, `smtp_session_t::suggested_eight_bit`, `smtp_session_t::suggested_length`, and `smtp_session_t::virus`.

Referenced by `smtp_auth_login()`, `smtp_auth_plain()`, `smtp_data_inbound()`, `smtp_data_outbound()`, `smtp_mail_from()`, `smtp_rset()`, and `smtp_starttls()`.

void smtp_sort (void)

Sort the SMTP command table to be ready for binary searches.

Returns:

This function returns no value.

Definition at line 31 of file `commands.c`.

References `smtp_commands`, and `smtp_compare()`.

Referenced by `protocol_init()`.

void smtp_starttls (connection_t * con)

TODO: Review error messages and update them with the appropriate response code.

Initialize a TLS session for an unauthenticated SMTP session.

Parameters:

con the connection of the SMTP endpoint requesting the transport layer security upgrade.

Returns:

This function returns no value.

Definition at line 22 of file smtp.c.

References connection_t::buffer, con_secure(), con_write_bl(), connection_t::line, log_pedantic, M_SSL_BIO_NOCLOSE, connection_t::network, pl_null(), connection_t::server, smtp_session_reset(), connection_t::sockd, connection_t::ssl, ssl_alloc(), st_length_set(), stats_increment_by_name(), and connection_t::status.

int_t smtp_store_message (smtp_inbound_prefs_t * prefs, stringer_t ** local)

Store a received SMTP message as a generic mail message, both on disk and in the database.

See also:

mail_store_messages()

Returns:

-1 on failure or 1 on success.

Definition at line 20 of file accept.c.

References smtp_inbound_prefs_t::foldernum, log_error, log_pedantic, MAIL_MARK_BLACKHOLED, MAIL_MARK_INFECTED, MAIL_MARK_JUNK, MAIL_MARK_PHISHING, MAIL_MARK_SPOOFED, MAIL_STATUS_RECENT, MAIL_STATUS_SEEN, mail_store_message(), smtp_inbound_prefs_t::mark, smtp_inbound_prefs_t::messagenum, meta_data_user_build_storage_keys(), OBJECT_MESSAGES, smtp_inbound_prefs_t::secure, serial_increment(), smtp_inbound_prefs_t::signum, SMTP_MARK_PHISH, SMTP_MARK_RBL, SMTP_MARK_READ, SMTP_MARK_SPAM, SMTP_MARK_SPOOF, SMTP_MARK_VIRUS, smtp_inbound_prefs_t::spamkey, status, user_lock(), user_unlock(), and smtp_inbound_prefs_t::usernum.

Referenced by smtp_accept_message().

bool_t smtp_store_spamsig (smtp_inbound_prefs_t * prefs, int_t spam)

Generate a random key for a spam signature and store it in the database.

Parameters:

prefs the user's smtp inbound preferences object, with the spam signature field set.

spam the dspam return code associated with the spam signature.

Returns:

true if the key was inserted into the database successfully, or false on failure.

Definition at line 177 of file accept.c.

References imap_fetch_response_t::key, log_pedantic, rand_get_uint64(), smtp_inbound_prefs_t::signum, smtp_insert_spamsig(), smtp_inbound_prefs_t::spamkey, and uint64_digits().

Referenced by smtp_accept_message().

void smtp_update_receive_stats (connection_t * con, smtp_inbound_prefs_t * prefs)

Update the receiving statistics and per-user log tables in the database for a successfully received smtp message.

Note:

The Receiving table is updated with the subnet address from which the message was received; the Log table for the user is updated to reflect the newly calculated totals of bounces or messages received.

Parameters:

con the connection across which the smtp message was received.

prefs a pointer to the user's smtp inbound mail preferences.

Returns:

This function returns no value.

Definition at line 356 of file datatier.c.

References smtp_inbound_prefs_t::bounces, con_addr_subnet(), log_pedantic, smtp_session_t::mailfrom, mm_wipe(), PLACER, connection_t::smtp, st_char_get(), st_cmp_cs_eq(), st_free(), st_length_get(), stmt_exec(), and smtp_inbound_prefs_t::usernum.

Referenced by smtp_accept_message().

void smtp_update_transmission_stats (connection_t * con)

Update the transmission and per-user log tables in the database for a successfully sent smtp message.

Note:

The Transmitting table is updated with the timestamp of this transaction; the Log table for the user is updated to reflect the newly calculated total for messages sent.

Parameters:

con a pointer to the connection object across which the smtp message was sent.

prefs a pointer to the user's smtp inbound mail preferences.

Returns:

This function returns no value.

Definition at line 480 of file datatier.c.

References log_pedantic, mm_wipe(), smtp_session_t::num_recipients, smtp_session_t::out_prefs, connection_t::smtp, stmt_exec(), and smtp_outbound_prefs_t::usernum.

Referenced by smtp_data_outbound().

void submission_init (connection_t * con)

Definition at line 1194 of file smtp.c.

References connection_t::smtp, smtp_init(), and smtp_session_t::submission.

Referenced by protocol_enqueue().

magma/network/write.c File Reference

Functions used to write data out via the network.

```
#include "magma.h"
```

Functions

- `int64_t con_write_bl (connection_t *con, char *block, size_t length)`
- *Write data to a network connection.* `int64_t con_write_st (connection_t *con, stringer_t *string)`
- *Write a managed string to a network connection.* `int64_t con_write_ns (connection_t *con, char *string)`
- *Write a null-terminated string to a network connection.* `int64_t con_write_pl (connection_t *con, placer_t string)`
- *Write a placer to a network connection.* `int64_t con_print (connection_t *con, chr_t *format,...)`
- *Write a formatted string to a network connection.* `int64_t client_write (client_t *client, stringer_t *s)`
- *Write data to a network client.* `int64_t client_print (client_t *client, chr_t *format,...)`

Write a formatted string to a network client.

Detailed Description

Functions used to write data out via the network.

Definition in file **write.c**.

Function Documentation

`int64_t client_print (client_t * client, chr_t * format, ...)`

Write a formatted string to a network client.

write.c

See also:

`client_write()`

Parameters:

client the network client connection to which the supplied data will be written.

format the format string for the data to be written to the network client connection.

... a `va_arg` style collection of variables to be expanded by the passed format string.

Returns:

-1 on general network failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 279 of file `write.c`.

References `client_write()`, `client_t::sockd`, `st_free()`, and `st_vaprint`.

Referenced by `smtp_client_send_helo()`, `smtp_client_send_mailfrom()`, `smtp_client_send_nullfrom()`, and `smtp_client_send_rcptto()`.

`int64_t client_write (client_t * client, stringer_t * s)`

Write data to a network client.

Note:

This function works regardless of whether or not the client connection is ssl-enabled. If the network write requires multiple system calls, then this code will loop until all the data has been transmitted.

Parameters:

client the network client connection to which the supplied data will be written.

s a managed string containing the data to be written to the network client connection.

Returns:

-1 on general network failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 199 of file write.c.

References `buflen`, `bufptr`, `length`, `log_pedantic`, `client_t::sockd`, `client_t::ssl`, `SSL_get_error_d`, `ssl_write()`, `st_empty_out()`, and `client_t::status`.

Referenced by `client_print()`, `smtp_client_close()`, and `smtp_client_send_data()`.

int64_t con_print (connection_t * con, chr_t * format, ...)

Write a formatted string to a network connection.

See also:

`con_write_bl()`

Parameters:

con the connection across which the supplied data will be written.

format a format string for the output string to be written to the connection's remote client.

... a variable argument list of parameters for the specified format string.

Returns:

-1 on general failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 142 of file write.c.

References `buflen`, `bufptr`, `con_write_bl()`, `length`, `mm_alloc()`, `mm_free()`, `connection_t::network`, `connection_t::sockd`, and `connection_t::status`.

Referenced by `http_parse_header()`, `http_print_301()`, `http_response_header()`, `http_response_options()`, `imap_append()`, `imap_capability()`, `imap_check()`, `imap_close()`, `imap_copy()`, `imap_create()`, `imap_delete()`, `imap_examine()`, `imap_expunge()`, `imap_fetch()`, `imap_id()`, `imap_idle()`, `imap_init()`, `imap_invalid()`, `imap_list()`, `imap_login()`, `imap_logout()`, `imap_lsub()`, `imap_noop()`, `imap_process()`, `imap_rename()`, `imap_search()`, `imap_select()`, `imap_starttls()`, `imap_status()`, `imap_store()`, `imap_subscribe()`, `imap_unsubscribe()`, `molten_stats()`, `pop_capa()`, `pop_last()`, `pop_list()`, `pop_retr()`, `pop_stat()`, `pop_top()`, `pop_uidl()`, `register_print_captcha()`, `smtp_data()`, `smtp_data_inbound()`, `smtp_data_outbound()`, `smtp_disabled()`, `smtp_ehlo()`, `smtp_helo()`, `smtp_init()`, `smtp_rcpt_to()`, and `teacher_print_message()`.

int64_t con_write_bl (connection_t * con, char * block, size_t length)

Write data to a network connection.

HIGH: Create a simpler method of triggering a queue event following a connection write operation, and audit the code to ensure write calls do not accidentally orphan a connection by not queuing the connection upon completion. In other words, always ensure that **enqueue()** is being called on a connection after all processing is performed, so that it is not lost (whether it is to be kept or not).

Note:

This function works regardless of whether or not the connection is ssl-enabled. If the network write requires multiple system calls, then this code will loop until all the data has been transmitted.

Parameters:

con the connection across which the supplied data will be written.

block a pointer to a data buffer containing the data to be written to the connection's remote client.

length the length, in bytes, of the data buffer to be written.

Returns:

-1 on general network failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 28 of file write.c.

References `buflen`, `bufptr`, `log_pedantic`, `connection_t::network`, `connection_t::sockd`, `connection_t::ssl`, `SSL_get_error_d`, `ssl_write()`, and `connection_t::status`.

Referenced by `con_print()`, `con_write_ns()`, `con_write_pl()`, `con_write_st()`, `imap_fetch()`, `imap_logout()`, `imap_parse_literal()`, `imap_process()`, `molten_invalid()`, `molten_stats()`, `pop_delete()`, `pop_init()`, `pop_invalid()`, `pop_list()`, `pop_noop()`, `pop_pass()`, `pop_quit()`, `pop_retr()`, `pop_rset()`, `pop_starttls()`, `pop_top()`, `pop_uidl()`, `pop_user()`, `smtp_auth_login()`, `smtp_auth_plain()`, `smtp_data()`, `smtp_data_inbound()`, `smtp_data_outbound()`, `smtp_ehlo()`, `smtp_helo()`, `smtp_invalid()`, `smtp_mail_from()`, `smtp_noop()`, `smtp_quit()`, `smtp_rcpt_to()`, `smtp_rset()`, and `smtp_starttls()`.

int64_t con_write_ns (connection_t * con, char * string)

Write a null-terminated string to a network connection.

See also:

`con_write_bl()`

Parameters:

con the connection across which the supplied data will be written.

string a pointer to a null-terminated string containing the data to be written to the connection's remote client.

Returns:

-1 on general network failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 117 of file write.c.

References `con_write_bl()`, and `ns_length_get()`.

int64_t con_write_pl (connection_t * con, placer_t string)

Write a placer to a network connection.

See also:

`con_write_bl()`

Parameters:

con the connection across which the supplied data will be written.

string a placer pointing to the data to be written to the connection's remote client.

Returns:

-1 on general network failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 129 of file write.c.

References `con_write_bl()`, `pl_char_get()`, and `pl_length_get()`.

int64_t con_write_st (connection_t * *con*, stringer_t * *string*)

Write a managed string to a network connection.

See also:

`con_write_bl()`

Parameters:

con the connection across which the supplied data will be written.

string a managed string containing the data to be written to the connection's remote client.

Returns:

-1 on general network failure, -2 if the connection was reset or closed, or the number of bytes that were written across the connection.

Definition at line 105 of file write.c.

References `con_write_bl()`, `st_char_get()`, and `st_length_get()`.

Referenced by `contact_print_form()`, `contact_print_message()`, `http_print_400()`, `http_print_403()`, `http_print_404()`, `http_print_405()`, `http_print_500()`, `http_print_500_log()`, `http_print_501()`, `http_response()`, `imap_fetch()`, `imap_search()`, `pop_retr()`, `pop_top()`, `portal_endpoint_attachments_progress()`, `portal_endpoint_error()`, `portal_endpoint_response()`, `portal_endpoint_search()`, `portal_print_login()`, `register_print_captcha()`, `register_print_message()`, `register_print_step1()`, `register_print_step2()`, `register_print_step3()`, `smtp_data_outbound()`, `statistics_process()`, `teacher_print_form()`, and `teacher_print_message()`.

magma/objects/config/config.c File Reference

The user configuration interface.

```
#include "magma.h"
```

Functions

- void **user_config_entry_free** (**user_config_entry_t** *entry)
- *Destroy a user config entry.* void **user_config_free** (**user_config_t** *collection)
- *Free a user config collection object.* **user_config_t** * **user_config_alloc** (uint64_t usernum)
- *Allocate a user config collection object for a user.* **user_config_entry_t** * **user_config_entry_alloc** (**stringer_t** *key, **stringer_t** *value, uint64_t flags)
- *Create and initialize new user config entry object.* **user_config_t** * **user_config_create** (uint64_t usernum)
- *Create a new user config collection object for the specified user and populate it with config entries from the database.* **int_t** **user_config_update** (**user_config_t** *collection)
- *Update a collection of user config options from the database, if necessary.* **int_t** **user_config_edit** (**user_config_t** *collection, **stringer_t** *key, **stringer_t** *value)

Change the value of, or delete a key from a user's config collection.

Detailed Description

The user configuration interface.

Definition in file **config.c**.

Function Documentation

user_config_t* user_config_alloc (uint64_t usernum)

Allocate a user config collection object for a user.

config.c

Parameters:

usernum the numerical user id for the requested user.

Returns:

NULL on failure or a pointer to the newly allocated user config collection object on success.

Definition at line 49 of file config.c.

References `align()`, `inx_alloc()`, `log_pedantic`, `M_INX_TREE`, `mm_alloc()`, `mm_free()`, `user_config_entry_free()`, and `user_config_t`.

Referenced by `user_config_create()`.

user_config_t* user_config_create (uint64_t usernum)

Create a new user config collection object for the specified user and populate it with config entries from the database.

Parameters:

userid the numerical user id for the user to be queried.

Returns:

NULL on failure, or a pointer to the newly created user config collection object on success.

Definition at line 111 of file config.c.

References log_pedantic, OBJECT_CONFIG, serial_get(), user_config_alloc(), user_config_fetch(), user_config_free(), and user_config_t.

Referenced by portal_endpoint_config_edit(), and portal_endpoint_config_load().

int_t user_config_edit (user_config_t * *collection*, stringer_t * *key*, stringer_t * *value*)

Change the value of, or delete a key from a user's config collection.

Parameters:

collection a pointer to a collection of user's config entries.

key a pointer to a managed string containing the name of the user config key to be modified.

value a pointer to a managed string containing the name of the new value of the key, or NULL if it is to be deleted.

Returns:

-1 on error or 1 on success.

Definition at line 165 of file config.c.

References inx_delete(), inx_find(), inx_replace(), log_pedantic, M_TYPE_STRINGER, OBJECT_CONFIG, serial_increment(), multi_t::st, st_cmp_cs_eq(), st_empty(), user_config_delete(), user_config_entry_alloc(), user_config_entry_free(), user_config_entry_t, user_config_upsert(), and multi_t::val.

Referenced by portal_endpoint_config_edit().

user_config_entry_t* user_config_entry_alloc (stringer_t * *key*, stringer_t * *value*, uint64_t *flags*)

Create and initialize new user config entry object.

Parameters:

key a managed string containing the key of the config entry object.

value a managed string containing the value of the config entry object.

flags a bitmask reflecting the flags of the config entry object.

Returns:

NULL on failure, or a pointer to the newly allocated and initialized user config entry object on success.

Definition at line 75 of file config.c.

References align(), FOREIGNDATA, JOINTED, log_pedantic, mm_alloc(), mm_copy(), PLACER_T, placer_t, st_data_get(), st_length_get(), STACK, and user_config_entry_t.

Referenced by user_config_edit(), and user_config_fetch().

void user_config_entry_free (user_config_entry_t * *entry*)

Destroy a user config entry.

Parameters:

entry a pointer to the user config entry object to be freed.

Returns:

This function returns no value.

Definition at line 20 of file config.c.

References mm_free().

Referenced by user_config_alloc(), user_config_edit(), and user_config_fetch().

void user_config_free (user_config_t * *collection*)

Free a user config collection object.

Parameters:

collection a pointer to the user config collection object to be freed.

Returns:

This function returns no value.

Definition at line 34 of file config.c.

References inx_cleanup(), and mm_free().

Referenced by portal_endpoint_config_edit(), portal_endpoint_config_load(), and user_config_create().

int_t user_config_update (user_config_t * *collection*)

Update a collection of user config options from the database, if necessary.

Note:

This function is not currently being used anywhere. If this function returns -1, the config entries are left in an undefined state and should not be relied upon.

Parameters:

collection a pointer to a collection of user config entries to be checked for updates.

Returns:

1 if the collection is up-to-date, 0 if it was refreshed to update changes, or -1 if the refresh operation failed.

Definition at line 137 of file config.c.

References inx_truncate(), OBJECT_CONFIG, serial_get(), and user_config_fetch().

magma/web/portal/config.c File Reference

Utilities functions for handling user config entries.

```
#include "magma.h"
```

Functions

- `json_t * portal_config_entry_flags (user_config_entry_t *entry)`
- *Get a user config entry's flags as a json object.* `json_t * portal_config_entry (user_config_entry_t *entry)`
- `json_t * portal_config_collection (user_config_t *collection)`

Get a collection of user config options as a json object.

Detailed Description

Utilities functions for handling user config entries.

Definition in file `config.c`.

Function Documentation

`json_t* portal_config_collection (user_config_t * collection)`

Get a collection of user config options as a json object.

`config.c`

Parameters:

collection a pointer to a user config object that contains all the config options pairs to be serialized.

Returns:

NULL on failure, or a pointer to a json object containing the collection of user config options on success.

Definition at line 67 of file `config.c`.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `json_decref_d`, `json_object_d`, `json_object_set_new_d`, `log_error`, `log_options`, `M_LOG_CRITICAL`, `portal_config_entry()`, `st_char_get()`, `USER_CONF_STATUS_CRITICAL`, and `user_config_entry_t`.

Referenced by `portal_endpoint_config_load()`.

`json_t* portal_config_entry (user_config_entry_t * entry)`

Definition at line 40 of file `config.c`.

References `json_decref_d`, `json_pack_ex_d`, `log_pedantic`, `portal_config_entry_flags()`, `st_char_get()`, and `st_length_int()`.

Referenced by `portal_config_collection()`.

`json_t* portal_config_entry_flags (user_config_entry_t * entry)`

Get a user config entry's flags as a json object.

Parameters:

entry a pointer to the user config entry object to have its flags serialized.

Returns:

NULL on failure or a pointer to a json object storing the user's flags on success.

Definition at line 20 of file config.c.

References `json_array_append_new_d`, `json_array_d`, `json_decref_d`, `json_string_d`, and `USER_CONF_STATUS_CRITICAL`.

Referenced by `portal_config_entry()`.

magma/objects/contacts/contacts.c File Reference

The interface to managing user address book contacts.

```
#include "magma.h"
```

Functions

- void **contact_free** (**contact_t** *contact)
- *Free a contact entry object.* void **contact_detail_free** (**contact_detail_t** *detail)
- *Free a contact entry detail.* **contact_t** * **contact_alloc** (uint64_t contactnum, **stringer_t** *name)
- *Create a new contact entry object.* **contact_detail_t** * **contact_detail_alloc** (**stringer_t** *key, **stringer_t** *value, uint64_t flags)
- *Create a new contact detail from a key-value pair.* void **contact_name** (**contact_t** *contact, **stringer_t** *name)
- *Update the name (in memory) of a contact entry.* **inx_t** * **contacts_update** (uint64_t usernum)
- *Retrieve all of a user's contacts (and contact folders) from the database.* **contact_t** * **contact_create** (uint64_t usernum, uint64_t foldernum, **stringer_t** *name)
- *Create a new contact entry object and insert it into the database.* **int_t** **contact_remove** (**contact_t** *contact, uint64_t usernum, uint64_t foldernum)
- *Remove a contact entry and its associated details from the database.* **int_t** **contact_move** (**contact_folder_t** *source, **contact_folder_t** *target, **contact_t** *contact, uint64_t usernum)
- *Move a contact entry from one contact to another.* **int_t** **contact_edit** (**contact_t** *contact, uint64_t usernum, uint64_t foldernum, **stringer_t** *key, **stringer_t** *value)
- *Change a detail of a contact entry.* **bool_t** **contact_validate_name** (**stringer_t** *name, **chr_t** **errmsg)
- *Validate the name of a requested contact entry name.* **bool_t** **contact_validate_detail** (**stringer_t** *key, **stringer_t** *value, **chr_t** **errmsg)

Validate the value of a requested contact entry detail.

Detailed Description

The interface to managing user address book contacts.

Definition in file **contacts.c**.

Function Documentation

contact_t* **contact_alloc** (uint64_t *contactnum*, **stringer_t** * *name*)

Create a new contact entry object.

contacts.c

Parameters:

contactnum the numerical id of the new contact entry.
name a managed string containing the name of the contact.

Returns:

NULL on failure, or a pointer to the newly allocated and initialized contact entry object on success.

Definition at line 53 of file contacts.c.

References [align\(\)](#), [contact_detail_free\(\)](#), [contact_t](#), [FOREIGNDATA](#), [inx_alloc\(\)](#), [JOINTED](#), [log_pedantic](#), [M_INX_TREE](#), [mm_alloc\(\)](#), [mm_copy\(\)](#), [mm_free\(\)](#), [PLACER_T](#), [placer_t](#), [st_data_get\(\)](#), [st_length_get\(\)](#), and [STACK](#).

Referenced by `contact_create()`, `contact_move()`, and `contacts_fetch()`.

`contact_t* contact_create (uint64_t usernum, uint64_t foldernum, stringer_t * name)`

Create a new contact entry object and insert it into the database.

Parameters:

usernum the numerical id of the user to whom the new contact will belong.

foldernum the numerical id of the parent folder that will contain the contact entry.

name a pointer to a managed string containing the name of the new contact entry.

Returns:

NULL on failure, or a pointer to the newly created contact entry object on success.

Definition at line 185 of file `contacts.c`.

References `contact_alloc()`, `contact_insert()`, `contact_t`, `log_pedantic`, and `st_empty()`.

Referenced by `portal_endpoint_contacts_add()`, and `portal_endpoint_contacts_copy()`.

`contact_detail_t* contact_detail_alloc (stringer_t * key, stringer_t * value, uint64_t flags)`

Create a new contact detail from a key-value pair.

Parameters:

key a managed string containing the name of the detail.

value a managed string containing the value associated with the detail.

flags a bitmask of flags associated with the detail.

Returns:

NULL on failure, or a pointer to the newly allocated and initialized contact detail object on success.

Definition at line 84 of file `contacts.c`.

References `align()`, `contact_detail_t`, `FOREIGNDATA`, `JOINTED`, `log_pedantic`, `mm_alloc()`, `mm_copy()`, `PLACER_T`, `placer_t`, `st_data_get()`, `st_length_get()`, and `STACK`.

Referenced by `contact_details_fetch()`, and `contact_edit()`.

`void contact_detail_free (contact_detail_t * detail)`

Free a contact entry detail.

Parameters:

detail a pointer to the contact entry detail to be freed.

Returns:

This function returns no value.

Definition at line 41 of file `contacts.c`.

References `mm_cleanup()`.

Referenced by `contact_alloc()`, `contact_details_fetch()`, and `contact_edit()`.

int_t contact_edit (contact_t * *contact*, uint64_t *usernum*, uint64_t *foldernum*, stringer_t * *key*, stringer_t * *value*)

Change a detail of a contact entry.

Parameters:

contact a pointer to the target contact entry.

usernum the numerical id of the user to whom the specified contact entry belongs.

foldernum

key a managed string containing the name of the contact entry detail to be changed. If the special value "name" is passed as the key, the contact name will be changed instead.

value a managed string containing the new value of the specified contact entry detail; if NULL is passed, the contact detail will be deleted rather than modified.

Returns:

-1 on failure or 1 on success.

LOW: It would be nice if we didn't call this function each time we updated a contact, but rather we called it when all of the updates have been completed.

LOW: It would be nice if we didn't call this function each time we updated a contact, but rather we called it when all of the updates have been completed.

Definition at line 274 of file contacts.c.

References `contact_detail_alloc()`, `contact_detail_delete()`, `contact_detail_free()`, `contact_detail_t`, `contact_detail_upsert()`, `contact_name()`, `contact_update()`, `contact_update_stamp()`, `inx_delete()`, `inx_replace()`, `log_pedantic`, `M_TYPE_STRINGER`, `PLACER`, `multi_t::st`, `st_cmp_ci_eq()`, `st_empty()`, and `multi_t::val`.

Referenced by `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, and `portal_endpoint_contacts_edit()`.

void contact_free (contact_t * *contact*)

Free a contact entry object.

Parameters:

contact a pointer to the contact entry to be freed.

Returns:

This function returns no value.

Definition at line 21 of file contacts.c.

References `FOREIGNDATA`, `inx_cleanup()`, `mm_free()`, `st_free()`, and `st_opt_get()`.

Referenced by `contact_folder_alloc()`, `contact_move()`, `contacts_fetch()`, `portal_endpoint_contacts_add()`, and `portal_endpoint_contacts_copy()`.

int_t contact_move (contact_folder_t * *source*, contact_folder_t * *target*, contact_t * *contact*, uint64_t *usernum*)

Move a contact entry from one contact to another.

Parameters:

source a pointer to the original parent contact folder object of the contact entry to be moved.
target a pointer to the contact folder object that will become the new parent of the specified contact entry.
contact a pointer to the contact object that is to be moved.
usernum the numerical id of the user to whom the specified contact entry belongs.

Returns:

-1 on error, or 1 if the contact move operation was successful.

Definition at line 229 of file contacts.c.

References `contact_alloc()`, `contact_free()`, `contact_t`, `contact_update()`, `inx_delete()`, `inx_insert()`, `log_pedantic`, `M_TYPE_UINT64`, `OBJECT_CONTACTS`, `serial_increment()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `portal_endpoint_contacts_move()`.

void contact_name (contact_t * *contact*, stringer_t * *name*)

Update the name (in memory) of a contact entry.

Parameters:

contact a pointer to the contact entry object to be updated.
name a managed string containing the new name of the specified contact entry.

Returns:

This function returns no value.

Definition at line 123 of file contacts.c.

References `data`, `FOREIGNDATA`, `mm_dupe()`, `mm_free()`, `st_data_get()`, `st_data_set()`, `st_empty_out()`, `st_length_set()`, `st_opt_get()`, and `st_opt_set()`.

Referenced by `contact_edit()`.

int_t contact_remove (contact_t * *contact*, uint64_t *usernum*, uint64_t *foldernum*)

Remove a contact entry and its associated details from the database.

See also:

`contact_delete()`

Note:

This function is not currently in use anywhere in the code.

Parameters:

contact a pointer to the contact entry to be deleted from the database.
usernum the numerical id of the user to whom the contact entry belongs.
foldernum the numerical id of the parent folder containing the specified contact entry.

Returns:

1 if the specified contact was deleted successfully, 0 if no matching contact was found in the database, or -1 on general failure.

Definition at line 211 of file contacts.c.

References `contact_delete()`, and `log_pedantic`.

bool_t contact_validate_detail (stringer_t * key, stringer_t * value, chr_t ** errmsg)

Validate the value of a requested contact entry detail.

Parameters:

key a pointer to a managed string containing the name of the contact detail key to be validated.
value a pointer to a managed string containing the value of the contact detail to be validated.
errmsg if not NULL, the address of a string pointer that will receive a descriptive error message upon validation failure.

Returns:

true if the proposed contact key/value pair was valid, or false if it was not.

Definition at line 373 of file contacts.c.

References chr_printable(), and st_empty_out().

Referenced by portal_endpoint_contacts_add().

bool_t contact_validate_name (stringer_t * name, chr_t ** errmsg)

Validate the name of a requested contact entry name.

Parameters:

name a pointer to a managed string containing the name of the contact to be validated.
errmsg if not NULL, the address of a string pointer that will receive a descriptive error message upon validation failure.

Returns:

true if the proposed contact name was valid, or false if it was not.

Definition at line 338 of file contacts.c.

References chr_printable(), and st_empty_out().

Referenced by portal_endpoint_contacts_add().

inx_t* contacts_update (uint64_t usernum)

Retrieve all of a user's contacts (and contact folders) from the database.

Parameters:

usenum the numerical id of the user whose contact folders and contact entries should be fetched.

Returns:

NULL on failure, or a pointer to an inx holder containing all of the specified user's contact folders and entries on success.

Definition at line 152 of file contacts.c.

References contacts_fetch(), inx_cleanup(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_free(), log_pedantic, M_FOLDER_CONTACTS, and magma_folder_fetch().

Referenced by meta_contacts_update().

magma/objects/folders/contacts.c File Reference

Interface for user contact folders.

```
#include "magma.h"
```

Functions

- void **contact_folder_free** (**contact_folder_t** *folder)
- *Free a contact folder object.* **contact_folder_t** * **contact_folder_alloc** (uint64_t foldernum, uint64_t parent, uint32_t order, **stringer_t** *name)
- *Allocate and initialize a new contact folder object.* **int_t** **contact_folder_create** (uint64_t usernum, uint64_t parent, **stringer_t** *name, **inx_t** *folders)
- *Create a new contact folder.* **int_t** **contact_folder_remove** (uint64_t usernum, uint64_t foldernum, **inx_t** *folders)
- *Remove a user's contact folder.* **bool_t** **contact_folder_rename** (uint64_t usernum, uint64_t foldernum, **stringer_t** *rename, **inx_t** *folders)

Rename a user's contact folder.

Detailed Description

Interface for user contact folders.

Definition in file **contacts.c**.

Function Documentation

contact_folder_t* **contact_folder_alloc** (uint64_t *foldernum*, uint64_t *parent*, uint32_t *order*,
stringer_t * *name*)

Allocate and initialize a new contact folder object.

contacts.c

Parameters:

foldernum the numerical id of this contact folder.
parent the numerical id of the parent folder of this contact folder.
order the order number of this folder in its parent folder.
name a managed string containing the name of this folder.

Returns:

NULL on failure, or a pointer to the newly initialized contact folder object on success.

Definition at line 34 of file contacts.c.

References **contact_free**(), **inx_alloc**(), **M_INX_TREE**, **magma_folder_alloc**(), and **mm_cleanup**().

Referenced by **contact_folder_create**(), and **magma_folder_funcs**().

int_t **contact_folder_create** (uint64_t *usernum*, uint64_t *parent*, **stringer_t** * *name*, **inx_t** *
folders)

Create a new contact folder.

Parameters:

usenum the numerical id of the user to whom the new imap folder will belong.
parent the numerical id of the parent folder to contain the new contact folder, or 0 for a root folder.
name a managed string containing the fully qualified name of the new contact folder to be created.
folders an inx holder containing the set of folders into which the new contact folder will be inserted.

Returns:

1 on success or <= 0 on failure. 0: An invalid parameter was passed to the function, or an internal failure occurred. -1: The specified new folder name was invalid. -2: Part of the folder path name exceeds 16 characters (FOLDER_LENGTH_LIMIT) after being unescaped for quotation marks. -3: The specified folder name already exists. -4: An invalid parent folder was specified. -5: Part of the folder path name exceeds 16 characters (FOLDER_LENGTH_LIMIT) after being unescaped for quotation marks.

Definition at line 60 of file contacts.c.

References `contact_folder_alloc()`, `contact_folder_free()`, `FOLDER_LENGTH_LIMIT`, `IMAP_FOLDER_RECURSION_LMIT`, `imap_valid_folder_name()`, `inx_insert()`, `log_pedantic`, `M_FOLDER_CONTACTS`, `M_TYPE_UINT64`, `magma_folder_children()`, `magma_folder_find_name()`, `magma_folder_find_number()`, `magma_folder_insert()`, `magma_folder_t`, `st_empty()`, `st_length_get()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `portal_folder_contacts_add()`.

void contact_folder_free (contact_folder_t * folder)

Free a contact folder object.

Parameters:

folder a pointer to the contact folder object to be freed.

Returns:

This function returns no value.

Definition at line 20 of file contacts.c.

References `magma_folder_free()`.

Referenced by `contact_folder_create()`, and `magma_folder_funcs()`.

int_t contact_folder_remove (uint64_t usenum, uint64_t foldernum, inx_t * folders)

Remove a user's contact folder.

See also:

`magma_folder_delete()`

Note:

This operation will only succeed if the specified contact folder is empty.

Parameters:

usenum the numerical id of the user requesting the contact folder removal.
foldernum the numerical id of the folder to be removed.
folders an inx holder containing the contact folders to be searched for the folder specified for removal.

Returns:

0 on success or < 0 on failure. -1: There was an unexpected internal error. -2: The specified folder could not be found. -3: Contact folder was not empty.

Definition at line 126 of file contacts.c.

References `inx_count()`, `inx_delete()`, `inx_find()`, `log_pedantic`, `M_FOLDER_CONTACTS`, `M_TYPE_UINT64`, `magma_folder_children()`, and `magma_folder_delete()`.

Referenced by `portal_folder_contacts_remove()`.

bool_t contact_folder_rename (uint64_t usernum, uint64_t foldernum, stringer_t * rename, inx_t * folders)

Rename a user's contact folder.

See also:

`magma_folder_rename()`

Parameters:

usernum the numerical id of the user requesting the contact folder renaming.

foldernum the numerical id of the folder to be renamed.

rename a managed string containing the new name of the specified folder.

folders an inx holder containing the contact folders to be searched for the folder specified to be renamed.

Returns:

true on success or false on failure.

Definition at line 166 of file contacts.c.

References `FOLDER_LENGTH_LIMIT`, `imap_valid_folder_name()`, `inx_find()`, `log_pedantic`, `M_FOLDER_CONTACTS`, `M_TYPE_UINT64`, `magma_folder_find_name()`, `magma_folder_rename()`, `st_dupe()`, `st_empty()`, `st_free()`, and `st_length_get()`.

Referenced by `portal_endpoint_folders_rename()`.

magma/objects/users/contacts.c File Reference

The user context interface for contacts.

```
#include "magma.h"
```

Functions

- **int_t meta_contacts_update** (meta_user_t *user, META_LOCK_STATUS locked)

Update a user's contacts if necessary.

Detailed Description

The user context interface for contacts.

Definition in file **contacts.c**.

Function Documentation

int_t meta_contacts_update (meta_user_t * *user*, META_LOCK_STATUS *locked*)

Update a user's contacts if necessary.

contacts.c

Note:

This function will try to retrieve the folders from the cache, if possible, or fall back to the database.

Parameters:

user a pointer to the meta user object that will have its message folders updated.

locked if META_NEED_LOCK is specified, the meta user object will be locked for operation.

Returns:

-1 on failure or 1 on success.

Definition at line 22 of file contacts.c.

References meta_user_t::contacts, contacts_update(), inx_free(), M_FOLDER_CONTACTS, META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), OBJECT_CONTACTS, serial_get(), serial_increment(), meta_user_t::serials, and meta_user_t::usernum.

Referenced by meta_get(), and sess_update().

magma/web/portal/contacts.c File Reference

Functions for handling user contact list entries.

```
#include "magma.h"
```

Functions

- `json_t * portal_contact_detail_flags (contact_detail_t *detail)`
- *TODO: Right now this function doesn't really do anything. It needs to be updated if/once flag values are determined.* `json_t * portal_contact_details (contact_t *contact)`

Retrieve all the details about a contact entry as a json object.

Detailed Description

Functions for handling user contact list entries.

Definition in file `contacts.c`.

Function Documentation

`json_t* portal_contact_detail_flags (contact_detail_t * detail)`

TODO: Right now this function doesn't really do anything. It needs to be updated if/once flag values are determined.

`contacts.c`

Pack a json array representing the flags associated with a user contact entry detail.

Parameters:

detail a pointer to the user contact detail to have its flags queried.

Returns:

NULL on failure or a pointer to a json array containing strings describing the specified contact detail's flags.

Definition at line 21 of file `contacts.c`.

References `CONTACT_DETAIL_FLAG_CRITICAL`, `json_array_append_new_d`, `json_array_d`, and `json_string_d`.

Referenced by `portal_contact_details()`.

`json_t* portal_contact_details (contact_t * contact)`

Retrieve all the details about a contact entry as a json object.

Parameters:

contact a pointer to the contact object whose details will be retrieved.

Returns:

a pointer to a json object containing the details of the specified contact entry.

Definition at line 40 of file contacts.c.

References `contact_detail_t`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `json_array_d`, `json_decref_d`, `json_object_set_new_d`, `json_pack_ex_d`, `log_error`, `log_pedantic`, `portal_contact_detail_flags()`, and `st_char_get()`.

Referenced by `portal_endpoint_contacts_load()`.

magma/objects/contacts/contacts.h File Reference

Interface for managing user address book contacts.

Enumerations

- enum { **CONTACT_DETAIL_FLAG_NONE** = 0, **CONTACT_DETAIL_FLAG_CRITICAL** = 1 }

Functions

- struct **__attribute__((__packed__))**
- contact_t** * **contact_alloc** (uint64_t contactnum, **stringer_t** *name)
- contacts.c* **contact_t** * **contact_create** (uint64_t usernum, uint64_t foldernum, **stringer_t** *name)
- Create a new contact entry object and insert it into the database. **contact_detail_t** * **contact_detail_alloc** (**stringer_t** *key, **stringer_t** *value, uint64_t flags)
- Create a new contact detail from a key-value pair. void **contact_detail_free** (**contact_detail_t** *detail)
- Free a contact entry detail. **int_t** **contact_edit** (**contact_t** *contact, uint64_t usernum, uint64_t foldernum, **stringer_t** *key, **stringer_t** *value)
- Change a detail of a contact entry. void **contact_free** (**contact_t** *contact)
- Free a contact entry object. **int_t** **contact_move** (**contact_folder_t** *source, **contact_folder_t** *target, **contact_t** *contact, uint64_t usernum)
- Move a contact entry from one contact to another. void **contact_name** (**contact_t** *contact, **stringer_t** *name)
- Update the name (in memory) of a contact entry. **int_t** **contact_remove** (**contact_t** *contact, uint64_t usernum, uint64_t foldernum)
- Remove a contact entry and its associated details from the database. **bool_t** **contact_validate_name** (**stringer_t** *name, **chr_t** **errmsg)
- Validate the name of a requested contact entry name. **bool_t** **contact_validate_detail** (**stringer_t** *key, **stringer_t** *value, **chr_t** **errmsg)
- Validate the value of a requested contact entry detail. **inx_t** * **contacts_update** (uint64_t usernum)
- Retrieve all of a user's contacts (and contact folders) from the database. **int_t** **contact_delete** (uint64_t contactnum, uint64_t usernum, uint64_t foldernum)
- datatier.c* **int_t** **contact_detail_delete** (uint64_t contactnum, **stringer_t** *key)
- Delete the specified contact detail of a contact entry from the database. **int_t** **contact_detail_upsert** (uint64_t contactnum, **stringer_t** *key, **stringer_t** *value, uint64_t flags)
- Update a specified contact detail in the database, or insert it if it does not exist. **int_t** **contact_details_fetch** (**contact_t** *contact)
- Populate a contact entry with its details from the database. uint64_t **contact_insert** (uint64_t usernum, uint64_t foldernum, **stringer_t** *name)
- Insert a new contact entry into the database. **int_t** **contact_update** (uint64_t contactnum, uint64_t usernum, uint64_t cur_folder, uint64_t target_folder, **stringer_t** *name)
- Update a user contact entry in the database. **int_t** **contact_update_stamp** (uint64_t contactnum, uint64_t usernum, uint64_t foldernum)
- Touch the last updated time stamp of a contact entry in the database. **int_t** **contacts_fetch** (uint64_t usernum, **contact_folder_t** *folder)
- Retrieve all of a user's contact entries in a specified contacts folder from the database. **contact_t** * **contact_find_detail** (**contact_folder_t** *folder, **stringer_t** *key, **stringer_t** *target)
- find.c* **contact_t** * **contact_find_name** (**contact_folder_t** *folder, **stringer_t** *target)
- Get a contact entry by name (case insensitive). **contact_t** * **contact_find_number** (**contact_folder_t** *folder, uint64_t target)

Find a contact entry in a contact folder by number. Variables

- contact_detail_t**
- contact_t**

Detailed Description

Interface for managing user address book contacts.

Definition in file **contacts.h**.

Enumeration Type Documentation

anonymous enum

Enumerator:

CONTACT_DETAIL_FLAG_NONE
CONTACT_DETAIL_FLAG_CRITICAL

Definition at line 16 of file contacts.h.

Function Documentation

struct __attribute__ ((__packed__)) [read]

Definition at line 26 of file contacts.h.

contact_t* contact_alloc (uint64_t *contactnum*, stringer_t * *name*)

contacts.c

contacts.c

Parameters:

contactnum the numerical id of the new contact entry.
name a managed string containing the name of the contact.

Returns:

NULL on failure, or a pointer to the newly allocated and initialized contact entry object on success.

Definition at line 53 of file contacts.c.

References align(), contact_detail_free(), contact_t, FOREIGNDATA, inx_alloc(), JOINTED, log_pedantic, M_INX_TREE, mm_alloc(), mm_copy(), mm_free(), PLACER_T, placer_t, st_data_get(), st_length_get(), and STACK.

Referenced by contact_create(), contact_move(), and contacts_fetch().

contact_t* contact_create (uint64_t *usernum*, uint64_t *foldernum*, stringer_t * *name*)

Create a new contact entry object and insert it into the database.

Parameters:

userid the numerical id of the user to whom the new contact will belong.
folderid the numerical id of the parent folder that will contain the contact entry.
name a pointer to a managed string containing the name of the new contact entry.

Returns:

NULL on failure, or a pointer to the newly created contact entry object on success.
 Definition at line 185 of file contacts.c.
 References `contact_alloc()`, `contact_insert()`, `contact_t`, `log_pedantic`, and `st_empty()`.
 Referenced by `portal_endpoint_contacts_add()`, and `portal_endpoint_contacts_copy()`.

int_t contact_delete (uint64_t *contactid*, uint64_t *userid*, uint64_t *folderid*)

datatier.c
 datatier.c

Parameters:

contactid the numerical id of the contact entry to be deleted.
userid the numerical id of the user to whom the specified contact entry belongs.
folderid the numerical id of the parent contact folder containing the contact entry.

Returns:

1 if the specified contact was deleted successfully, 0 if no matching contact was found in the database, or -1 on general failure.
 Definition at line 66 of file datatier.c.
 References `log_check`, `log_pedantic`, `mm_wipe()`, and `stmt_exec_affected()`.
 Referenced by `contact_remove()`, and `portal_endpoint_contacts_remove()`.

contact_detail_t* contact_detail_alloc (stringer_t * *key*, stringer_t * *value*, uint64_t *flags*)

Create a new contact detail from a key-value pair.

Parameters:

key a managed string containing the name of the detail.
value a managed string containing the value associated with the detail.
flags a bitmask of flags associated with the detail.

Returns:

NULL on failure, or a pointer to the newly allocated and initialized contact detail object on success.
 Definition at line 84 of file contacts.c.
 References `align()`, `contact_detail_t`, `FOREIGNDATA`, `JOINTED`, `log_pedantic`, `mm_alloc()`, `mm_copy()`, `PLACER_T`, `placer_t`, `st_data_get()`, `st_length_get()`, and `STACK`.
 Referenced by `contact_details_fetch()`, and `contact_edit()`.

int_t contact_detail_delete (uint64_t *contactid*, stringer_t * *key*)

Delete the specified contact detail of a contact entry from the database.

Parameters:

contactnum the numerical id of the contact entry to have the specified detail removed.
key a managed string containing the name of the contact detail to be removed from the entry.

Returns:

-1 on error, 0 if no matching detail was found in the database, or 1 if the delete operation was successful.

Definition at line 209 of file `datatier.c`.

References `log_check`, `log_pedantic`, `mm_wipe()`, `st_char_get()`, `st_length_get()`, `st_length_int()`, and `stmt_exec_affected()`.

Referenced by `contact_edit()`.

void contact_detail_free (contact_detail_t * *detail*)

Free a contact entry detail.

Parameters:

detail a pointer to the contact entry detail to be freed.

Returns:

This function returns no value.

Definition at line 41 of file `contacts.c`.

References `mm_cleanup()`.

Referenced by `contact_alloc()`, `contact_details_fetch()`, and `contact_edit()`.

int_t contact_detail_upsert (uint64_t *contactnum*, stringer_t * *key*, stringer_t * *value*, uint64_t *flags*)

Update a specified contact detail in the database, or insert it if it does not exist.

Parameters:

contactnum the numerical id of the contact entry to be modified.
key a managed string containing the name of the contact detail to be updated.
value a managed string containing the new value of the specified contact detail.
flags a bitmask of flags to be associated with the specified contact entry detail.

Returns:

-1 on error, 0 if no update was necessary, 1 if a new contact detail was inserted into the database, or 2 if the specified contact detail was updated successfully.

Definition at line 246 of file `datatier.c`.

References `log_check`, `log_pedantic`, `mm_wipe()`, `st_char_get()`, `st_length_get()`, `st_length_int()`, and `stmt_exec_affected()`.

Referenced by `contact_edit()`.

int_t contact_details_fetch (contact_t * *contact*)

Populate a contact entry with its details from the database.

Parameters:

contact a pointer to the contact entry object that will be updated.

Returns:

-1 on failure or 1 on success.

Definition at line 290 of file `datatier.c`.

References `contact_detail_alloc()`, `contact_detail_free()`, `contact_detail_t`, `inx_insert()`, `log_info`, `log_pedantic`, `M_TYPE_STRINGER`, `mm_wipe()`, `PLACER`, `res_field_block()`, `res_field_length()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `multi_t::st`, `stmt_get_result()`, and `multi_t::val`.

Referenced by `contacts_fetch()`.

```
int_t contact_edit (contact_t * contact,    uint64_t usernum,    uint64_t foldernum,    stringer_t *
key,    stringer_t * value)
```

Change a detail of a contact entry.

Parameters:

contact a pointer to the target contact entry.

usernum the numerical id of the user to whom the specified contact entry belongs.

foldernum

key a managed string containing the name of the contact entry detail to be changed. If the special value "name" is passed as the key, the contact name will be changed instead.

value a managed string containing the new value of the specified contact entry detail; if NULL is passed, the contact detail will be deleted rather than modified.

Returns:

-1 on failure or 1 on success.

LOW: It would be nice if we didn't call this function each time we updated a contact, but rather we called it when all of the updates have been completed.

LOW: It would be nice if we didn't call this function each time we updated a contact, but rather we called it when all of the updates have been completed.

Definition at line 274 of file `contacts.c`.

References `contact_detail_alloc()`, `contact_detail_delete()`, `contact_detail_free()`, `contact_detail_t`, `contact_detail_upsert()`, `contact_name()`, `contact_update()`, `contact_update_stamp()`, `inx_delete()`, `inx_replace()`, `log_pedantic`, `M_TYPE_STRINGER`, `PLACER`, `multi_t::st`, `st_cmp_ci_eq()`, `st_empty()`, and `multi_t::val`.

Referenced by `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, and `portal_endpoint_contacts_edit()`.

```
contact_t* contact_find_detail (contact_folder_t * folder,    stringer_t * key,    stringer_t * target)
```

`find.c`

Note:

This is a bit of a weird function that currently isn't being called from anywhere. It may or may not be here to stay.

Definition at line 18 of file `find.c`.

References `contact_detail_t`, `contact_t`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_find()`, `M_TYPE_STRINGER`, `st_cmp_ci_eq()`, and `st_empty()`.

`contact_t* contact_find_name (contact_folder_t * folder, stringer_t * target)`

Get a contact entry by name (case insensitive).

Parameters:

folder a pointer to the contact folder to have its contents searched for the entry.

target a managed string containing the name of the target entry.

Returns:

NULL on failure or a pointer to the found user contact entry on success.

Definition at line 55 of file `find.c`.

References `contact_t`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `st_cmp_ci_eq()`, and `st_empty()`.

`contact_t* contact_find_number (contact_folder_t * folder, uint64_t target)`

Find a contact entry in a contact folder by number.

Parameters:

folder a pointer to a contact folder object containing the contact entries to be searched.

target the numerical id of the contact entry to be located in the specified folder.

Returns:

NULL on failure or a pointer to the found user contact entry on success.

Definition at line 84 of file `find.c`.

References `contact_t`, `inx_find()`, `log_pedantic`, and `M_TYPE_UINT64`.

Referenced by `portal_endpoint_contacts_copy()`, `portal_endpoint_contacts_edit()`, `portal_endpoint_contacts_move()`, and `portal_endpoint_contacts_remove()`.

`void contact_free (contact_t * contact)`

Free a contact entry object.

Parameters:

contact a pointer to the contact entry to be freed.

Returns:

This function returns no value.

Definition at line 21 of file `contacts.c`.

References `FOREIGNDATA`, `inx_cleanup()`, `mm_free()`, `st_free()`, and `st_opt_get()`.

Referenced by `contact_folder_alloc()`, `contact_move()`, `contacts_fetch()`, `portal_endpoint_contacts_add()`, and `portal_endpoint_contacts_copy()`.

uint64_t contact_insert (uint64_t *usernum*, uint64_t *foldernum*, stringer_t * *name*)

Insert a new contact entry into the database.

Parameters:

usernum the numerical id of the user to whom the contact entry belongs.

foldernum the numerical id of the parent folder to contain the contact entry.

name a pointer to a managed string containing the name of the new contact entry.

Returns:

-1 on failure, 0 if no item was inserted into the database, or the id of the newly inserted contact entry in the database on success.

Definition at line 177 of file `datatier.c`.

References `mm_wipe()`, `st_char_get()`, `st_length_get()`, and `stmt_insert()`.

Referenced by `contact_create()`.

**int_t contact_move (contact_folder_t * *source*, contact_folder_t * *target*, contact_t * *contact*,
uint64_t *usernum*)**

Move a contact entry from one contact to another.

Parameters:

source a pointer to the original parent contact folder object of the contact entry to be moved.

target a pointer to the contact folder object that will become the new parent of the specified contact entry.

contact a pointer to the contact object that is to be moved.

usernum the numerical id of the user to whom the specified contact entry belongs.

Returns:

-1 on error, or 1 if the contact move operation was successful.

Definition at line 229 of file `contacts.c`.

References `contact_alloc()`, `contact_free()`, `contact_t`, `contact_update()`, `inx_delete()`, `inx_insert()`, `log_pedantic`, `M_TYPE_UINT64`, `OBJECT_CONTACTS`, `serial_increment()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `portal_endpoint_contacts_move()`.

void contact_name (contact_t * *contact*, stringer_t * *name*)

Update the name (in memory) of a contact entry.

Parameters:

contact a pointer to the contact entry object to be updated.

name a managed string containing the new name of the specified contact entry.

Returns:

This function returns no value.

Definition at line 123 of file `contacts.c`.

References `data`, `FOREIGNDATA`, `mm_dupe()`, `mm_free()`, `st_data_get()`, `st_data_set()`, `st_empty_out()`, `st_length_set()`, `st_opt_get()`, and `st_opt_set()`.

Referenced by `contact_edit()`.

int_t contact_remove (contact_t * *contact*, uint64_t *usernum*, uint64_t *foldernum*)

Remove a contact entry and its associated details from the database.

See also:

contact_delete()

Note:

This function is not currently in use anywhere in the code.

Parameters:

contact a pointer to the contact entry to be deleted from the database.

usernum the numerical id of the user to whom the contact entry belongs.

foldernum the numerical id of the parent folder containing the specified contact entry.

Returns:

1 if the specified contact was deleted successfully, 0 if no matching contact was found in the database, or -1 on general failure.

Definition at line 211 of file contacts.c.

References **contact_delete()**, and **log_pedantic**.

int_t contact_update (uint64_t *contactnum*, uint64_t *usernum*, uint64_t *cur_folder*, uint64_t *target_folder*, stringer_t * *name*)

Update a user contact entry in the database.

Parameters:

contactnum the numerical id of the contact entry to be modified.

usernum the numerical id of the user to whom the specified contact entry belongs.

cur_folder the numerical id of the parent contact containing the specified contact entry.

target_folder if not 0, sets the new parent contact folder to which the specified contact entry will belong.

name if not NULL, sets the new name of the specified contact entry.

Returns:

-1 on error, 0 if the specified contact entry was not found in the database, or 1 if the contact entry was successfully updated.

Definition at line 111 of file datatier.c.

References **ISNULL**, **log_check**, **log_pedantic**, **mm_wipe()**, **st_char_get()**, **st_empty()**, **st_length_get()**, and **stmt_exec_affected()**.

Referenced by **contact_edit()**, and **contact_move()**.

int_t contact_update_stamp (uint64_t *contactnum*, uint64_t *usernum*, uint64_t *foldernum*)

Touch the last updated time stamp of a contact entry in the database.

Parameters:

contactnum the numerical id of the contact entry to have its time stamp updated.

usernum the numerical id of the user to whom the contact entry belongs.

foldernum the numerical id of the parent folder containing the contact entry.

Returns:

1 if the contact time stamp was updated successfully, 0 if the specified contact was not found, or -1 on general failure.

Definition at line 22 of file `datatier.c`.

References `log_check`, `log_pedantic`, `mm_wipe()`, and `stmt_exec_affected()`.

Referenced by `contact_edit()`.

`bool_t contact_validate_detail (stringer_t * key, stringer_t * value, chr_t ** errmsg)`

Validate the value of a requested contact entry detail.

Parameters:

key a pointer to a managed string containing the name of the contact detail key to be validated.

value a pointer to a managed string containing the value of the contact detail to be validated.

errmsg if not NULL, the address of a string pointer that will receive a descriptive error message upon validation failure.

Returns:

true if the proposed contact key/value pair was valid, or false if it was not.

Definition at line 373 of file `contacts.c`.

References `chr_printable()`, and `st_empty_out()`.

Referenced by `portal_endpoint_contacts_add()`.

`bool_t contact_validate_name (stringer_t * name, chr_t ** errmsg)`

Validate the name of a requested contact entry name.

Parameters:

name a pointer to a managed string containing the name of the contact to be validated.

errmsg if not NULL, the address of a string pointer that will receive a descriptive error message upon validation failure.

Returns:

true if the proposed contact name was valid, or false if it was not.

Definition at line 338 of file `contacts.c`.

References `chr_printable()`, and `st_empty_out()`.

Referenced by `portal_endpoint_contacts_add()`.

`int_t contacts_fetch (uint64_t usernum, contact_folder_t * folder)`

Retrieve all of a user's contact entries in a specified contacts folder from the database.

Parameters:

usenum the numerical id of the user whose contacts will be retrieved.

folder a pointer to the contact folder which will have its contents listed.

Returns:

-1 on failure or 1 on success.

LOW: Should we bother with error checking?

Definition at line 344 of file `datatier.c`.

References `contact_alloc()`, `contact_details_fetch()`, `contact_free()`, `contact_t`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_insert()`, `log_info`, `log_pedantic`, `M_TYPE_UINT64`, `mm_wipe()`, `PLACER`, `res_field_block()`, `res_field_length()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `stmt_get_result()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `contacts_update()`.

`inx_t* contacts_update (uint64_t usernum)`

Retrieve all of a user's contacts (and contact folders) from the database.

Parameters:

usernum the numerical id of the user whose contact folders and contact entries should be fetched.

Returns:

NULL on failure, or a pointer to an `inx` holder containing all of the specified user's contact folders and entries on success.

Definition at line 152 of file `contacts.c`.

References `contacts_fetch()`, `inx_cleanup()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_free()`, `log_pedantic`, `M_FOLDER_CONTACTS`, and `magma_folder_fetch()`.

Referenced by `meta_contacts_update()`.

Variable Documentation**`contact_detail_t`**

Definition at line 24 of file `contacts.h`.

Referenced by `contact_detail_alloc()`, `contact_details_fetch()`, `contact_edit()`, `contact_find_detail()`, `portal_contact_details()`, `portal_endpoint_contacts_copy()`, and `portal_endpoint_contacts_list()`.

`contact_t`

Definition at line 30 of file `contacts.h`.

Referenced by `contact_alloc()`, `contact_create()`, `contact_find_detail()`, `contact_find_name()`, `contact_find_number()`, `contact_move()`, `contacts_fetch()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_contacts_edit()`, `portal_endpoint_contacts_list()`, `portal_endpoint_contacts_load()`, `portal_endpoint_contacts_move()`, and `portal_endpoint_contacts_remove()`.

magma/objects/contacts/find.c File Reference

Functions to search a collection of contacts.

```
#include "magma.h"
```

Functions

- **contact_t * contact_find_detail** (**contact_folder_t** *folder, **stringer_t** *key, **stringer_t** *target)
- *find.c* **contact_t * contact_find_name** (**contact_folder_t** *folder, **stringer_t** *target)
- *Get a contact entry by name (case insensitive).* **contact_t * contact_find_number** (**contact_folder_t** *folder, **uint64_t** target)

Find a contact entry in a contact folder by number.

Detailed Description

Functions to search a collection of contacts.

Definition in file **find.c**.

Function Documentation

contact_t* contact_find_detail (**contact_folder_t** * *folder*, **stringer_t** * *key*, **stringer_t** * *target*)

find.c

Note:

This is a bit of a weird function that currently isn't being called from anywhere. It may or may not be here to stay.

Definition at line 18 of file find.c.

References `contact_detail_t`, `contact_t`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_find()`, `M_TYPE_STRINGER`, `st_cmp_ci_eq()`, and `st_empty()`.

contact_t* contact_find_name (**contact_folder_t** * *folder*, **stringer_t** * *target*)

Get a contact entry by name (case insensitive).

Parameters:

folder a pointer to the contact folder to have its contents searched for the entry.

target a managed string containing the name of the target entry.

Returns:

NULL on failure or a pointer to the found user contact entry on success.

Definition at line 55 of file find.c.

References `contact_t`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `st_cmp_ci_eq()`, and `st_empty()`.

contact_t* contact_find_number (contact_folder_t * *folder*, uint64_t *target*)

Find a contact entry in a contact folder by number.

Parameters:

folder a pointer to a contact folder object containing the contact entries to be searched.
target the numerical id of the contact entry to be located in the specified folder.

Returns:

NULL on failure or a pointer to the found user contact entry on success.

Definition at line 84 of file find.c.

References contact_t, inx_find(), log_pedantic, and M_TYPE_UINT64.

Referenced by portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(), portal_endpoint_contacts_move(), and portal_endpoint_contacts_remove().

magma/objects/folders/find.c File Reference

Functions to search for specified folders.

```
#include "magma.h"
```

Functions

- **magma_folder_t * magma_folder_find_name** (inx_t *folders, stringer_t *target, uint64_t parent, bool_t check_inbox)
- *Get a magma folder by name.* **magma_folder_t * magma_folder_find_full_name** (inx_t *folders, stringer_t *target, bool_t check_inbox)
- *Get a magma folder by its fully qualified name.* **magma_folder_t * magma_folder_find_number** (inx_t *folders, uint64_t target)

Get a magma folder by number.

Detailed Description

Functions to search for specified folders.

Definition in file **find.c**.

Function Documentation

magma_folder_t* magma_folder_find_full_name (inx_t * *folders*, stringer_t * *target*, bool_t *check_inbox*)

Get a magma folder by its fully qualified name.

See also:

magma_folder_name()

Parameters:

folders an inx holder containing the collection of magma folders to be searched.

target a managed string containing the fully qualified (expanded) name of the magma folder to be found.

Returns:

NULL on failure, or a pointer to the found folder on success.

Definition at line 61 of file find.c.

References inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), magma_folder_name(), magma_folder_t, PLACER, st_cleanup(), st_cmp_ci_eq(), st_cmp_cs_eq(), and st_empty().

magma_folder_t* magma_folder_find_name (inx_t * *folders*, stringer_t * *target*, uint64_t *parent*, bool_t *check_inbox*)

Get a magma folder by name.

find.c

Parameters:

folders an inx holder containing the collection of magma folders to be searched.
target a managed string containing the name of the magma folder to be found.
parent the numerical id of the parent folder in which to search for the specified magma folder.
check_inbox if true, a case-insensitive comparison will be used if the specified folder name is "Inbox".

Returns:

NULL on failure, or a pointer to the found folder on success.

Definition at line 23 of file find.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `magma_folder_t`, `PLACER`, `st_cmp_ci_eq()`, `st_cmp_cs_eq()`, and `st_empty()`.

Referenced by `contact_folder_create()`, `contact_folder_rename()`, and `portal_folder_contacts_add()`.

magma_folder_t* magma_folder_find_number (inx_t * *folders*, uint64_t *target*)

Get a magma folder by number.

Parameters:

folders an inx holder containing the collection of magma folders to be searched.
target the numerical id of the magma folder to be found.

Returns:

NULL on failure, or a pointer to the found folder on success.

Definition at line 100 of file find.c.

References `inx_find()`, and `M_TYPE_UINT64`.

Referenced by `contact_folder_create()`, `magma_folder_name()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_contacts_edit()`, `portal_endpoint_contacts_list()`, `portal_endpoint_contacts_load()`, `portal_endpoint_contacts_move()`, `portal_endpoint_contacts_remove()`, `portal_endpoint_folders_rename()`, `portal_folder_contacts_add()`, and `portal_folder_contacts_remove()`.

magma/objects/folders/folders.c File Reference

Interface with user folders.

```
#include "magma.h"
```

Functions

- **stringer_t * magma_folder_name** (**inx_t** *folders, **magma_folder_t** *target)
- *TODO: The assumption is that message folders will start using the interface as well.* **int_t magma_folder_children** (**inx_t** *folders, **uint64_t** target)
- *Get the number of child folders a specified folder has.* **void magma_folder_free** (**magma_folder_t** *folder)
- *Free a magma folder object and its underlying records.* **magma_folder_t * magma_folder_alloc** (**uint64_t** foldernum, **uint64_t** parent, **uint32_t** order, **stringer_t** *name)
- *Create a new magma folder object.* **bool_t magma_folder_funcs** (**uint_t** type, **magma_folder_t** *(*folder_alloc)(**uint64_t**, **uint64_t**, **uint32_t**, **stringer_t** *), **void** (**folder_free)(**magma_folder_t** *))

Retrieve the appropriate folder management functions for message folders or contact folders.

Detailed Description

Interface with user folders.

Definition in file **folders.c**.

Function Documentation

magma_folder_t* magma_folder_alloc (**uint64_t** foldernum, **uint64_t** parent, **uint32_t** order, **stringer_t** * name)

Create a new magma folder object.

folders.c

Parameters:

foldernum the numerical id of this folder.

parent the folder id of this folder's containing parent folder.

order the order number of this folder in its parent folder.

name a managed string with the name of the target folder.

Returns:

NULL on failure or a pointer to the newly allocated magma folder object on success.

Definition at line 108 of file folders.c.

References `align()`, `FOREIGNDATA`, `JOINTED`, `log_pedantic`, `magma_folder_t`, `mm_alloc()`, `mm_copy()`, `mm_free()`, `PLACER_T`, `placer_t`, `rwlock_init()`, `st_data_get()`, `st_length_get()`, and `STACK`.

Referenced by `contact_folder_alloc()`, and `message_folder_alloc()`.

int_t magma_folder_children (**inx_t** * folders, **uint64_t** target)

Get the number of child folders a specified folder has.

Parameters:

folders an `inx` object holding all of the user's folders.
target the folder number of the target folder.

Returns:

the number of child folders contained by the specified folder, or 0 on failure.

Definition at line 62 of file `folders.c`.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, and `magma_folder_t`.

Referenced by `contact_folder_create()`, `contact_folder_remove()`, `message_folder_create()`, and `message_folder_remove()`.

void magma_folder_free (magma_folder_t * folder)

Free a magma folder object and its underlying records.

Returns:

This function returns no value.

Definition at line 89 of file `folders.c`.

References `inx_cleanup()`, `mm_free()`, and `rwlock_destroy()`.

Referenced by `contact_folder_free()`, and `message_folder_free()`.

bool_t magma_folder_funcs (uint_t type, magma_folder_t **)(uint64_t, uint64_t, uint32_t, stringer_t *) folder_alloc, void()(magma_folder_t *) folder_free)**

Retrieve the appropriate folder management functions for message folders or contact folders.

Parameters:

type the magma folder class of the desired management functions (`M_FOLDER_MESSAGES` or `M_FOLDER_CONTACTS`).

folder_alloc the address of a folder allocation function pointer that will be updated appropriately.

folder_free the address of a folder free function pointer that will be updated appropriately.

Returns:

true on success or false on failure.

Definition at line 141 of file `folders.c`.

References `contact_folder_alloc()`, `contact_folder_free()`, `M_FOLDER_CONTACTS`, `M_FOLDER_MESSAGES`, `message_folder_alloc()`, and `message_folder_free()`.

Referenced by `magma_folder_fetch()`.

stringer_t* magma_folder_name (inx_t * folders, magma_folder_t * target)

TODO: The assumption is that message folders will start using the interface as well.

Get the fully expanded name of a folder.

Note:

The folder hierarchy will be delimited with a "." character.

Parameters:

folders the inx object holding all of a user's magma folders.

target a pointer to the magma folder object to have its name expanded.

Returns:

NULL on failure or a pointer to a managed string with the fully expanded name of the specified folder.

Definition at line 24 of file folders.c.

References FOLDER_RECURSION_LIMIT, HEAP, JOINTED, log_pedantic, magma_folder_find_number(), MANAGED_T, PLACER, st_append, st_dupe_opts(), and st_empty().

Referenced by imap_list(), magma_folder_find_full_name(), and portal_endpoint_folders_rename().

magma/objects/users/folders.c File Reference

Functions used for handling message folders.

```
#include "magma.h"
```

Functions

- **meta_stats_tag_t * meta_folder_stats_tag_alloc** (stringer_t *tag)
- *Allocate a new meta tag stat object for tracking message tags.* **inx_t * meta_folders_stats_tags** (inx_t *messages, uint64_t folder)
- *Create a collection of meta tag stats for a set of messages.* **stringer_t * meta_folders_name** (inx_t *list, meta_folder_t *folder)
- *Get a folder's fully qualified name.* **meta_folder_t * meta_folders_by_name** (inx_t *folders, stringer_t *name)
- *Get a folder by its fully qualified name.* **meta_folder_t * meta_folders_by_number** (inx_t *folders, uint64_t number)
- *Get a folder by number.* **int_t meta_folders_children** (inx_t *folders, uint64_t number)
- *Get the number of direct child folders of a specified parent folder.* **int_t meta_folders_update** (meta_user_t *user, META_LOCK_STATUS locked)

Update a user's message folders if necessary.

Detailed Description

Functions used for handling message folders.

\$Author:\$ \$Author\$ \$Revision\$

Definition in file **folders.c**.

Function Documentation

meta_stats_tag_t * meta_folder_stats_tag_alloc (stringer_t * tag)

Allocate a new meta tag stat object for tracking message tags.

folders.c

Parameters:

tag a managed string containing the name of the message tag.

Returns:

NULL on failure, or a newly allocated and initialized meta tag stat object on success.

Definition at line 20 of file folders.c.

References align(), FOREIGNDATA, JOINTED, log_pedantic, mm_alloc(), mm_copy(), PLACER_T, placer_t, st_data_get(), st_length_get(), STACK, and meta_stats_tag_t::tag.

Referenced by meta_folders_stats_tags().

meta_folder_t * meta_folders_by_name (inx_t * folders, stringer_t * name)

Get a folder by its fully qualified name.

Parameters:

folders a pointer to an inx holder containing a list of folders to be traversed in the search.
folder a pointer to the meta folder object that is the leaf node of the folder path.

Returns:

NULL on failure or a managed string containing the fully qualified folder name on success.
 Definition at line 143 of file folders.c.

References `CONSTANT`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `meta_folders_name()`, `st_cmp_ci_eq()`, `st_cmp_cs_eq()`, and `st_free()`.

Referenced by `imap_append()`, `imap_copy()`, `imap_folder_create()`, `imap_folder_remove()`, `imap_folder_rename()`, `imap_folder_status()`, `imap_subscribe()`, and `portal_folder_mail_add()`.

meta_folder_t* meta_folders_by_number (inx_t * *folders*, uint64_t *number*)

Get a folder by number.

Parameters:

folders a pointer to an inx holder containing the folders to be searched.
number the numerical id of the folder to be retrieved.

Returns:

NULL on failure, or a pointer to the found meta folder object of the specified folder on success.
 Definition at line 184 of file folders.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, and `inx_cursor_value_next()`.

Referenced by `meta_folders_name()`, `portal_endpoint_folders_rename()`, `portal_endpoint_folders_tags()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_tag()`, `portal_folder_mail_add()`, and `portal_folder_mail_remove()`.

int_t meta_folders_children (inx_t * *folders*, uint64_t *number*)

Get the number of direct child folders of a specified parent folder.

Parameters:

folders a pointer to an inx holder containing the the collection of folders to be traversed.
number the folder id of the parent folder to be scanned for children.

Returns:

the number of children folders in the specified parent folder, or 0 on failure.
 Definition at line 212 of file folders.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, and `meta_folder_t::parent`.

Referenced by `imap_folder_remove()`.

stringer_t* meta_folders_name (inx_t * *list*, meta_folder_t * *folder*)

Get a folder's fully qualified name.

Note:

This function will prepend the entire ancestor path of a folder to its name, with each level delimited by periods.

Parameters:

list a pointer to an inx holder containing a list of folders to be searched for parent folders.

folder a pointer to the meta folder object that is the leaf node of the folder path.

Returns:

NULL on failure or a managed string containing the fully qualified folder name on success.

Definition at line 104 of file folders.c.

References FOLDER_RECURSION_LIMIT, log_pedantic, meta_folders_by_number(), meta_folder_t::name, ns_length_get(), meta_folder_t::parent, st_free(), st_import(), and st_merge.

Referenced by imap_folder_name_escaped(), imap_narrow_folders(), meta_folders_by_name(), portal_endpoint_folders_rename(), portal_folder_mail_add(), and portal_folder_mail_remove().

inx_t* meta_folders_stats_tags (inx_t * *messages*, uint64_t *folder*)

Create a collection of meta tag stats for a set of messages.

Note:

Each meta tag stat will contain the name of a message tag, along with the number of messages with which it was associated.

Parameters:

messages an inx holder containing the list of messages to be examined.

folder the numerical id of the parent folder that contains all target messages.

Returns:

NULL on failure, or an inx holder containing all of the messages' meta tag stats on success.

LOW: This is a very inefficient method for counting the number of times each tag appears.

Definition at line 45 of file folders.c.

References ar_field_st(), ar_length_get(), meta_stats_tag_t::count, meta_message_t::foldernum, inx_alloc(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_find(), inx_insert(), log_pedantic, M_INX_HASHED, M_TYPE_STRINGER, meta_folder_stats_tag_alloc(), mm_free(), multi_t::st, meta_message_t::tags, and multi_t::val.

Referenced by portal_endpoint_folders_tags().

int_t meta_folders_update (meta_user_t * *user*, META_LOCK_STATUS *locked*)

Update a user's message folders if necessary.

Note:

This function will try to retrieve the folders from the cache, if possible, or fall back to the database.

Parameters:

user a pointer to the meta user object that will have its message folders updated.

locked if META_NEED_LOCK is specified, the meta user object will be locked for operation.

Returns:

-1 on failure or 1 on success.

Definition at line 242 of file folders.c.

References meta_user_t::folders, meta_user_t::messages, meta_data_fetch_folders(), meta_messages_update_sequences(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), OBJECT_FOLDERS, meta_user_t::pop, meta_user_t::refs, serial_get(), serial_increment(), meta_user_t::serials, and meta_user_t::usernum.

Referenced by imap_session_update(), meta_get(), and sess_update().

magma/servers/imap/folders.c File Reference

Functions used to handle IMAP commands/actions.

```
#include "magma.h"
```

Functions

- **stringer_t * imap_folder_name_escaped** (inx_t *folders, meta_folder_t *active)
- *Get an imap folder's escaped name.* uint64_t **imap_next_folder_order** (inx_t *folders, uint64_t parent)
- *Get the next order value for an imap folder.* int_t **imap_folder_compare** (stringer_t *name, stringer_t *compare)
- **bool_t imap_valid_folder_name** (stringer_t *name)
- *Check to see if a name is a valid imap folder name.* int_t **imap_count_folder_levels** (stringer_t *name)
- *Get the highest node level indicated by an imap folder name.* int_t **imap_folder_create** (uint64_t usernum, inx_t *folders, stringer_t *name)
- *Create a new imap folder.* int_t **imap_folder_remove** (uint64_t usernum, inx_t *folders, inx_t *messages, stringer_t *name)
- *Remove an imap folder, both in memory and on the database.* int_t **imap_folder_rename** (uint64_t usernum, inx_t *folders, stringer_t *original, stringer_t *rename)
- *Rename an imap folder.* int_t **imap_folder_status** (inx_t *folders, inx_t *messages, stringer_t *name, imap_folder_status_t *status)
- *Get the status of a folder.* inx_t * **imap_narrow_folders** (inx_t *folders, stringer_t *reference, stringer_t *mailbox)

Detailed Description

Functions used to handle IMAP commands/actions.

Definition in file **folders.c**.

Function Documentation

int_t imap_count_folder_levels (stringer_t * *name*)

Get the highest node level indicated by an imap folder name.

folders.c

Note:

The smallest possible node level value is 1.

Parameters:

name a managed string containing the name of the imap folder.

Returns:

0 on failure, or the number of node levels indicated by the specified imap folder name, on success.

Definition at line 223 of file folders.c.

References IMAP_FOLDER_RECURSION_LMIIT, log_pedantic, and st_empty_out().

Referenced by imap_folder_create(), and imap_folder_rename().

int_t imap_folder_compare (stringer_t * *name*, stringer_t * *compare*)

Definition at line 96 of file folders.c.

References PLACER, st_char_get(), st_cmp_ci_eq(), and st_length_get().

Referenced by imap_narrow_folders().

int_t imap_folder_create (uint64_t *usernum*, inx_t * *folders*, stringer_t * *name*)

Create a new imap folder.

Note:

If they don't already exist, any parent folders specified in the fully qualified folder name will automatically be created first.

Parameters:

usernum the numerical id of the user to whom the new imap folder will belong.

folders an inx holder containing the set of folders into which the new imap folder will be inserted.

name a managed string containing the fully qualified name of the new imap folder to be created.

Returns:

1 on success or <= 0 on failure. 0: An invalid parameter was passed to the function, or an internal failure occurred. -1: The specified new folder name was invalid. -2: The node depth of the new folder is too large (imap folder recursion limit reached [IMAP_FOLDER_RECURSION_LMIIT]). -3: Special folder "Inbox" cannot contain subfolders. -4: Part of the folder path name exceeds 16 characters (FOLDER_LENGTH_LIMIT) after being unescaped for quotation marks. -5: The specified folder name already exists.

Definition at line 269 of file folders.c.

References CONTIGUOUS, FOLDER_LENGTH_LIMIT, meta_folder_t::foldernum, HEAP, imap_count_folder_levels(), IMAP_FOLDER_RECURSION_LMIIT, imap_next_folder_order(), imap_valid_folder_name(), inx_insert(), log_pedantic, M_TYPE_UINT64, MANAGED_T, meta_data_insert_folder(), meta_folders_by_name(), mm_alloc(), mm_copy(), mm_free(), meta_folder_t::name, meta_folder_t::order, meta_folder_t::parent, pl_data_get(), pl_length_get(), PLACER, placer_t, st_cleanup(), st_cmp_ci_eq(), st_dupe(), st_dupe_opts(), st_free(), st_merge, st_replace(), tok_get_st(), multi_t::u64, and multi_t::val.

Referenced by imap_create(), imap_subscribe(), and portal_folder_mail_add().

stringer_t* imap_folder_name_escaped (inx_t * *folders*, meta_folder_t * *active*)

Get an imap folder's escaped name.

TODO: Since the Portal needs access to some of these folder manipulation functions the common logic should be extracted and moved into the folder object interface.

Parameters:

folders an inx holder containing the ancestor folders of the specified imap folder.

active a pointer to the meta folder object of the folder to have its name escaped.

Returns:

NULL on failure, or a pointer to a managed string containing the escaped name of the specified imap folder on success.

Definition at line 24 of file folders.c.

References `meta_folders_name()`, `PLACER`, `st_free()`, `st_merge`, and `st_replace()`.

Referenced by `imap_list()`, and `imap_lsub()`.

int_t imap_folder_remove (uint64_t *usernum*, inx_t * *folders*, inx_t * *messages*, stringer_t * *name*)

Remove an imap folder, both in memory and on the database.

Note:

Any child messages of the specified folder will be deleted, but if it has child folders, the folder will not be deleted. The function also protects the special "Inbox" folder from deletion.

Parameters:

usernum the numerical id of the user that owns the imap folder to be deleted.

folders an inx holder containing a collection of folders for looking up the specified imap folder by name.

messages an inx holder containing a collection of messages for looking up the contents of the specified imap folder.

name a managed string containing the name of the imap folder to be deleted.

Returns:

<= 0 on failure, or 1 on success. 0: General failure or if there was an error removing the folder from the database. -1: The specified folder name was invalid. -2: Removal failed because the "Inbox" folder was specified. -3: The specified folder could not be found.

Definition at line 417 of file `folders.c`.

References `meta_folder_t::foldernum`, `meta_message_t::foldernum`, `imap_valid_folder_name()`,
`inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_delete()`, `log_error`, `log_pedantic`,
`M_TYPE_UINT64`, `mail_remove_message()`, `meta_message_t::messagenum`, `meta_data_delete_folder()`,
`meta_folders_by_name()`, `meta_folders_children()`, `PLACER`, `placer_t`, `meta_message_t::server`,
`meta_message_t::size`, `st_cmp_ci_eq()`, `tok_get_st()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `imap_delete()`, and `portal_folder_mail_remove()`.

int_t imap_folder_rename (uint64_t *usernum*, inx_t * *folders*, stringer_t * *original*, stringer_t * *rename*)

Rename an imap folder.

Note:

Any parent folder components of the folder's fully qualified name will be created if they do not already exist.

Parameters:

usernum the numerical id of the user requesting the folder renaming.

folders an inx holder containing the folders to be searched for the folder specified to be renamed.

original a managed string containing the original name of the folder to be renamed.

rename a managed string containing the new name of the specified folder.

Returns:

1 on success or 0 or less on failure. 0: One of the parameters passed to the function was invalid, or there was a memory allocation failure. -1: Either the original or new folder name was invalid. -2: Was unable to rename the "Inbox" folder. -3: The specified imap folder did not exist. -4: Either the original or new name exceeded the imap folder recursion limit. -5: A folder already exists with the new folder name. -6: A segment of the folder name was larger than `FOLDER_LENGTH_LIMIT` (16 bytes).

Definition at line 497 of file folders.c.

References CONTIGUOUS, FOLDER_LENGTH_LIMIT, meta_folder_t::foldernum, HEAP, imap_count_folder_levels(), IMAP_FOLDER_RECURSION_LIMIT, imap_next_folder_order(), imap_valid_folder_name(), inx_insert(), log_error, log_pedantic, M_TYPE_UINT64, MANAGED_T, meta_data_insert_folder(), meta_data_update_folder_name(), meta_folders_by_name(), mm_alloc(), mm_copy(), mm_free(), mm_wipe(), meta_folder_t::name, meta_folder_t::order, meta_folder_t::parent, pl_data_get(), pl_length_get(), PLACER, placer_t, st_cleanup(), st_cmp_ci_eq(), st_dupe(), st_dupe_opts(), st_free(), st_merge, st_replace(), tok_get_st(), multi_t::u64, and multi_t::val.

Referenced by imap_rename(), and portal_endpoint_folders_rename().

**int_t imap_folder_status (inx_t * *folders*, inx_t * *messages*, stringer_t * *name*,
imap_folder_status_t * *status*)**

Get the status of a folder.

Note:

This function will count the number of messages in a folder, as well as the number of messages marked recent or unseen, as well as the numerical id of the first message in the folder and the UIDNEXT of the specified folder.

Parameters:

folders an inx holder containing a list of folders to be searched for the specified folder.

messages an inx holder containing a complete list of a user's messages to be examined for gathering statistics.

name a managed string containing the name of the imap folder to be queried.

status a pointer to an imap folder status object to receive the folder's status information.

Returns:

1 on success or <= 0 on failure. 0: General failure. -1: The specified folder name was invalid. -2: The folder did not exist.

Definition at line 687 of file folders.c.

References imap_folder_status_t::first, meta_message_t::foldernum, meta_folder_t::foldernum, imap_folder_status_t::foldernum, imap_valid_folder_name(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), log_pedantic, MAIL_STATUS_RECENT, MAIL_STATUS_SEEN, meta_message_t::messagenum, imap_folder_status_t::messages, meta_folders_by_name(), mm_wipe(), imap_folder_status_t::recent, meta_message_t::status, imap_folder_status_t::uidnext, and imap_folder_status_t::unseen.

Referenced by imap_examine(), imap_select(), and imap_status().

inx_t* imap_narrow_folders (inx_t * *folders*, stringer_t * *reference*, stringer_t * *mailbox*)

Definition at line 752 of file folders.c.

References meta_folder_t::foldernum, imap_folder_compare(), inx_alloc(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_insert(), log_pedantic, M_INX_LINKED, M_TYPE_UINT64, meta_folders_name(), mm_dupe(), mm_free(), st_free(), st_length_get(), st_merge, multi_t::u64, and multi_t::val.

Referenced by imap_list(), and imap_lsub().

uint64_t imap_next_folder_order (inx_t * *folders*, uint64_t *parent*)

Get the next order value for an imap folder.

Note:

The next order value is the current highest order value of any child folder in the specified directory, incremented by one.

Parameters:

folder an `inx` holder containing the collection of imap folders to be scanned.
parent the numerical id of the folder to be queried.

Returns:

0 on failure, or the next order value of the specified folder on success.

Definition at line 72 of file `folders.c`.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `meta_folder_t::order`, and `meta_folder_t::parent`.

Referenced by `imap_folder_create()`, and `imap_folder_rename()`.

`bool_t imap_valid_folder_name (stringer_t * name)`

Check to see if a name is a valid imap folder name.

Note:

The following naming checks are enforced: 1. The name can't be empty or begin with a period. 2. It cannot contain a non-printable character. 3. Trailing periods will be removed. 4. The folder name cannot contain consecutive periods.

Parameters:

name a managed string containing the imap folder name to be validated.

Returns:

true on success or false on failure.

Definition at line 162 of file `folders.c`.

References `log_pedantic`, `st_empty_out()`, `st_length_get()`, and `st_length_set()`.

Referenced by `contact_folder_create()`, `contact_folder_rename()`, `imap_folder_create()`, `imap_folder_remove()`, `imap_folder_rename()`, and `imap_folder_status()`.

magma/web/portal/folders.c File Reference

Folder manipulation functions for use by the portal.

```
#include "magma.h"
```

Functions

- void **portal_folder_mail_add** (connection_t *con, stringer_t *name, uint64_t parent)
 - *Add a new mail folder for a user.* void **portal_folder_contacts_add** (connection_t *con, stringer_t *name, uint64_t parent)
 - *Add a new contact folder for a user.* void **portal_folder_mail_remove** (connection_t *con, uint64_t foldernum)
 - *Remove a user's mail folder.* void **portal_folder_contacts_remove** (connection_t *con, uint64_t foldernum)
- Remove a user's contacts folder.*
-

Detailed Description

Folder manipulation functions for use by the portal.

Definition in file **folders.c**.

Function Documentation

void portal_folder_contacts_add (connection_t * con, stringer_t * name, uint64_t parent)

Add a new contact folder for a user.

folders.c

Note:

If parent is 0, then the new folder is assumed to be a root folder. This function returns no value, but returns the appropriate portal json-rpc response directly.

Parameters:

con a pointer to the connection object across which the add folder response will be sent.

name a managed string containing the name of the new folder.

parent the numerical id of the folder that will be the parent of the new folder (or 0 to specify a root folder).

Returns:

This function returns no value.

Definition at line 86 of file folders.c.

References `contact_folder_create()`, `connection_t::http`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `magma_folder_find_name()`, `magma_folder_find_number()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_CONTACTS`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_FOLDERS_ADD`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `sess_serial_check()`, and `st_cleanup()`.

Referenced by `portal_endpoint_folders_add()`.

void portal_folder_contacts_remove (connection_t * con, uint64_t foldernum)

Remove a user's contacts folder.

Parameters:

con a pointer to the connection object across which the remove folder response will be sent.
foldernum the numerical id of the contacts folder that will be removed.

Returns:

This function returns no value.

Definition at line 205 of file folders.c.

References `contact_folder_remove()`, `connection_t::http`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `magma_folder_find_number()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_CONTACTS`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONSTRAINT_VIOLATION`, `PORTAL_ENDPOINT_ERROR_FOLDERS_REMOVE`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, and `sess_serial_check()`.

Referenced by `portal_endpoint_folders_remove()`.

void portal_folder_mail_add (connection_t * con, stringer_t * name, uint64_t parent)

Add a new mail folder for a user.

Note:

If parent is 0, then the new folder is assumed to be a root folder. This function returns no value, but returns the appropriate portal json-rpc response directly.

Parameters:

con a pointer to the connection object across which the add folder response will be sent.
name a managed string containing the name of the new folder.
parent the numerical id of the folder that will be the parent of the new folder (or 0 to specify a root folder).

Returns:

This function returns no value.

Definition at line 25 of file folders.c.

References `meta_folder_t::foldernum`, `connection_t::http`, `imap_folder_create()`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `meta_folders_by_name()`, `meta_folders_by_number()`, `meta_folders_name()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_FOLDERS`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_FOLDERS_ADD`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `sess_serial_check()`, `st_cleanup()`, and `st_merge()`.

Referenced by `portal_endpoint_folders_add()`.

void portal_folder_mail_remove (connection_t * con, uint64_t foldernum)

Remove a user's mail folder.

Parameters:

con a pointer to the connection object across which the remove folder response will be sent.
foldernum the numerical id of the mail folder that will be removed.

Returns:

This function returns no value.

Definition at line 152 of file folders.c.

References connection_t::http, imap_folder_remove(), JSON_RPC_2_ERROR_SERVER_INTERNAL,
meta_folders_by_number(), meta_folders_name(), meta_user_unlock(), meta_user_wlock(),
OBJECT_FOLDERS, portal_endpoint_error(),
PORTAL_ENDPOINT_ERROR_CONSTRAINT_VIOLATION,
PORTAL_ENDPOINT_ERROR_FOLDERS_REMOVE, PORTAL_ENDPOINT_ERROR_REFERENCE,
portal_endpoint_response(), sess_serial_check(), and st_cleanup().

Referenced by portal_endpoint_folders_remove().

magma/objects/folders/folders.h File Reference

Interface to managing user folders.

Defines

- #define **FOLDER_LENGTH_LIMIT** 16
- #define **FOLDER_RECURSION_LIMIT** 16

Typedefs

- typedef **magma_folder_t** **contact_folder_t**
- typedef **magma_folder_t** **message_folder_t**

Enumerations

- enum { **M_FOLDER_MESSAGES** = 1, **M_FOLDER_CONTACTS** = 2 }

Functions

- struct **__attribute__((__packed__))**
- **message_folder_t** * **message_folder_alloc** (uint64_t foldernum, uint64_t parent, uint32_t order, **stringer_t** *name)
- *messages.c* **int_t** **message_folder_create** (uint64_t usernum, uint64_t parent, **stringer_t** *name, **inx_t** *folders)
- void **message_folder_free** (**message_folder_t** *folder)
- *Free a message folder object.* **bool_t** **message_folder_remove** (uint64_t usernum, uint64_t foldernum, **inx_t** *folders)
- *Remove a user's message folder.* **contact_folder_t** * **contact_folder_alloc** (uint64_t foldernum, uint64_t parent, uint32_t order, **stringer_t** *name)
- *contacts.c* **int_t** **contact_folder_create** (uint64_t usernum, uint64_t parent, **stringer_t** *name, **inx_t** *folders)
- *Create a new contact folder.* void **contact_folder_free** (**contact_folder_t** *folder)
- *Free a contact folder object.* **int_t** **contact_folder_remove** (uint64_t usernum, uint64_t foldernum, **inx_t** *folders)
- *Remove a user's contact folder.* **bool_t** **contact_folder_rename** (uint64_t usernum, uint64_t foldernum, **stringer_t** *rename, **inx_t** *folders)
- *Rename a user's contact folder.* **bool_t** **magma_folder_delete** (uint64_t usernum, uint64_t foldernum, **uint_t** type)
- *datatier.c* **inx_t** * **magma_folder_fetch** (uint64_t usernum, **uint_t** type)
- *Fetch all of a user's folders of a specified type from the database.* uint64_t **magma_folder_insert** (uint64_t usernum, **stringer_t** *name, uint64_t parent, uint32_t order, **uint_t** type)
- *Insert a new folder into the database.* **bool_t** **magma_folder_rename** (uint64_t usernum, uint64_t foldernum, **uint_t** type, **stringer_t** *rename)
- *Rename a folder in the database.* **magma_folder_t** * **magma_folder_find_name** (**inx_t** *folders, **stringer_t** *target, uint64_t parent, **bool_t** check_inbox)
- *find.c* **magma_folder_t** * **magma_folder_find_full_name** (**inx_t** *folders, **stringer_t** *target, **bool_t** check_inbox)
- *Get a magma folder by its fully qualified name.* **magma_folder_t** * **magma_folder_find_number** (**inx_t** *folders, uint64_t target)
- *Get a magma folder by number.* **magma_folder_t** * **magma_folder_alloc** (uint64_t foldernum, uint64_t parent, uint32_t order, **stringer_t** *name)
- *folders.c* **int_t** **magma_folder_children** (**inx_t** *folders, uint64_t target)
- *Get the number of child folders a specified folder has.* void **magma_folder_free** (**magma_folder_t** *folder)

- *Free a magma folder object and its underlying records.* **bool_t magma_folder_funcs** (uint_t type, **magma_folder_t** *(*folder_alloc)(uint64_t, uint64_t, uint32_t, **stringer_t** *), void(**folder_free)(**magma_folder_t** *))
- *Retrieve the appropriate folder management functions for message folders or contact folders.* **stringer_t** * **magma_folder_name** (inx_t *folders, **magma_folder_t** *target)

TODO: The assumption is that message folders will start using the interface as well. Variables

- **magma_folder_t**

Detailed Description

Interface to managing user folders.

Definition in file **folders.h**.

Define Documentation

#define FOLDER_LENGTH_LIMIT 16

Definition at line 16 of file folders.h.

Referenced by `contact_folder_create()`, `contact_folder_rename()`, `imap_create()`, `imap_folder_create()`, `imap_folder_rename()`, `imap_rename()`, and `imap_subscribe()`.

#define FOLDER_RECURSION_LIMIT 16

Definition at line 17 of file folders.h.

Referenced by `magma_folder_name()`, and `meta_folders_name()`.

Typedef Documentation

typedef magma_folder_t contact_folder_t

Definition at line 35 of file folders.h.

typedef magma_folder_t message_folder_t

Definition at line 36 of file folders.h.

Enumeration Type Documentation

anonymous enum

Enumerator:

M_FOLDER_MESSAGES M_FOLDER_MESSAGES.

M_FOLDER_CONTACTS M_FOLDER_CONTACTS.

Definition at line 22 of file folders.h.

Function Documentation

struct __attribute__((__packed__)) [read]

Definition at line 27 of file folders.h.

References lock.

**contact_folder_t* contact_folder_alloc (uint64_t *foldernum*, uint64_t *parent*, uint32_t *order*,
stringer_t * *name*)**

contacts.c

contacts.c

Parameters:

foldernum the numerical id of this contact folder.
parent the numerical id of the parent folder of this contact folder.
order the order number of this folder in its parent folder.
name a managed string containing the name of this folder.

Returns:

NULL on failure, or a pointer to the newly initialized contact folder object on success.

Definition at line 34 of file contacts.c.

References contact_free(), inx_alloc(), M_INX_TREE, magma_folder_alloc(), and mm_cleanup().

Referenced by contact_folder_create(), and magma_folder_funcs().

**int_t contact_folder_create (uint64_t *usernum*, uint64_t *parent*, stringer_t * *name*, inx_t *
folders)**

Create a new contact folder.

Parameters:

usernum the numerical id of the user to whom the new imap folder will belong.
parent the numerical id of the parent folder to contain the new contact folder, or 0 for a root folder.
name a managed string containing the fully qualified name of the new contact folder to be created.
folders an inx holder containing the set of folders into which the new contact folder will be inserted.

Returns:

1 on success or <= 0 on failure. 0: An invalid parameter was passed to the function, or an internal failure occurred. -1: The specified new folder name was invalid. -2: Part of the folder path name exceeds 16 characters (FOLDER_LENGTH_LIMIT) after being unescaped for quotation marks. -3: The specified folder name already exists. -4: An invalid parent folder was specified. -5: Part of the folder path name exceeds 16 characters (FOLDER_LENGTH_LIMIT) after being unescaped for quotation marks.

Definition at line 60 of file contacts.c.

References `contact_folder_alloc()`, `contact_folder_free()`, `FOLDER_LENGTH_LIMIT`, `IMAP_FOLDER_RECURSION_LMIT`, `imap_valid_folder_name()`, `inx_insert()`, `log_pedantic`, `M_FOLDER_CONTACTS`, `M_TYPE_UINT64`, `magma_folder_children()`, `magma_folder_find_name()`, `magma_folder_find_number()`, `magma_folder_insert()`, `magma_folder_t`, `st_empty()`, `st_length_get()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `portal_folder_contacts_add()`.

void contact_folder_free (contact_folder_t * folder)

Free a contact folder object.

Parameters:

folder a pointer to the contact folder object to be freed.

Returns:

This function returns no value.

Definition at line 20 of file contacts.c.

References `magma_folder_free()`.

Referenced by `contact_folder_create()`, and `magma_folder_funcs()`.

int_t contact_folder_remove (uint64_t usernum, uint64_t foldernum, inx_t * folders)

Remove a user's contact folder.

See also:

`magma_folder_delete()`

Note:

This operation will only succeed if the specified contact folder is empty.

Parameters:

usernum the numerical id of the user requesting the contact folder removal.

foldernum the numerical id of the folder to be removed.

folders an inx holder containing the contact folders to be searched for the folder specified for removal.

Returns:

0 on success or < 0 on failure. -1: There was an unexpected internal error. -2: The specified folder could not be found. -3: Contact folder was not empty.

Definition at line 126 of file contacts.c.

References `inx_count()`, `inx_delete()`, `inx_find()`, `log_pedantic`, `M_FOLDER_CONTACTS`, `M_TYPE_UINT64`, `magma_folder_children()`, and `magma_folder_delete()`.

Referenced by `portal_folder_contacts_remove()`.

**bool_t contact_folder_rename (uint64_t *usernum*, uint64_t *foldernum*, stringer_t * *rename*,
inx_t * *folders*)**

Rename a user's contact folder.

See also:

magma_folder_rename()

Parameters:

usernum the numerical id of the user requesting the contact folder renaming.

foldernum the numerical id of the folder to be renamed.

rename a managed string containing the new name of the specified folder.

folders an inx holder containing the contact folders to be searched for the folder specified to be renamed.

Returns:

true on success or false on failure.

Definition at line 166 of file contacts.c.

References FOLDER_LENGTH_LIMIT, imap_valid_folder_name(), inx_find(), log_pedantic, M_FOLDER_CONTACTS, M_TYPE_UINT64, magma_folder_find_name(), magma_folder_rename(), st_dupe(), st_empty(), st_free(), and st_length_get().

Referenced by portal_endpoint_folders_rename().

**magma_folder_t* magma_folder_alloc (uint64_t *foldernum*, uint64_t *parent*, uint32_t *order*,
stringer_t * *name*)**

folders.c

folders.c

Parameters:

foldernum the numerical id of this folder.

parent the folder id of this folder's containing parent folder.

order the order number of this folder in its parent folder.

name a managed string with the name of the target folder.

Returns:

NULL on failure or a pointer to the newly allocated magma folder object on success.

Definition at line 108 of file folders.c.

References align(), FOREIGNDATA, JOINTED, log_pedantic, magma_folder_t, mm_alloc(), mm_copy(), mm_free(), PLACER_T, placer_t, rwlock_init(), st_data_get(), st_length_get(), and STACK.

Referenced by contact_folder_alloc(), and message_folder_alloc().

int_t magma_folder_children (inx_t * *folders*, uint64_t *target*)

Get the number of child folders a specified folder has.

Parameters:

folders an inx object holding all of the user's folders.

target the folder number of the target folder.

Returns:

the number of child folders contained by the specified folder, or 0 on failure.

Definition at line 62 of file folders.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, and `magma_folder_t`.

Referenced by `contact_folder_create()`, `contact_folder_remove()`, `message_folder_create()`, and `message_folder_remove()`.

`bool_t magma_folder_delete (uint64_t usernum, uint64_t foldernum, uint_t type)`

`datatier.c`

`datatier.c`

Parameters:

usernum the numerical id of the user requesting the folder removal.

foldernum the numerical id of the folder to be removed.

type the type of the folder to be deleted (can be `M_FOLDER_MESSAGES` or `M_FOLDER_CONTACTS`).

Returns:

true on success or false on failure.

Definition at line 141 of file `datatier.c`.

References `mm_wipe()`, and `stmt_exec_affected()`.

Referenced by `contact_folder_remove()`, and `message_folder_remove()`.

`inx_t* magma_folder_fetch (uint64_t usernum, uint_t type)`

Fetch all of a user's folders of a specified type from the database.

Parameters:

usernum the numerical id of the requested user.

type the folder class of the folders to be retrieved (can be `M_FOLDER_MESSAGES` or `M_FOLDER_CONTACTS`).

Returns:

NULL on failure, or an `inx` object holding all of the requested folders on success.

Definition at line 21 of file `datatier.c`.

References `inx_alloc()`, `inx_free()`, `inx_insert()`, `log_info`, `log_pedantic`, `M_INX_TREE`, `M_TYPE_UINT64`, `magma_folder_funcs()`, `magma_folder_t`, `mm_wipe()`, `PLACER`, `res_field_block()`, `res_field_length()`, `res_field_uint32()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `stmt_get_result()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `contacts_update()`, and `messages_update()`.

`magma_folder_t* magma_folder_find_full_name (inx_t * folders, stringer_t * target, bool_t check_inbox)`

Get a magma folder by its fully qualified name.

See also:

`magma_folder_name()`

Parameters:

folders an inx holder containing the collection of magma folders to be searched.

target a managed string containing the fully qualified (expanded) name of the magma folder to be found.

Returns:

NULL on failure, or a pointer to the found folder on success.

Definition at line 61 of file find.c.

References inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), magma_folder_name(), magma_folder_t, PLACER, st_cleanup(), st_cmp_ci_eq(), st_cmp_cs_eq(), and st_empty().

magma_folder_t* magma_folder_find_name (inx_t * *folders*, stringer_t * *target*, uint64_t *parent*, bool_t *check_inbox*)

find.c

find.c

Parameters:

folders an inx holder containing the collection of magma folders to be searched.

target a managed string containing the name of the magma folder to be found.

parent the numerical id of the parent folder in which to search for the specified magma folder.

check_inbox if true, a case-insensitive comparison will be used if the specified folder name is "Inbox".

Returns:

NULL on failure, or a pointer to the found folder on success.

Definition at line 23 of file find.c.

References inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), magma_folder_t, PLACER, st_cmp_ci_eq(), st_cmp_cs_eq(), and st_empty().

Referenced by contact_folder_create(), contact_folder_rename(), and portal_folder_contacts_add().

magma_folder_t* magma_folder_find_number (inx_t * *folders*, uint64_t *target*)

Get a magma folder by number.

Parameters:

folders an inx holder containing the collection of magma folders to be searched.

target the numerical id of the magma folder to be found.

Returns:

NULL on failure, or a pointer to the found folder on success.

Definition at line 100 of file find.c.

References inx_find(), and M_TYPE_UINT64.

Referenced by contact_folder_create(), magma_folder_name(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(), portal_endpoint_contacts_list(), portal_endpoint_contacts_load(), portal_endpoint_contacts_move(), portal_endpoint_contacts_remove(), portal_endpoint_folders_rename(), portal_folder_contacts_add(), and portal_folder_contacts_remove().

void magma_folder_free (magma_folder_t * *folder*)

Free a magma folder object and its underlying records.

Returns:

This function returns no value.

Definition at line 89 of file folders.c.

References `inx_cleanup()`, `mm_free()`, and `rwlock_destroy()`.

Referenced by `contact_folder_free()`, and `message_folder_free()`.

bool_t magma_folder_funcs (uint_t type, magma_folder_t **)(uint64_t, uint64_t, uint32_t, stringer_t *) folder_alloc, void()(magma_folder_t *) folder_free)**

Retrieve the appropriate folder management functions for message folders or contact folders.

Parameters:

type the magma folder class of the desired management functions (M_FOLDER_MESSAGES or M_FOLDER_CONTACTS).

folder_alloc the address of a folder allocation function pointer that will be updated appropriately.

folder_free the address of a folder free function pointer that will be updated appropriately.

Returns:

true on success or false on failure.

Definition at line 141 of file folders.c.

References `contact_folder_alloc()`, `contact_folder_free()`, `M_FOLDER_CONTACTS`, `M_FOLDER_MESSAGES`, `message_folder_alloc()`, and `message_folder_free()`.

Referenced by `magma_folder_fetch()`.

uint64_t magma_folder_insert (uint64_t usernum, stringer_t * name, uint64_t parent, uint32_t order, uint_t type)

Insert a new folder into the database.

Parameters:

usenum the numerical id of the user to whom the new folder belongs.

name a managed string containing the name of the new folder.

parent the numerical id of the mail folder to be the parent of the new mail folder.

order the order number of this folder in its parent folder.

type the type of the new folder (can be M_FOLDER_MESSAGES or M_FOLDER_CONTACTS).

Returns:

0 on failure, or the numerical id of the newly inserted folder in the database on success.

Definition at line 96 of file datatier.c.

References `mm_wipe()`, `st_char_get()`, `st_length_get()`, and `stmt_insert()`.

Referenced by `contact_folder_create()`, and `message_folder_create()`.

stringer_t* magma_folder_name (inx_t * folders, magma_folder_t * target)

TODO: The assumption is that message folders will start using the interface as well.

Get the fully expanded name of a folder.

Note:

The folder hierarchy will be delimited with a "." character.

Parameters:

folders the inx object holding all of a user's magma folders.

target a pointer to the magma folder object to have its name expanded.

Returns:

NULL on failure or a pointer to a managed string with the fully expanded name of the specified folder.

Definition at line 24 of file folders.c.

References FOLDER_RECURSION_LIMIT, HEAP, JOINTED, log_pedantic, magma_folder_find_number(), MANAGED_T, PLACER, st_append, st_dupe_opts(), and st_empty().

Referenced by imap_list(), magma_folder_find_full_name(), and portal_endpoint_folders_rename().

**bool_t magma_folder_rename (uint64_t *usernum*, uint64_t *foldernum*, uint_t *type*,
stringer_t * *rename*)**

Rename a folder in the database.

Parameters:

usernum the numerical id of the user requesting the folder renaming.

foldernum the numerical id of the folder to be renamed.

type the type of the folder to be renamed (can be M_FOLDER_MESSAGES or M_FOLDER_CONTACTS).

rename a managed string containing the new name of the specified folder.

Returns:

true on success or false on failure.

Definition at line 181 of file datatier.c.

References mm_wipe(), st_char_get(), st_length_get(), and stmt_exec_affected().

Referenced by contact_folder_rename().

**message_folder_t* message_folder_alloc (uint64_t *foldernum*, uint64_t *parent*, uint32_t *order*,
stringer_t * *name*)**

messages.c

messages.c

Parameters:

foldernum the numerical id of the new message folder.

parent the numerical id of the parent folder containing the target message folder.

order the order number of this folder in its parent folder.

name a managed string containing the name of the new message folder.

Returns:

NULL on failure or a pointer to the newly allocated and initialized message folder object on success.

Definition at line 33 of file messages.c.

References inx_alloc(), M_INX_TREE, magma_folder_alloc(), message_free(), and mm_cleanup().

Referenced by magma_folder_funcs(), and message_folder_create().

int_t message_folder_create (uint64_t *usernum*, uint64_t *parent*, stringer_t * *name*, inx_t * *folders*)

TODO: Add validation: check whether the parent exists, and enforce the length and depth limits. Also make sure the name doesn't conflict with any built in folder names. Finally, setup a list of error codes and return the appropriate value.

Note:

This function isn't called from anywhere else at the moment.

Definition at line 50 of file messages.c.

References inx_insert(), log_pedantic, M_FOLDER_MESSAGES, M_TYPE_UINT64, magma_folder_children(), magma_folder_insert(), message_folder_alloc(), message_folder_free(), st_empty(), multi_t::u64, and multi_t::val.

void message_folder_free (message_folder_t * *folder*)

Free a message folder object.

Returns:

This function returns no value.

Definition at line 19 of file messages.c.

References magma_folder_free().

Referenced by magma_folder_funcs(), and message_folder_create().

bool_t message_folder_remove (uint64_t *usernum*, uint64_t *foldernum*, inx_t * *folders*)

Remove a user's message folder.

See also:

magma_folder_delete()

Note:

This operation will only succeed if the specified message folder is empty.

Parameters:

usernum the numerical id of the user requesting the message folder removal.

foldernum the numerical id of the folder to be removed.

folders an inx holder containing the message folders to be searched for the folder specified for removal.

Returns:

true on success or false on failure.

Definition at line 86 of file messages.c.

References inx_count(), inx_delete(), inx_find(), log_pedantic, M_FOLDER_MESSAGES, M_TYPE_UINT64, magma_folder_children(), and magma_folder_delete().

Variable Documentation

magma_folder_t

Definition at line 33 of file folders.h.

Referenced by contact_folder_create(), magma_folder_alloc(), magma_folder_children(),
magma_folder_fetch(), magma_folder_find_full_name(), magma_folder_find_name(),
portal_endpoint_folders_list(), and portal_endpoint_folders_rename().

magma/objects/folders/messages.c File Reference

Interface with user message folders.

```
#include "magma.h"
```

Functions

- void **message_folder_free** (**message_folder_t** *folder)
- *Free a message folder object.* **message_folder_t** * **message_folder_alloc** (uint64_t foldernum, uint64_t parent, uint32_t order, **stringer_t** *name)
- *Create and initialize a new message folder object.* **int_t** **message_folder_create** (uint64_t usernum, uint64_t parent, **stringer_t** *name, **inx_t** *folders)
- **bool_t** **message_folder_remove** (uint64_t usernum, uint64_t foldernum, **inx_t** *folders)

Remove a user's message folder.

Detailed Description

Interface with user message folders.

Definition in file **messages.c**.

Function Documentation

message_folder_t* **message_folder_alloc** (uint64_t *foldernum*, uint64_t *parent*, uint32_t *order*,
stringer_t * *name*)

Create and initialize a new message folder object.

messages.c

Parameters:

foldernum the numerical id of the new message folder.

parent the numerical id of the parent folder containing the target message folder.

order the order number of this folder in its parent folder.

name a managed string containing the name of the new message folder.

Returns:

NULL on failure or a pointer to the newly allocated and initialized message folder object on success.

Definition at line 33 of file messages.c.

References `inx_alloc()`, `M_INX_TREE`, `magma_folder_alloc()`, `message_free()`, and `mm_cleanup()`.

Referenced by `magma_folder_funcs()`, and `message_folder_create()`.

int_t **message_folder_create** (uint64_t *usernum*, uint64_t *parent*, **stringer_t** * *name*, **inx_t** *
folders)

TODO: Add validation: check whether the parent exists, and enforce the length and depth limits. Also make sure the name doesn't conflict with any built in folder names. Finally, setup a list of error codes and return the appropriate value.

Note:

This function isn't called from anywhere else at the moment.

Definition at line 50 of file messages.c.

References `inx_insert()`, `log_pedantic`, `M_FOLDER_MESSAGES`, `M_TYPE_UINT64`, `magma_folder_children()`, `magma_folder_insert()`, `message_folder_alloc()`, `message_folder_free()`, `st_empty()`, `multi_t::u64`, and `multi_t::val`.

void message_folder_free (message_folder_t * *folder*)

Free a message folder object.

Returns:

This function returns no value.

Definition at line 19 of file messages.c.

References `magma_folder_free()`.

Referenced by `magma_folder_funcs()`, and `message_folder_create()`.

bool_t message_folder_remove (uint64_t *usernum*, uint64_t *foldernum*, inx_t * *folders*)

Remove a user's message folder.

See also:

`magma_folder_delete()`

Note:

This operation will only succeed if the specified message folder is empty.

Parameters:

usernum the numerical id of the user requesting the message folder removal.

foldernum the numerical id of the folder to be removed.

folders an inx holder containing the message folders to be searched for the folder specified for removal.

Returns:

true on success or false on failure.

Definition at line 86 of file messages.c.

References `inx_count()`, `inx_delete()`, `inx_find()`, `log_pedantic`, `M_FOLDER_MESSAGES`, `M_TYPE_UINT64`, `magma_folder_children()`, and `magma_folder_delete()`.

magma/objects/messages/messages.c File Reference

Mail message interface functions.

```
#include "magma.h"
```

Functions

- void **message_free** (**message_t** *message)
- *Free a message object and its underlying data.* **message_t** * **message_alloc** (uint64_t messagenum, uint64_t created, uint64_t signature, uint64_t key, uint64_t flags, **stringer_t** *server, size_t size)
- *Allocate a new message object.* **inx_t** * **messages_update** (uint64_t usernum)

Fetch all of a user's message folders and their child messages from the database.

Detailed Description

Mail message interface functions.

\$Author:\$ \$Author\$ \$Revision\$

Definition in file **messages.c**.

Function Documentation

message_t* **message_alloc** (uint64_t *messagenum*, uint64_t *created*, uint64_t *signature*,
uint64_t *key*, uint64_t *flags*, stringer_t * *server*, size_t *size*)

Allocate a new message object.

messages.c

Parameters:

messagenum the numerical message id of the new message.

created the message creation timestamp.

signature the message's spam filter signature number.

key the message's spam filter access key.

flags the message's flags value.

server a managed string containing the name of the server where the message will be stored.

size the size, in bytes, of the raw message data.

Returns:

NULL on failure, or the newly allocated message object on success.

Definition at line 65 of file messages.c.

References [align\(\)](#), [FOREIGNDATA](#), [JOINTED](#), [log_pedantic](#), [message_t](#), [mm_alloc\(\)](#), [mm_copy\(\)](#), [mm_free\(\)](#), [PLACER_T](#), [placer_t](#), [rwlock_init\(\)](#), [st_data_get\(\)](#), [st_length_get\(\)](#), and [STACK](#).

Referenced by [messages_fetch\(\)](#).

void message_free (**message_t** * *message*)

Free a message object and its underlying data.

Returns:

This function returns no value.

Definition at line 44 of file messages.c.

References mm_cleanup(), rwlock_destroy(), and st_cleanup().

Referenced by message_folder_alloc(), and messages_fetch().

inx_t* messages_update (uint64_t *usernum*)

Fetch all of a user's message folders and their child messages from the database.

Parameters:

usernum the numerical id of the target user.

Returns:

NULL on failure or an inx object holding all the retrieved and populated message folder objects on success.

Definition at line 101 of file messages.c.

References inx_cleanup(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_free(), log_pedantic, M_FOLDER_MESSAGES, magma_folder_fetch(), and messages_fetch().

Referenced by meta_message_folders_update().

magma/objects/users/messages.c File Reference

The user context interface for message folders.

```
#include "magma.h"
```

Functions

- **int_t meta_message_folders_update** (meta_user_t *user, META_LOCK_STATUS locked)

Refresh a user's message folders if stale, or retrieve them from the database if they are not in memory.

Detailed Description

The user context interface for message folders.

Definition in file **messages.c**.

Function Documentation

int_t meta_message_folders_update (meta_user_t * user, META_LOCK_STATUS *locked*)

Refresh a user's message folders if stale, or retrieve them from the database if they are not in memory.

messages.c

TODO: The serial number scheme needs to refreshing. There should be a serial number to indicate changes in the folder structure, and then individual serial numbers for each folder to indicate when the contents are changed. If that were the case this function really only needs to update the list of folders.

See also:

messages_update()

Parameters:

user a pointer to the meta user object requesting the folders.

locked if META_LOCKED, lock the meta user object for the duration of the request.

Returns:

-1 on failure, or 1 on success.

Definition at line 27 of file messages.c.

References meta_user_t::folders, inx_free(), meta_user_t::message_folders, messages_update(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), OBJECT_FOLDERS, serial_get(), serial_increment(), meta_user_t::serials, and meta_user_t::usernum.

Referenced by meta_get().

magma/servers/imap/messages.c File Reference

Functions used to handle IMAP commands and actions.

```
#include "magma.h"
```

Functions

- **int_t imap_append_message** (**connection_t** *con, **meta_folder_t** *folder, uint32_t flags, **stringer_t** *message, uint64_t *outnum)
- *messages.c* **int_t imap_message_expunge** (**connection_t** *con, **meta_message_t** *message)
- **int_t imap_message_copier** (**connection_t** *con, **meta_message_t** *message, uint64_t target, uint64_t *outnum)

Detailed Description

Functions used to handle IMAP commands and actions.

Definition in file **messages.c**.

Function Documentation

int_t imap_append_message (**connection_t** * con, **meta_folder_t** * folder, **uint32_t** flags, **stringer_t** * message, **uint64_t** * outnum)

messages.c

Definition at line 15 of file messages.c.

References magma_t::active, meta_folder_t::foldernum, connection_t::imap, inx_insert(), log_pedantic, M_TYPE_UINT64, magma, MAIL_STATUS_APPENDED, MAIL_STATUS_RECENT, mail_store_message(), meta_messages_update_sequences(), META_USER_ENCRYPT_DATA, mm_alloc(), mm_free(), OBJECT_MESSAGES, serial_get(), serial_increment(), st_char_get(), st_length_get(), st_length_int(), magma_t::storage, multi_t::u64, and multi_t::val.

Referenced by imap_append().

int_t imap_message_copier (**connection_t** * con, **meta_message_t** * message, **uint64_t** target, **uint64_t** * outnum)

Definition at line 70 of file messages.c.

References meta_message_t::created, connection_t::imap, inx_insert(), log_pedantic, M_TYPE_UINT64, mail_copy_message(), MAIL_STATUS_DELETED, MAIL_STATUS_HIDDEN, MAIL_STATUS_RECENT, meta_message_t::messagenum, meta_message_dupe(), meta_messages_update_sequences(), mm_free(), OBJECT_MESSAGES, serial_get(), serial_increment(), meta_message_t::server, meta_message_t::sigkey, meta_message_t::signum, meta_message_t::size, meta_message_t::status, status, multi_t::type, multi_t::u64, and multi_t::val.

Referenced by imap_copy().

int_t imap_message_expunge (connection_t * *con*, meta_message_t * *message*)

Definition at line 58 of file messages.c.

References connection_t::imap, inx_delete(), M_TYPE_UINT64, mail_remove_message(), meta_message_t::messagenum, meta_message_t::server, and meta_message_t::size.

Referenced by imap_close(), and imap_expunge().

magma/servers/smtp/messages.c File Reference

Handle the SMTP message structure.

```
#include "magma.h"
```

Detailed Description

Handle the SMTP message structure.

Definition in file **messages.c**.

magma/web/portal/messages.c File Reference

Functions to help assemble mail message for output.

```
#include "magma.h"
```

Functions

- `json_t * portal_message_meta (meta_message_t *meta)`
- `json_t * portal_message_source (meta_message_t *meta)`
- `json_t * portal_message_security (meta_message_t *meta)`
- `json_t * portal_message_server (meta_message_t *meta)`
- `json_t * portal_message_header (meta_message_t *meta, mail_message_t *data)`
- *Build the "header" section response to a rpc-json "messages.load" request.* `json_t * portal_message_body (meta_message_t *meta, mail_message_t *data)`
- `json_t * portal_message_attachments (meta_message_t *meta, mail_message_t *data)`
- *messages.c* `json_t * portal_message_info (meta_message_t *meta)`

Build the "info" section response to a rpc-json "messages.load" request.

Detailed Description

Functions to help assemble mail message for output.

Definition in file **messages.c**.

Function Documentation

`json_t* portal_message_attachments (meta_message_t * meta, mail_message_t * data)`

messages.c

LOW: We also need to detect messages that have no readable content so the entire body is just a blob.

Create a function to search for the filename. Common locations are: Content-Type: image/png;
name="webmail-php-download.png" Content-Disposition: attachment;
filename="webmail-php-download.png"

Definition at line 169 of file messages.c.

References `ar_field_ptr()`, `ar_length_get()`, `mail_mime_t::body`, `mail_mime_t::children`, `count`, `mail_mime_t::header`, `json_array_append_new_d`, `json_array_d`, `json_decref_d`, `json_pack_ex_d`, `log_pedantic`, `mail_mime_type_group()`, `mail_mime_type_sub()`, `MESSAGE_TYPE_HTML`, `MESSAGE_TYPE_MULTI_MIXED`, `MESSAGE_TYPE_PLAIN`, `mail_message_t::mime`, `PLACER`, `st_aprint()`, `st_char_get()`, `st_cleanup()`, `st_length_get()`, `st_merge`, `mail_mime_t::type`, and `type()`.

Referenced by `portal_endpoint_messages_load()`.

`json_t* portal_message_body (meta_message_t * meta, mail_message_t * data)`

TODO: I consider it a miracle whenever the above logic actually manages to select the message content. But if it doesn't we fall through to this fixed error message, at least until we can improve the selection process.

HIGH: Because JSON wants NULLERS, and were using PLACEHOLDERS, its printing out the entire message, not just the section of interest.

Definition at line 113 of file messages.c.

References `ar_field_ptr()`, `mail_mime_t::body`, `mail_mime_t::children`, `CONTIGUOUS`, `HEAP`, `json_pack_ex_d`, `log_pedantic`, `MAIL_MIME_RECURSION_LIMIT`, `MESSAGE_TYPE_HTML`, `MESSAGE_TYPE_MULTI_ALTERNATIVE`, `MESSAGE_TYPE_MULTI_MIXED`, `MESSAGE_TYPE_MULTI_RELATED`, `MESSAGE_TYPE_MULTI_UNKOWN`, `MESSAGE_TYPE_PLAIN`, `mail_message_t::mime`, `NULLER_T`, `pl_char_get()`, `pl_length_get()`, `PLACER`, `st_char_get()`, `st_cleanup()`, `st_dupe_opts()`, `st_nullify()`, `mail_mime_t::type`, and `type()`.

Referenced by `portal_endpoint_messages_load()`.

`json_t* portal_message_header (meta_message_t * meta, mail_message_t * data)`

Build the "header" section response to a rpc-json "messages.load" request.

Note:

The following header fields will be returned: To, CC, BCC, From, Replyto, Sender, Return-Path, Subject, Date, and Size.

Parameters:

meta a pointer to the meta message object of the requested message.

data a pointer to the mail message object containing the requested message's data.

Returns:

a pointer to a json object containing the header fields of the requested message.

Definition at line 84 of file messages.c.

References `mail_message_t::header_length`, `json_pack_ex_d`, `log_pedantic`, `mail_header_fetch_cleaned()`, `mm_wipe()`, `PLACER`, `meta_message_t::size`, `st_char_get()`, `st_cleanup()`, and `mail_message_t::text`.

Referenced by `portal_endpoint_messages_load()`.

`json_t* portal_message_info (meta_message_t * meta)`

Build the "info" section response to a rpc-json "messages.load" request.

Parameters:

meta a pointer to the meta message object of the requested message.

Returns:

a pointer to a json object containing the appropriate information about the requested message.

Definition at line 275 of file messages.c.

References `json_pack_ex_d`, and `log_pedantic`.

Referenced by `portal_endpoint_messages_load()`.

json_t* portal_message_meta (meta_message_t * meta)

Definition at line 18 of file messages.c.

References meta_message_t::foldernum, json_pack_ex_d, log_pedantic, meta_message_t::messagenum, portal_message_flags_array(), and portal_message_tags_array().

Referenced by portal_endpoint_messages_load().

json_t* portal_message_security (meta_message_t * meta)

TODO: Replace hard coded values with actual data.

Definition at line 47 of file messages.c.

References json_pack_ex_d, and log_pedantic.

Referenced by portal_endpoint_messages_load().

json_t* portal_message_server (meta_message_t * meta)

TODO: Replace hard coded values with actual data.

Definition at line 61 of file messages.c.

References meta_message_t::created, json_pack_ex_d, and log_pedantic.

Referenced by portal_endpoint_messages_load().

json_t* portal_message_source (meta_message_t * meta)

TODO: Replace hard coded values with actual data.

Definition at line 32 of file messages.c.

References json_pack_ex_d, log_pedantic, and rand_get_int64().

Referenced by portal_endpoint_messages_load().

magma/objects/locks.c File Reference

Functions for managing locks synchronized via memcached.

```
#include "magma.h"
```

Defines

- `#define MAGMA_LOCK_TIMEOUT 60`
- `#define MAGMA_LOCK_EXPIRATION 600`

Functions

- `int_t lock_get (stringer_t *key)`
- *Acquire a named lock, with synchronization provided via memcached.* `void lock_release (stringer_t *key)`
- *Release a named lock, with synchronization provided via memcached.* `int_t user_lock (uint64_t usernum)`
- *Acquire a lock in the magma.user keyspace.* `void user_unlock (uint64_t usernum)`

Unlock a lock in the magma.user keyspace.

Detailed Description

Functions for managing locks synchronized via memcached.

Definition in file **locks.c**.

Define Documentation

#define MAGMA_LOCK_EXPIRATION 600

Definition at line 16 of file locks.c.

Referenced by lock_get().

#define MAGMA_LOCK_TIMEOUT 60

Definition at line 15 of file locks.c.

Referenced by lock_get().

Function Documentation

int_t lock_get (stringer_t * key)

Acquire a named lock, with synchronization provided via memcached.

locks.c

See also:

`cache_silent_add()`

Note:

The lock will be held for 10 seconds, and locking attempts will occur periodically for 60 seconds prior to failure.

Parameters:

key a managed string containing the name of the lock to be acquired.

Returns:

-1 on general failure, 0 on memcached failure, or 1 on success.

Definition at line 25 of file locks.c.

References `cache_silent_add()`, `lock`, `log_pedantic`, `MAGMA_LOCK_EXPIRATION`, `MAGMA_LOCK_TIMEOUT`, `MANAGEDBUF`, `PLACER`, `st_char_get()`, `st_empty()`, `st_length_int()`, and `st_sprint()`.

Referenced by `smtp_reply()`, and `user_lock()`.

void lock_release (stringer_t * key)

Release a named lock, with synchronization provided via memcached.

See also:

`cache_delete()`

Note:

The lock will be held for 10 seconds, and locking attempts will occur periodically for 60 seconds prior to failure.

Parameters:

key a managed string containing the name of the lock to be released.

Returns:

-1 on general failure, 0 on memcached failure, or 1 on success.

LOW: At some point we should add logic to check whether this cluster node even owns the lock before taking the drastic step of deleting it.

Definition at line 64 of file locks.c.

References `cache_delete()`, `lock`, `log_pedantic`, `MANAGEDBUF`, `st_char_get()`, `st_empty()`, `st_length_int()`, and `st_sprint()`.

Referenced by `smtp_reply()`, and `user_unlock()`.

int_t user_lock (uint64_t usernum)

Acquire a lock in the magma.user keyspace.

Parameters:

usernum the numerical id of the user for whom the lock will be acquired.

Returns:

-1 on general failure, 0 on memcached failure, or 1 on success.

Definition at line 84 of file locks.c.

References `lock_get()`, `MANAGEDBUF`, and `st_sprint()`.

Referenced by `imap_close()`, `imap_copy()`, `imap_expunge()`, `smtp_rollout()`, and `smtp_store_message()`.

`void user_unlock (uint64_t usernum)`

Unlock a lock in the `magma.user` keyspace.

Parameters:

usernum the numerical id of the user for whom the lock will be unlocked.

Returns:

This function returns no value.

Definition at line 100 of file `locks.c`.

References `lock_release()`, `MANAGEDDBUF`, and `st_sprint()`.

Referenced by `imap_close()`, `imap_copy()`, `imap_expunge()`, `smtp_rollout()`, and `smtp_store_message()`.

magma/objects/mail/cleanup.c File Reference

Functions for cleaning up mail messages.

```
#include "magma.h"
```

Functions

- void **mail_destroy_header** (**stringer_t** *header)
 - Destroy a mail header. **bool_t** **mail_message_cleanup** (**stringer_t** **message)
Clean up the body of a message read in via smtp, enforcing magma.smtp.wrap_line_length.
-

Detailed Description

Functions for cleaning up mail messages.

Definition in file **cleanup.c**.

Function Documentation

void mail_destroy_header (stringer_t * header)

Destroy a mail header.

cleanup.c

Parameters:

header a managed string containing the mail header to be freed.

Returns:

This function returns no value.

Definition at line 20 of file cleanup.c.

References `st_cleanup()`.

Referenced by `imap_fetch_body()`, `imap_fetch_message()`, `imap_fetch_return_message()`, `imap_fetch_return_mime()`, `imap_fetch_return_text()`, and `imap_search_messages()`.

bool_t mail_message_cleanup (stringer_t ** message)

Clean up the body of a message read in via smtp, enforcing magma.smtp.wrap_line_length.

Note:

This function fixes broken line separators by making sure each `\` is followed by a space and vice versa. All Return-Path: header lines are also removed. New lines are begun whenever the current length of any line reaches the configuration value set in magma.smtp.wrap_line_length. The trailing dot at the end of the smtp DATA command is also stripped.

If the original message ends with `.`, it will have `\.` appended to it.

Parameters:

message a pointer to a managed string that contains the message input, and will also store the cleaned output on success.

Returns:

true on success or false on failure.

Definition at line 36 of file cleanup.c.

References HEAP, JOINTED, length, log_pedantic, magma, MAPPED_T, mm_cmp_ci_eq(), magma_t::smtp, st_alloc_opts(), st_char_get(), st_free(), st_length_get(), st_length_set(), and magma_t::wrap_line_length.

Referenced by smtp_data().

magma/objects/mail/counters.c File Reference

Functions used to calculate mail message metrics needed to implement business rules.

```
#include "magma.h"
```

Functions

- `uint32_t mail_count_received (stringer_t *message)`
- *Count the number of "Received:" lines in a mail message.* `size_t mail_header_end (stringer_t *message)`

Get the number of bytes taken up by a mail header.

Detailed Description

Functions used to calculate mail message metrics needed to implement business rules.

Definition in file `counters.c`.

Function Documentation

`uint32_t mail_count_received (stringer_t * message)`

Count the number of "Received:" lines in a mail message.

`counters.c`

Parameters:

message a managed string containing the mail message to be scanned.

Returns:

the number of matching lines found, or 0 on failure.

Definition at line 20 of file `counters.c`.

References `log_pedantic`, `PLACER`, `st_cmp_ci_starts()`, and `st_empty_out()`.

Referenced by `smtp_data()`.

`size_t mail_header_end (stringer_t * message)`

Get the number of bytes taken up by a mail header.

Parameters:

message a managed string containing the full smtp mail message.

Returns:

0 on failure, or the number of bytes occupied by the mail header.

Definition at line 61 of file `counters.c`.

References `length`, `log_pedantic`, `st_char_get()`, and `st_length_get()`.

Referenced by `mail_add_forward_headers()`, `mail_add_outbound_headers()`, `mail_add_required_headers()`, `mail_create_message()`, `mail_headers()`, `mail_load_header()`, `mail_message()`, `mail_mod_subject()`, `mail_setup_basic()`, and `smtp_check_filters()`.

magma/providers/consumers/counters.c File Reference

#include "magma.h"

Functions

- int **stamp_counter_check** (char *key, size_t keylen, unsigned long interval)
 - void **stamp_counter_increment** (char *key, size_t keylen)
-

Function Documentation

int stamp_counter_check (char * *key*, size_t *keylen*, unsigned long *interval*)

Definition at line 15 of file counters.c.

References `cache_get()`, `count`, `length`, `log_pedantic`, `PLACER`, `placer_t`, `st_free()`, `tok_get_count_st()`, `tok_get_st()`, and `uint64_conv_pl()`.

void stamp_counter_increment (char * *key*, size_t *keylen*)

Definition at line 46 of file counters.c.

References `cache_append()`, `log_pedantic`, and `PLACER`.

magma/objects/mail/headers.c File Reference

Functions used to handle mail message header information.

```
#include "magma.h"
```

Functions

- **stringer_t * mail_header_fetch_cleaned** (stringer_t *header, stringer_t *key)
 - *Fetch the value of a specified line from a mail header, performing whitespace and line cleaning.* **stringer_t * mail_header_fetch_all** (stringer_t *header, stringer_t *key)
 - *Get all the values that match a named header line in a message header.* **placer_t mail_header_pop** (stringer_t *header, size_t *position)
 - *Pop the next header line from a mail message header.* **placer_t mail_store_header** (chr_t *stream, size_t length)
 - *Return a placer pointing to the trimmed value of a mail header.* **bool_t mail_headers** (smtp_message_t *message)
 - *Parse an smtp message header and store the To, From, Date, and Subject header values.* void **mail_mod_subject** (stringer_t **message, chr_t *label)
 - *Prepend text to a Subject header line inside of a mail message.* **bool_t mail_add_required_headers** (connection_t *con, smtp_message_t *message)
 - *Generate any missing required headers for an smtp message object.* **stringer_t * mail_add_inbound_headers** (connection_t *con, smtp_inbound_prefs_t *prefs)
 - *Prepend the Return-Path: and Received: headers to an inbound smtp message.* void **mail_add_forward_headers** (server_t *server, stringer_t **message, stringer_t *id, int_t mark, uint64_t signum, uint64_t sigkey)
 - *Add necessary headers to a forwarded mail message.* **int_t mail_add_outbound_headers** (connection_t *con)
-
- Add a Received: header and dkim signature to an outbound relayed smtp message.*

Detailed Description

Functions used to handle mail message header information.

Definition in file **headers.c**.

Function Documentation

void mail_add_forward_headers (server_t * server, stringer_t ** message, stringer_t * id, int_t mark, uint64_t signum, uint64_t sigkey)

Add necessary headers to a forwarded mail message.

headers.c

Note:

This function will prepend tags to the Subject line like "JUNK", "INFECTED", "SPOOFED", etc. The following headers will be stripped from the message: Sender, Return-Path, DKIM-Signature, and DomainKey-Signature. This daemon will be reinserted into the headers as the origin of the Sender: header value. If signum and sigkey are both set, then a spam training url will also be inserted into the message. Finally, if **magma.dkim.enabled** is set, a dkim signature will be added to the message.

Parameters:

server the server object of the web server where the teacher application is hosted.
message the address of a managed string containing the message data that will be set to the modified message data on success.
id a managed string containing the message id (for dkim).
mark a bitmask of marks to be tagged in the message subject (SMTP_MARK_SPAM, SMTP_MARK_VIRUS, SMTP_MARK_SPOOF, SMTP_MARK_RBL, and SMTP_MARK_PHISH).
signum the optional spam signature number referenced by the teacher url.
sigkey the optional spam signature key for client verification in the teacher app.

Returns:

This function returns no value.

Definition at line 732 of file headers.c.

References CONSTANT, magma_t::dkim, dkim_create(), magma_t::enabled, HEAP, JOINTED, log_pedantic, magma, mail_destroy(), mail_header_end(), mail_header_pop(), MAIL_MARK_JUNK, mail_message(), mail_mod_subject(), mail_signature_add(), MAPPED_T, mm_cmp_ci_eq(), pl_empty(), PLACER, placer_t, SMTP_MARK_PHISH, SMTP_MARK_RBL, SMTP_MARK_SPAM, SMTP_MARK_SPOOF, SMTP_MARK_VIRUS, st_alloc_opts(), st_append_opts(), st_char_get(), st_cleanup(), st_cmp_ci_starts(), st_data_get(), st_dupe(), st_dupe_opts(), st_free(), st_length_get(), st_merge_opts(), status, and mail_message_t::text.

Referenced by smtp_forward_message().

stringer_t* mail_add_inbound_headers (connection_t * con, smtp_inbound_prefs_t * prefs)

Prepend the Return-Path: and Received: headers to an inbound smtp message.

Parameters:

con the connection across which the inbound smtp message was accepted.
prefs the smtp inbound preferences corresponding to the connection, with the rcptto field populated.

Returns:

NULL on failure, or a managed string containing the message data preceded by the Return-Path and Received headers on success.

Definition at line 640 of file headers.c.

References con_addr_presentation(), con_reverse_check(), CONSTANT, server_t::domain, smtp_session_t::esmtpl, HEAP, smtp_session_t::helo, smtp_message_t::id, JOINTED, log_pedantic, smtp_session_t::mailfrom, MANAGEDBUF, MAPPED_T, smtp_session_t::message, smtp_inbound_prefs_t::rcptto, connection_t::server, connection_t::smtp, st_cmp_ci_eq(), st_cmp_cs_eq(), st_merge_opts(), and smtp_message_t::text.

Referenced by smtp_accept_message().

int_t mail_add_outbound_headers (connection_t * con)

Add a Received: header and dkim signature to an outbound relayed smtp message.

Note:

If the message already has a Received header, the dkim signature will be inserted right before the first instance.

Parameters:

con the connection across which the outbound smtp message is being sent.

Returns:

NULL on failure, or a managed string containing the message data preceded by the Received header and including the dkim signature on success.

Definition at line 849 of file headers.c.

References smtp_recipients_t::address, con_addr_presentation(), con_reverse_check(), magma_t::dkim, dkim_create(), server_t::domain, magma_t::enabled, smtp_session_t::esmtplib, smtp_session_t::helo, smtp_message_t::id, JOINTED, log_pedantic, magma, mail_header_end(), mail_header_pop(), MANAGEDBUF, MAPPED_T, smtp_session_t::message, mm_cmp_ci_eq(), smtp_session_t::num_recipients, smtp_session_t::out_prefs, pl_empty(), PLACER, placer_t, smtp_outbound_prefs_t::recipients, connection_t::server, connection_t::smtp, st_char_get(), st_cleanup(), st_cmp_cs_eq(), st_data_get(), st_dup(), st_empty(), st_free(), st_import(), st_length_get(), st_merge_opts(), and smtp_message_t::text.

Referenced by smtp_relay_message().

bool_t mail_add_required_headers (connection_t * con, smtp_message_t * message)

Generate any missing required headers for an smtp message object.

Note:

These required headers include To, From, Subject and Date, and if they are absent, they will be generated from the specified connection's inbound or outbound preferences, accordingly.

Parameters:

con a pointer to the connection object across which the smtp message was received.

message the smtp message object to be updated for any missing required headers.

Returns:

true on success or false on failure.

Definition at line 459 of file headers.c.

References smtp_recipients_t::address, smtp_session_t::authenticated, smtp_message_t::date, smtp_message_t::from, smtp_message_t::header_length, HEAP, smtp_session_t::in_prefs, JOINTED, length, log_pedantic, mail_header_end(), mail_headers(), smtp_session_t::mailfrom, MAPPED_T, smtp_recipients_t::next, smtp_inbound_prefs_t::next, smtp_session_t::out_prefs, PLACER, smtp_inbound_prefs_t::rcptto, smtp_outbound_prefs_t::recipients, connection_t::smtp, st_alloc, st_avail_get(), st_char_get(), st_cleanup(), st_empty(), st_free(), st_import(), st_length_get(), st_length_set(), st_merge, st_merge_opts(), smtp_message_t::subject, smtp_message_t::text, and smtp_message_t::to.

Referenced by smtp_data().

stringer_t* mail_header_fetch_all (stringer_t * header, stringer_t * key)

Get all the values that match a named header line in a message header.

Parameters:

header a managed string containing the message header.

key a managed string containing the header name to be searched for.

Returns:

NULL on failure or a managed string containing all the matching header line values separated by newlines.

Definition at line 154 of file headers.c.

References mm_cmp_ci_eq(), PLACER, st_char_get(), st_empty(), st_free(), st_import(), st_length_get(), and st_merge.

Referenced by `imap_fetch_body_mime()`, `imap_search_messages_header()`, `mail_discover_encoding()`, `mail_discover_type()`, `mail_get_boundary()`, `mail_mime_boundary()`, and `smtp_check_filters()`.

`stringer_t* mail_header_fetch_cleaned (stringer_t * header, stringer_t * key)`

Fetch the value of a specified line from a mail header, performing whitespace and line cleaning.

Note:

This function handles mail header values that span new lines, although the output excludes line breaks and consecutive space characters.

Parameters:

header a managed string containing the full header of the target mail message.

key a managed string containing the name of the desired header name to be extracted, excluding the trailing ":".

Returns:

NULL on failure, or a managed string containing the cleaned value of the specified mail header line on success.

Definition at line 22 of file `headers.c`.

References `PLACER`, `st_alloc`, `st_char_get()`, `st_cmp_ci_starts()`, `st_empty()`, `st_length_get()`, and `st_length_set()`.

Referenced by `imap_fetch_bodystructure()`, `imap_fetch_envelope()`, `imap_search_messages_date()`, `mail_mime_content_encoding()`, `mail_mime_content_id()`, `mail_mime_encoding()`, `mail_mime_type()`, `mail_mime_type_group()`, `mail_mime_type_parameters()`, `mail_mime_type_sub()`, `portal_endpoint_messages_list()`, and `portal_message_header()`.

`placer_t mail_header_pop (stringer_t * header, size_t * position)`

Pop the next header line from a mail message header.

Parameters:

header a managed string containing the entire mail message header.

position a pointer to a `size_t` that contains the tracker index into the entire header for the pop operation, and will also receive the new value of the tracker for the subsequent call, when a new header line is located.

Returns:

`pl_null()` on error or a `placer` pointing to the next mail message subject line on success.

Definition at line 243 of file `headers.c`.

References `pl_init()`, `pl_null()`, `st_char_get()`, `st_empty()`, and `st_length_get()`.

Referenced by `imap_fetch_body_header()`, `mail_add_forward_headers()`, `mail_add_outbound_headers()`, and `mail_mod_subject()`.

`bool_t mail_headers (smtp_message_t * message)`

Parse an smtp message header and store the To, From, Date, and Subject header values.

Note:

The found header fields will be stored inside the same smtp message object passed by the caller as input.

Parameters:

message a pointer to a partially populated smtp message object that contains the raw message data to be parsed.

Returns:

true on success or false on failure.

Definition at line 325 of file headers.c.

References smtp_message_t::date, smtp_message_t::from, smtp_message_t::header_length, length, log_pedantic, mail_header_end(), mail_store_header(), mm_cmp_ci_eq(), st_char_get(), st_length_get(), smtp_message_t::subject, smtp_message_t::text, and smtp_message_t::to.

Referenced by mail_add_required_headers(), and mail_create_message().

void mail_mod_subject (stringer_t ** *message*, chr_t * *label*)

Prepend text to a Subject header line inside of a mail message.

Note:

If no Subject line is found in the header of the message, one will be created and inserted with the specified label at the end of the header.

Parameters:

message a pointer to the address of a managed string that contains the mail message header or mail message body, and will receive the value of the resulting managed string that will be allocated to hold the modified contents.

label a null-terminated string that will be prepended to the value of the subject header value.

Returns:

This function returns no value.

Definition at line 391 of file headers.c.

References CONSTANT, log_pedantic, mail_header_end(), mail_header_pop(), pl_empty(), PLACER, placer_t, st_char_get(), st_cmp_ci_starts(), st_data_get(), st_free(), st_length_get(), and st_merge.

Referenced by mail_add_forward_headers(), mail_load_header(), mail_load_message(), and smtp_check_filters().

placer_t mail_store_header (chr_t * *stream*, size_t *length*)

Return a placer pointing to the trimmed value of a mail header.

Note:

This function merely marks a string residing between leading whitespace and a trailing or

.

Parameters:

stream a pointer to a buffer containing the start of the mail header data.

length the length, in bytes, of the mail header field. a placer containing the trimmed value of the mail header line.

Definition at line 298 of file headers.c.

References pl_init().

Referenced by mail_headers(), mail_message(), and mail_setup_basic().

magma/objects/mail/load_message.c File Reference

Functions used to load mail messages.

```
#include "magma.h"
```

Functions

- **stringer_t * mail_load_header** (**meta_message_t** *meta, **meta_user_t** *user)
- *Get the header of a message, checking first in the cache and then on disk.* **mail_message_t * mail_load_message** (**meta_message_t** *meta, **meta_user_t** *user, **server_t** *server, **bool_t** parse)
- *Load a stored mail message from disk.* **mail_message_t * mail_load_message_top** (**meta_message_t** *meta, **meta_user_t** *user, **server_t** *server, **uint64_t** lines, **bool_t** parse)

Get the top of a mail message, up to a specified maximum number of lines of content.

Detailed Description

Functions used to load mail messages.

Definition in file **load_message.c**.

Function Documentation

stringer_t* mail_load_header (**meta_message_t** * *meta*, **meta_user_t** * *user*)

Get the header of a message, checking first in the cache and then on disk.

Note:

The file data is first unencrypted and decompressed, according to the file header flags. When extracted, the header's Subject line is branded with any applicable labels such as JUNK, INFECTED, SPOOFED, BLACKHOLED, PHISHING.

Parameters:

meta the meta message object of the message to be queried.
user the meta user object of the user that owns the message.

Returns:

NULL on failure or a managed string containing the message's header on success.

Definition at line 23 of file load_message.c.

References `cache_add()`, `cache_get()`, `compress_block_length()`, `decompress_block_lzo()`, `decompress_lzo()`, `ecies_decrypt()`, `ECIES_PRIVATE_BINARY`, `FMESSAGE_MAGIC_1`, `FMESSAGE_MAGIC_2`, `FMESSAGE_OPT_ENCRYPTED`, `mail_message_t::header_length`, `log_pedantic`, `mail_db_hide_message()`, `mail_destroy()`, `mail_header_end()`, `mail_load_message()`, `MAIL_MARK_BLACKHOLED`, `MAIL_MARK_INFECTED`, `MAIL_MARK_JUNK`, `MAIL_MARK_PHISHING`, `MAIL_MARK_SPOOFED`, `mail_message_path()`, `mail_mod_subject()`, `MAIL_STATUS_ENCRYPTED`, `message_fheader_t`, `meta_message_t::messagenum`, `mm_alloc()`, `ns_free()`, `OBJECT_MESSAGES`, `PLACER`, `serial_increment()`, `meta_message_t::server`, `st_char_get()`, `st_free()`, `st_import()`, `st_length_get()`, `st_length_set()`, `meta_message_t::status`, `meta_user_t::storage_privkey`, `mail_message_t::text`, and `meta_user_t::usernum`.

Referenced by `imap_fetch_return_header()`, `imap_search_messages_date()`, `imap_search_messages_header()`, and `portal_endpoint_messages_list()`.

mail_message_t* mail_load_message (meta_message_t * *meta*, meta_user_t * *user*, server_t * *server*, bool_t *parse*)

Load a stored mail message from disk.

load_message.c

Note:

The mail message will always, at the very least, be compressed using the lzo algorithm; however, on-disk encryption may be enabled. If parsing is enabled, a spam signature training link may be embedded in the message.

Parameters:

meta the meta message object of the message to be loaded from disk.

user the meta user object of the user that owns the requested message.

server the server object of the web server where the spam teacher application is hosted.

parse if true, the header's Subject line is branded with any applicable labels such as JUNK, INFECTED, SPOOFED, BLACKHOLED, PHISHING.

Returns:

NULL on failure or a mail message object containing the retrieved mail message data on success.

Definition at line 238 of file load_message.c.

References `cache_add()`, `cache_get()`, `compress_import()`, `decompress_lzo()`, `ecies_decrypt()`, `ECIES_PRIVATE_BINARY`, `meta_user_t::flags`, `FMESSAGE_MAGIC_1`, `FMESSAGE_MAGIC_2`, `FMESSAGE_OPT_ENCRYPTED`, `mail_message_t::header_length`, `log_info`, `log_pedantic`, `mail_cache_get()`, `mail_cache_set()`, `mail_db_hide_message()`, `MAIL_MARK_BLACKHOLED`, `MAIL_MARK_INFECTED`, `MAIL_MARK_JUNK`, `MAIL_MARK_PHISHING`, `MAIL_MARK_SPOOFED`, `mail_message()`, `mail_message_path()`, `mail_mod_subject()`, `mail_signature_add()`, `MAIL_STATUS_ENCRYPTED`, `message_fheader_t`, `meta_message_t::messagenum`, `META_USER_ENCRYPT_DATA`, `mm_free()`, `ns_free()`, `OBJECT_MESSAGES`, `PLACER`, `serial_increment()`, `meta_message_t::server`, `meta_message_t::sigkey`, `meta_message_t::signum`, `st_alloc`, `st_char_get()`, `st_data_get()`, `st_free()`, `st_import()`, `st_length_set()`, `meta_message_t::status`, `meta_user_t::storage_privkey`, `mail_message_t::text`, and `meta_user_t::usernum`.

Referenced by `imap_fetch_message()`, `imap_fetch_return_message()`, `imap_fetch_return_mime()`, `imap_fetch_return_text()`, `imap_search_messages_body()`, `imap_search_messages_text()`, `mail_load_header()`, `mail_load_message_top()`, `pop_retr()`, and `portal_endpoint_messages_load()`.

mail_message_t* mail_load_message_top (meta_message_t * *meta*, meta_user_t * *user*, server_t * *server*, uint64_t *lines*, bool_t *parse*)

Get the top of a mail message, up to a specified maximum number of lines of content.

See also:

`mail_load_message()`

Parameters:

meta the meta message object of the message to be loaded from disk.

user the meta user object of the user that owns the requested message.

server the server object of the web server where the spam teacher application is hosted.

lines the maximum number of lines to be retrieved from the loaded message.

parse sets the parse parameter passed to `mail_load_message()`.

Returns:

NULL on failure or a mail message object containing the retrieved mail message data (truncated if necessary) on success.

Definition at line 479 of file load_message.c.

References `length`, `mail_load_message()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, and `mail_message_t::text`.

Referenced by `pop_top()`.

magma/objects/mail/mail.h File Reference

Functions used to interface with and manage message data.

Data Structures

- struct **mail_cache_t**
- struct **basic_message_t**
- struct **mail_mime_t**
- struct **mail_message_t**
- struct **media_type_t**

Defines

- #define **MAIL_MIME_RECURSION_LIMIT** 16
- #define **MAIL_SIGNATURES_RECURSION_LIMIT** 16

Functions

- void **mail_cache_destroy** (void *holder)
- *cache.c* **stringer_t * mail_cache_get** (uint64_t messagenum)
- *Attempt to retrieve the contents of a message from the thread's cached data.* void **mail_cache_reset** (void)
- *Reset the thread's mail cache and free any held message.* void **mail_cache_set** (uint64_t messagenum, **stringer_t** *text)
- *Set the contents of the thread's mail cache.* **bool_t mail_cache_start** (void)
- *Initialize the thread-specific data key used for the mail message cache.* void **mail_cache_stop** (void)
- *Destroy the thread-specific data key used for the mail message cache.* void **mail_cache_thread_stop** (void)
- *Free the thread-specific data for the mail message cache.* void **mail_destroy_header** (**stringer_t** *header)
- *cleanup.c* **bool_t mail_message_cleanup** (**stringer_t** **message)
- *Clean up the body of a message read in via smtp, enforcing magma.smtp.wrap_line_length.* uint32_t **mail_count_received** (**stringer_t** *message)
- *counters.c* size_t **mail_header_end** (**stringer_t** *message)
- *Get the number of bytes taken up by a mail header.* **bool_t mail_db_delete_message** (uint64_t usernum, uint64_t messagenum, uint32_t size, **int_t** transaction)
- *datatier.c* void **mail_db_hide_message** (uint64_t messagenum)
- *Set a message invisible in the database.* uint64_t **mail_db_insert_duplicate_message** (uint64_t usernum, uint64_t foldernum, uint32_t status, uint32_t size, uint64_t signum, uint64_t sigkey, uint64_t created, **int_t** transaction)
- *Insert a duplicate entry for a message in the database.* uint64_t **mail_db_insert_message** (uint64_t usernum, uint64_t foldernum, uint32_t status, uint32_t size, uint64_t signum, uint64_t sigkey, **int_t** transaction)
- *Insert a mail message into the database.* **int_t mail_db_update_message_folder** (uint64_t usernum, uint64_t messagenum, uint64_t source, uint64_t target, **int_t** transaction)
- *Update a mail message's parent folder in the database.* void **mail_add_forward_headers** (**server_t** *server, **stringer_t** **message, **stringer_t** *id, **int_t** mark, uint64_t signum, uint64_t sigkey)
- *headers.c* **stringer_t * mail_add_inbound_headers** (**connection_t** *con, **smtp_inbound_prefs_t** *prefs)
- *Prepend the Return-Path: and Received: headers to an inbound smtp message.* **int_t mail_add_outbound_headers** (**connection_t** *con)
- *Add a Received: header and dkim signature to an outbound relayed smtp message.* **bool_t mail_add_required_headers** (**connection_t** *con, **smtp_message_t** *message)
- *Generate any missing required headers for an smtp message object.* **stringer_t * mail_header_fetch_all** (**stringer_t** *header, **stringer_t** *key)
- *Get all the values that match a named header line in a message header.* **stringer_t * mail_header_fetch_cleaned** (**stringer_t** *header, **stringer_t** *key)

- Fetch the value of a specified line from a mail header, performing whitespace and line cleaning. **placer_t mail_header_pop** (**stringer_t** *header, **size_t** *position)
- Pop the next header line from a mail message header. **bool_t mail_headers** (**smtp_message_t** *message)
- Parse an smtp message header and store the To, From, Date, and Subject header values. **void mail_mod_subject** (**stringer_t** **message, **chr_t** *label)
- Prepend text to a Subject header line inside of a mail message. **placer_t mail_store_header** (**chr_t** *stream, **size_t** length)
- Return a placer pointing to the trimmed value of a mail header. **mail_message_t * mail_load_message** (**meta_message_t** *meta, **meta_user_t** *user, **server_t** *server, **bool_t** parse)
- **load_message.c** **mail_message_t * mail_load_message_top** (**meta_message_t** *meta, **meta_user_t** *user, **server_t** *server, **uint64_t** lines, **bool_t** parse)
- Get the top of a mail message, up to a specified maximum number of lines of content. **stringer_t * mail_load_header** (**meta_message_t** *meta, **meta_user_t** *user)
- Get the header of a message, checking first in the cache and then on disk. **stringer_t * mail_mime_boundary** (**placer_t** header)
- **mime.c** **placer_t mail_mime_child** (**placer_t** body, **stringer_t** *boundary, **uint32_t** child)
- Get a placer pointing to the specified child inside a MIME body. **stringer_t * mail_mime_content_encoding** (**placer_t** header)
- Get the content encoding type from a mime header. **stringer_t * mail_mime_content_id** (**placer_t** header)
- Get the content id from a mime header. **uint32_t mail_mime_count** (**placer_t** body, **stringer_t** *boundary)
- Count the number of instances of a boundary string inside a MIME body. **int_t mail_mime_encoding** (**placer_t** header)
- Get the encoding type from a MIME header. **void mail_mime_free** (**mail_mime_t** *mime)
- Free a mail mime object and its underlying data, and recursively free its children parts. **placer_t mail_mime_header** (**stringer_t** *part)
- Get a placer pointing to a mime header in a mime part. **mail_mime_t * mail_mime_part** (**stringer_t** *part, **uint32_t** recursion)
- Parse a block of data into a mail mime object. **array_t * mail_mime_split** (**placer_t** body, **stringer_t** *boundary)
- Split a mime body into an array of children by a boundary string. **int_t mail_mime_type** (**placer_t** header)
- Get the content type from a MIME header. **stringer_t * mail_mime_type_group** (**placer_t** header)
- Get the value of the Content-Type header from a mime header. **array_t * mail_mime_type_parameters** (**placer_t** header)
- Get an array of the key/value pairs of parameters passed to the value of the mime Content-Type header. **stringer_t * mail_mime_type_parameters_key** (**stringer_t** *parameter)
- Get the key name of a mime header line parameter. **stringer_t * mail_mime_type_parameters_value** (**stringer_t** *parameter)
- Get the value of a mime header line parameter. **stringer_t * mail_mime_type_sub** (**placer_t** header)
- Get the subtype of the Content-Type header value from a mime header. **int_t mail_mime_update** (**mail_message_t** *message)
- Re-parse a mail message's data as a mime part, freeing any existing mime part(s) that may have already existed. **media_type_t * mail_mime_get_media_type** (**chr_t** *extension)
- Get the media type for a given file extension. **stringer_t * mail_mime_generate_boundary** (**array_t** *parts)
- Generate a MIME boundary string that is unique to a collection of content. **stringer_t * mail_mime_encode_part** (**stringer_t** *data, **stringer_t** *filename, **stringer_t** *boundary)
- Encode a MIME part for a provided block of data (file attachment) with the specified filename. **stringer_t * mail_mime_get_smtp_envelope** (**stringer_t** *from, **inx_t** *tos, **inx_t** *ccs, **inx_t** *bccs, **stringer_t** *subject, **stringer_t** *boundary, **bool_t** attached)
- Get smtp envelope data for an outbound message sent by a webmail client. **mail_message_t * mail_message** (**stringer_t** *text)
- **objects.c** **smtp_message_t * mail_create_message** (**stringer_t** *text)
- Create an smtp message object out of a buffer of raw data supplied via the smtp DATA command. **void mail_destroy** (**mail_message_t** *message)
- Free a mail message and all its underlying data. **void mail_destroy_message** (**smtp_message_t** *message)

- *Destroy an smtp message and free all of its underlying data.* void **mail_setup_basic** (**basic_message_t** *message, **stringer_t** *text)
- **stringer_t** * **mail_extract_address** (**stringer_t** *address)
- *parsing.c* **placer_t** * **mail_domain_get** (**stringer_t** *address, **placer_t** *output)
- *Get the domain portion of an email address.* **chr_t** * **mail_message_path** (uint64_t number, **chr_t** *server)
- *paths.c* **bool_t** **mail_create_directory** (uint64_t number, **chr_t** *server)
- *Create the on-disk directory structure necessary to hold a given message's file data.* **int_t** **mail_path_finder** (**chr_t** *string)
- **bool_t** **mail_remove_message** (uint64_t usernum, uint64_t messagenum, uint32_t size, **chr_t** *server)
- *remove_message.c* **stringer_t** * **mail_build_signature** (**server_t** *server, **int_t** content_type, **int_t** content_encoding, uint64_t signum, uint64_t sigkey, **int_t** disposition)
- *signatures.c* **int_t** **mail_discover_encoding** (**stringer_t** *header)
- *Get the value of the Content-Transfer-Encoding header in a mail message header.* **size_t** **mail_discover_insertion_point** (**stringer_t** *message, **stringer_t** *part, **int_t** type)
- *Find the position in a mail message where a custom message can be inserted.* **int_t** **mail_discover_type** (**stringer_t** *header)
- *Get the value of the Content-Type header in a mail message header.* **stringer_t** * **mail_extract_tag** (**chr_t** *stream, **size_t** length)
- *Extract an html tag from a data buffer, searching backwards.* **stringer_t** * **mail_get_boundary** (**stringer_t** *header)
- *Get the boundary string from a message header.* **stringer_t** * **mail_get_chunk** (**stringer_t** *message, **stringer_t** *boundary, **int_t** chunk)
- *Get a specified chunk (mime part) of a multipart mime message.* **stringer_t** * **mail_insert_chunk_base64** (**server_t** *server, **stringer_t** *message, **stringer_t** *part, uint64_t signum, uint64_t sigkey, **int_t** disposition, **int_t** type, **int_t** encoding)
- *Insert a spam signature training link into a base64-encoded message part.* **stringer_t** * **mail_insert_chunk_text** (**server_t** *server, **stringer_t** *message, **stringer_t** *part, uint64_t signum, uint64_t sigkey, **int_t** disposition, **int_t** type, **int_t** encoding)
- *Insert a spam signature training link into a plain text message part.* **int_t** **mail_modify_part** (**server_t** *server, **mail_message_t** *message, **stringer_t** *part, uint64_t signum, uint64_t sigkey, **int_t** disposition, **int_t** recursion)
- *Insert a spam signature training link into a specified part of a mime message.* void **mail_signature_add** (**mail_message_t** *message, **server_t** *server, uint64_t signum, uint64_t sigkey, **int_t** disposition)
- *Insert a spam signature training link into a mail message.* uint64_t **mail_copy_message** (uint64_t usernum, uint64_t original, **chr_t** *server, uint32_t size, uint64_t foldernum, uint32_t status, uint64_t signum, uint64_t sigkey, uint64_t created)
- *store_message.c* **int_t** **mail_move_message** (uint64_t usernum, uint64_t messagenum, uint64_t source, uint64_t target)
- *Move a message to a new folder in the database.* uint64_t **mail_store_message** (uint64_t usernum, **stringer_t** *pubkey, uint64_t foldernum, uint32_t *status, uint64_t signum, uint64_t sigkey, **stringer_t** *message)

Store a mail message, with its meta-information in the database, and the contents persisted to disk.

Detailed Description

Functions used to interface with and manage message data.

Definition in file **mail.h**.

Define Documentation

#define MAIL_MIME_RECURSION_LIMIT 16

Definition at line 16 of file mail.h.

Referenced by mail_mime_part(), and portal_message_body().

#define MAIL_SIGNATURES_RECURSION_LIMIT 16

Definition at line 17 of file mail.h.

Referenced by mail_modify_part().

Function Documentation

void mail_add_forward_headers (server_t * server, stringer_t ** message, stringer_t * id, int_t mark, uint64_t signum, uint64_t sigkey)

headers.c

headers.c

Note:

This function will prepend tags to the Subject line like "JUNK", "INFECTED", "SPOOFED", etc. The following headers will be stripped from the message: Sender, Return-Path, DKIM-Signature, and DomainKey-Signature. This daemon will be reinserted into the headers as the origin of the Sender: header value. If signum and sigkey are both set, then a spam training url will also be inserted into the message. Finally, if **magma.dkim.enabled** is set, a dkim signature will be added to the message.

Parameters:

server the server object of the web server where the teacher application is hosted.

message the address of a managed string containing the message data that will be set to the modified message data on success.

id a managed string containing the message id (for dkim).

mark a bitmask of marks to be tagged in the message subject (SMTP_MARK_SPAM, SMTP_MARK_VIRUS, SMTP_MARK_SPOOF, SMTP_MARK_RBL, and SMTP_MARK_PHISH).

signum the optional spam signature number referenced by the teacher url.

sigkey the optional spam signature key for client verification in the teacher app.

Returns:

This function returns no value.

Definition at line 732 of file headers.c.

References CONSTANT, magma_t::dkim, dkim_create(), magma_t::enabled, HEAP, JOINTED, log_pedantic, magma, mail_destroy(), mail_header_end(), mail_header_pop(), MAIL_MARK_JUNK, mail_message(), mail_mod_subject(), mail_signature_add(), MAPPED_T, mm_cmp_ci_eq(), pl_empty(), PLACER, placer_t, SMTP_MARK_PHISH, SMTP_MARK_RBL, SMTP_MARK_SPAM, SMTP_MARK_SPOOF, SMTP_MARK_VIRUS, st_alloc_opts(), st_append_opts(), st_char_get(), st_cleanup(), st_cmp_ci_starts(), st_data_get(), st_dupe(), st_dupe_opts(), st_free(), st_length_get(), st_merge_opts(), status, and mail_message_t::text.

Referenced by smtp_forward_message().

stringer_t* mail_add_inbound_headers (connection_t * con, smtp_inbound_prefs_t * prefs)

Prepend the Return-Path: and Received: headers to an inbound smtp message.

Parameters:

con the connection across which the inbound smtp message was accepted.

prefs the smtp inbound preferences corresponding to the connection, with the rcptto field populated.

Returns:

NULL on failure, or a managed string containing the message data preceded by the Return-Path and Received headers on success.

Definition at line 640 of file headers.c.

References con_addr_presentation(), con_reverse_check(), CONSTANT, server_t::domain, smtp_session_t::esmtplib, HEAP, smtp_session_t::helo, smtp_message_t::id, JOINTED, log_pedantic, smtp_session_t::mailfrom, MANAGEDBUF, MAPPED_T, smtp_session_t::message, smtp_inbound_prefs_t::rcptto, connection_t::server, connection_t::smtp, st_cmp_ci_eq(), st_cmp_cs_eq(), st_merge_opts(), and smtp_message_t::text.

Referenced by smtp_accept_message().

int_t mail_add_outbound_headers (connection_t * con)

Add a Received: header and dkim signature to an outbound relayed smtp message.

Note:

If the message already has a Received header, the dkim signature will be inserted right before the first instance.

Parameters:

con the connection across which the outbound smtp message is being sent.

Returns:

NULL on failure, or a managed string containing the message data preceded by the Received header and including the dkim signature on success.

Definition at line 849 of file headers.c.

References smtp_recipients_t::address, con_addr_presentation(), con_reverse_check(), magma_t::dkim, dkim_create(), server_t::domain, magma_t::enabled, smtp_session_t::esmtplib, HEAP, smtp_session_t::helo, smtp_message_t::id, JOINTED, log_pedantic, magma, mail_header_end(), mail_header_pop(), MANAGEDBUF, MAPPED_T, smtp_session_t::message, mm_cmp_ci_eq(), smtp_session_t::num_recipients, smtp_session_t::out_prefs, pl_empty(), PLACER, placer_t, smtp_outbound_prefs_t::recipients, connection_t::server, connection_t::smtp, st_char_get(), st_cleanup(), st_cmp_cs_eq(), st_data_get(), st_dupe(), st_empty(), st_free(), st_import(), st_length_get(), st_merge_opts(), and smtp_message_t::text.

Referenced by smtp_relay_message().

bool_t mail_add_required_headers (connection_t * con, smtp_message_t * message)

Generate any missing required headers for an smtp message object.

Note:

These required headers include To, From, Subject and Date, and if they are absent, they will be generated from the specified connection's inbound or outbound preferences, accordingly.

Parameters:

con a pointer to the connection object across which the smtp message was received.
message the smtp message object to be updated for any missing required headers.

Returns:

true on success or false on failure.

Definition at line 459 of file headers.c.

References smtp_recipients_t::address, smtp_session_t::authenticated, smtp_message_t::date, smtp_message_t::from, smtp_message_t::header_length, HEAP, smtp_session_t::in_prefs, JOINTED, length, log_pedantic, mail_header_end(), mail_headers(), smtp_session_t::mailfrom, MAPPED_T, smtp_recipients_t::next, smtp_inbound_prefs_t::next, smtp_session_t::out_prefs, PLACER, smtp_inbound_prefs_t::rcptto, smtp_outbound_prefs_t::recipients, connection_t::smtp, st_alloc, st_avail_get(), st_char_get(), st_cleanup(), st_empty(), st_free(), st_import(), st_length_get(), st_length_set(), st_merge, st_merge_opts(), smtp_message_t::subject, smtp_message_t::text, and smtp_message_t::to.

Referenced by smtp_data().

stringer_t* mail_build_signature (server_t * server, int_t content_type, int_t content_encoding, uint64_t signum, uint64_t sigkey, int_t disposition)

signatures.c

signatures.c

Parameters:

server the server object of the web server where the teacher application is hosted.
type MESSAGE_TYPE_HTML to generate an html-based signature, or any other value for plain text.
content_encoding if MESSAGE_ENCODING_QUOTED_PRINTABLE, set the encoding type to qp.
signum the spam signature number referenced by the teacher url.
sigkey the spam signature key for client verification in the teacher app.
disposition if 0, the message disposition is "innocent"; otherwise, the disposition specifies "spam".

Returns:

NULL on failure, or a newly allocated managed string containing the desired mail signature on success.

Definition at line 25 of file signatures.c.

References server_t::domain, length, log_pedantic, MESSAGE_ENCODING_QUOTED_PRINTABLE, MESSAGE_TYPE_HTML, qp_encode(), st_alloc, st_char_get(), st_cleanup(), st_free(), st_length_get(), st_merge, st_sprint(), and uint64_digits().

Referenced by mail_insert_chunk_base64(), and mail_insert_chunk_text().

void mail_cache_destroy (void * holder)

cache.c

cache.c

Parameters:

holder a pointer to the cached mail message object to be freed.

Returns:

This function returns no value.

Definition at line 22 of file cache.c.

References `mm_free()`, `st_cleanup()`, and `mail_cache_t::text`.

Referenced by `mail_cache_start()`, and `mail_cache_thread_stop()`.

`stringer_t* mail_cache_get (uint64_t messagenum)`

Attempt to retrieve the contents of a message from the thread's cached data.

Note:

The thread's cache only has the capacity to store a single message.

Parameters:

messagenum the id of the message to be retrieved.

Returns:

NULL on failure or a managed string containing the message data on success.

Definition at line 83 of file `cache.c`.

References `mail_cache_t::messagenum`, `st_dupe()`, and `mail_cache_t::text`.

Referenced by `mail_load_message()`.

`void mail_cache_reset (void)`

Reset the thread's mail cache and free any held message.

Returns:

This function returns no value.

Definition at line 102 of file `cache.c`.

References `mail_cache_t::messagenum`, `st_cleanup()`, and `mail_cache_t::text`.

Referenced by `imap_session_destroy()`, and `pop_session_destroy()`.

`void mail_cache_set (uint64_t messagenum, stringer_t * text)`

Set the contents of the thread's mail cache.

Parameters:

messagenum the numerical id of the message to be cached. *text* a managed string containing the contents of the specified message to be cached.

Returns:

This function returns no value.

Definition at line 121 of file `cache.c`.

References `CONTIGUOUS`, `HEAP`, `log_pedantic`, `MANAGED_T`, `mail_cache_t::messagenum`, `mm_alloc()`, `mm_free()`, `st_cleanup()`, `st_dupe_opts()`, and `mail_cache_t::text`.

Referenced by `mail_load_message()`.

`bool_t mail_cache_start (void)`

Initialize the thread-specific data key used for the mail message cache.

Returns:

true on success or false on failure.

Definition at line 38 of file cache.c.

References log_pedantic, and mail_cache_destroy().

Referenced by process_start().

void mail_cache_stop (void)

Destroy the thread-specific data key used for the mail message cache.

Returns:

This function returns no value.

Definition at line 52 of file cache.c.

References log_pedantic.

Referenced by process_stop().

void mail_cache_thread_stop (void)

Free the thread-specific data for the mail message cache.

Returns:

This function returns no value.

Definition at line 65 of file cache.c.

References mail_cache_destroy().

Referenced by thread_stop().

uint64_t mail_copy_message (uint64_t *usernum*, uint64_t *original*, chr_t * *server*, uint32_t *size*, uint64_t *foldernum*, uint32_t *status*, uint64_t *signum*, uint64_t *sigkey*, uint64_t *created*)

store_message.c

store_message.c

Parameters:

usernum the numerical id of the user to whom the mail message belongs.

original the numerical id of the mail message to be copied.

server a pointer to a null-terminated string containing the name of the server where the message contents are stored.

size the size, in bytes, of the mail message to be copied.

foldernum the numerical id of the folder to become the parent folder of the message copy.

signum the spam signature for the message.

sigkey the spam key for the message.

created the UNIX timestamp of when the message was created.

Returns:

0 on failure, or the ID of the copy of the mail message in the database on success.

Definition at line 222 of file store_message.c.

References log_error, log_pedantic, mail_create_directory(), mail_db_insert_duplicate_message(), mail_message_path(), ns_free(), tran_commit(), tran_rollback(), and tran_start().

Referenced by imap_message_copier(), and meta_messages_copier().

uint32_t mail_count_received (stringer_t * message)

counters.c

counters.c

Parameters:

message a managed string containing the mail message to be scanned.

Returns:

the number of matching lines found, or 0 on failure.

Definition at line 20 of file counters.c.

References log_pedantic, PLACER, st_cmp_ci_starts(), and st_empty_out().

Referenced by smtp_data().

bool_t mail_create_directory (uint64_t number, chr_t * server)

Create the on-disk directory structure necessary to hold a given message's file data.

Parameters:

number the mail message id.

server the hostname of the server where the message data resides or if NULL, the default server.

Returns:

true on success or false on failure.

Definition at line 71 of file paths.c.

References magma_t::active, log_error, magma, magma_t::root, st_char_get(), st_length_int(), and magma_t::storage.

Referenced by mail_copy_message(), and mail_store_message_data().

smtp_message_t* mail_create_message (stringer_t * text)

Create an smtp message object out of a buffer of raw data supplied via the smtp DATA command.

Note:

This function will also generate a random message ID.

Parameters:

text a pointer to a managed string containing the smtp data to be parsed.

Returns:

NULL on failure or a pointer to the newly initialized smtp message object wrapping the data on success.

Definition at line 177 of file objects.c.

References `smtp_message_t::header_length`, `smtp_message_t::id`, `log_pedantic`, `mail_header_end()`, `mail_headers()`, `mm_alloc()`, `mm_free()`, `rand_choices()`, `st_free()`, and `smtp_message_t::text`.

Referenced by `smtp_data()`.

`bool_t mail_db_delete_message (uint64_t usernum, uint64_t messagenum, uint32_t size, int_t transaction)`

`datatier.c`

`datatier.c`

Parameters:

usernum the user id to whom the target mail message belongs.

messagenum the message id of the mail message to be deleted.

size the storage size, in bytes, of the message to be deleted.

transaction the mysql connection id on which to execute the statements.

Returns:

0 on failure or 1 on success.

Definition at line 46 of file `datatier.c`.

References `log_error`, `mm_wipe()`, and `stmt_exec_affected_conn()`.

Referenced by `mail_remove_message()`.

`void mail_db_hide_message (uint64_t messagenum)`

Set a message invisible in the database.

Parameters:

messagenum the message id of the mail message to be hidden.

Returns:

This function returns no value.

Definition at line 20 of file `datatier.c`.

References `mm_wipe()`, and `stmt_exec()`.

Referenced by `mail_load_header()`, and `mail_load_message()`.

`uint64_t mail_db_insert_duplicate_message (uint64_t usernum, uint64_t foldernum, uint32_t status, uint32_t size, uint64_t signum, uint64_t sigkey, uint64_t created, int_t transaction)`

Insert a duplicate entry for a message in the database.

Note:

This function will also update the user's storage quota information in the database.

Parameters:

usernum the numerical id of the user that owns the message.

foldernum the numerical id of the parent folder containing the message.

status the status flags value for the message.

size the size, in bytes, of the mail message on disk.

signum the spam signature for the message.
sigkey the spam key for the message.
created the UNIX timestamp of when the message was created.
transaction the transaction id for the database operation, in case the caller wants to roll back the transaction.

Returns:

NULL on failure, or the ID of the newly inserted message on success.

Definition at line 309 of file `datatier.c`.

References `magma_t::active`, `ISNULL`, `log_pedantic`, `magma`, `MAIL_MARK_JUNK`, `mm_wipe()`, `st_char_get()`, `st_length_get()`, `stmt_exec_conn()`, `stmt_insert_conn()`, and `magma_t::storage`.

Referenced by `mail_copy_message()`.

`uint64_t mail_db_insert_message (uint64_t usernum, uint64_t foldernum, uint32_t status, uint32_t size, uint64_t signum, uint64_t sigkey, int_t transaction)`

Insert a mail message into the database.

Note:

This function will also update the user's storage quota information in the database.

Parameters:

usenum the numerical id of the user to whom the mail message will belong.
foldernum the numerical id of the folder that will be the parent folder of the message.
status the status flags of the mail message.
size the size, in bytes, of the mail message on disk.
signum the spam signature for the message.
sigkey the spam key for the message.
transaction the transaction id for the database operation, in case the caller wants to roll back the transaction.

Returns:

0 on failure or the id of the newly inserted mail message on success.

Definition at line 177 of file `datatier.c`.

References `magma_t::active`, `ISNULL`, `log_pedantic`, `magma`, `MAIL_MARK_JUNK`, `mm_wipe()`, `st_char_get()`, `st_length_get()`, `stmt_exec_conn()`, `stmt_insert_conn()`, and `magma_t::storage`.

Referenced by `mail_store_message()`.

`int_t mail_db_update_message_folder (uint64_t usernum, uint64_t messagenum, uint64_t source, uint64_t target, int64_t transaction)`

Update a mail message's parent folder in the database.

usenum the numerical id of the user to whom the mail message belongs. *messagenum* the numerical id of the target mail message of the operation. *source* the numerical id of the parent folder in which the mail message currently resides. *target* the numerical id of the destination folder which is to be the new parent of the mail message. *transaction* a transaction id for the database operation, in case the caller needs to roll back changes on failure.

Returns:

-1 on failure, 0 if the target message could not be located in the database, or 1 on success.

Definition at line 105 of file `datatier.c`.

References log_pedantic, mm_wipe(), mysql_stmt_errno_d, mysql_stmt_error_d, pool_get_obj(), sql_pool, and stmt_exec_affected_conn().

Referenced by mail_move_message().

void mail_destroy (mail_message_t * message)

Free a mail message and all its underlying data.

Parameters:

message a pointer to the mail message object to be freed.

Returns:

This function returns no value.

Definition at line 85 of file objects.c.

References mail_mime_free(), mail_message_t::mime, mm_free(), st_cleanup(), and mail_message_t::text.

Referenced by imap_fetch_body(), imap_fetch_message(), imap_fetch_return_header(), imap_fetch_return_message(), imap_fetch_return_mime(), imap_fetch_return_text(), imap_search_messages(), mail_add_forward_headers(), mail_load_header(), pop_retr(), pop_top(), and portal_endpoint_messages_load().

void mail_destroy_header (stringer_t * header)

cleanup.c

cleanup.c

Parameters:

header a managed string containing the mail header to be freed.

Returns:

This function returns no value.

Definition at line 20 of file cleanup.c.

References st_cleanup().

Referenced by imap_fetch_body(), imap_fetch_message(), imap_fetch_return_message(), imap_fetch_return_mime(), imap_fetch_return_text(), and imap_search_messages().

void mail_destroy_message (smtp_message_t * message)

Destroy an smtp message and free all of its underlying data.

Parameters:

message a pointer to the smtp message to be destroyed.

Returns:

This function returns no value.

Definition at line 20 of file objects.c.

References smtp_message_t::id, mm_free(), st_cleanup(), and smtp_message_t::text.

Referenced by smtp_data(), smtp_session_destroy(), and smtp_session_reset().

int_t mail_discover_encoding (stringer_t * header)

Get the value of the Content-Transfer-Encoding header in a mail message header.

Note:

Possible return values include MESSAGE_ENCODING_QUOTED_PRINTABLE, MESSAGE_ENCODING_BASE64, MESSAGE_ENCODING_8BIT, and MESSAGE_ENCODING_7BIT.

Parameters:

header a managed string containing the mail message header to be parsed.

Returns:

the MESSAGE_ENCODING code of the mail transfer encoding type, or MESSAGE_ENCODING_7BIT by default.

Definition at line 176 of file signatures.c.

References content, mail_header_fetch_all(), MESSAGE_ENCODING_7BIT, MESSAGE_ENCODING_8BIT, MESSAGE_ENCODING_BASE64, MESSAGE_ENCODING_QUOTED_PRINTABLE, MESSAGE_ENCODING_UNKNOWN, mm_cmp_ci_eq(), PLACER, st_char_get(), st_free(), and st_length_get().

Referenced by mail_modify_part().

size_t mail_discover_insertion_point (stringer_t * message, stringer_t * part, int_t type)

Find the position in a mail message where a custom message can be inserted.

Note:

The part parameter is expected to be a placer pointing into the contents of message. If the encoding type is not html, the insertion point is determined to be at the end of the part. If the encoding type is html, the following rules are followed: 1. Scanning backwards from the end of the message, skip trailing whitespace get the next html tag. 2. If that tag is NOT </html> or </body>, insert the signature AFTER it. 3. If that tag IS </html> or </body>, insert the signature right before it closes.

Parameters:

message a managed string containing the mail message body to be parsed.

part a managed string (placer) containing the part of the message where the custom message should be inserted.

type the encoding type of the message (MESSAGE_TYPE_HTML or other).

Returns:

the zero-based index of the position in the specified message where the custom message can be inserted.

Definition at line 301 of file signatures.c.

References length, mail_extract_tag(), MESSAGE_TYPE_HTML, PLACER, st_char_get(), st_cmp_ci_eq(), st_data_get(), st_free(), and st_length_get().

Referenced by mail_insert_chunk_base64(), and mail_insert_chunk_text().

int_t mail_discover_type (stringer_t * header)

Get the value of the Content-Type header in a mail message header.

Note:

Possible return values include MESSAGE_TYPE_MULTI_ALTERNATIVE, MESSAGE_TYPE_MULTI_RELATED, MESSAGE_TYPE_MULTI_MIXED, MESSAGE_TYPE_HTML, MESSAGE_TYPE_MULTI_UNKOWN, and MESSAGE_TYPE_PLAIN.

Parameters:

header a managed string containing the mail message header to be parsed.

Returns:

the MESSAGE_TYPE code of the mail content type, or MESSAGE_TYPE_PLAIN by default.

Definition at line 125 of file signatures.c.

References content, mail_header_fetch_all(), MESSAGE_TYPE_HTML, MESSAGE_TYPE_MULTI_ALTERNATIVE, MESSAGE_TYPE_MULTI_MIXED, MESSAGE_TYPE_MULTI_RELATED, MESSAGE_TYPE_MULTI_UNKOWN, MESSAGE_TYPE_PLAIN, mm_cmp_ci_eq(), PLACER, st_char_get(), st_free(), and st_length_get().

Referenced by mail_modify_part().

placer_t* mail_domain_get (stringer_t * address, placer_t * output)

Get the domain portion of an email address.

Parameters:

address a managed string containing the email address to be parsed.

output a pointer to a placer to receive the domain portion of the email address.

Returns:

NULL on failure or a pointer to the output of the domain extraction on success.

Definition at line 21 of file parsing.c.

References tok_get_count_st(), and tok_get_st().

Referenced by smtp_check_authorized_from(), and spf_check().

stringer_t* mail_extract_address (stringer_t * address)

parsing.c

parsing.c

Note:

The preference of this function is first to try to find the email addressed enclosed in <>. No valid email address may be enclosed in () or "".

Parameters:

address a managed string containing the buffer from which the email address will be extracted.

Returns:

NULL on failure or a managed string containing the email address on success.

Definition at line 37 of file parsing.c.

References comment, length, log_pedantic, st_alloc, st_char_get(), st_data_get(), st_empty_out(), and st_length_set().

Referenced by smtp_data_outbound().

stringer_t* mail_extract_tag (chr_t * *stream*, size_t *length*)

Extract an html tag from a data buffer, searching backwards.

Warning:

Remember that this function takes the end and not the start of a buffer!

Note:

The extracted tag contains the enclosing brackets and only printable characters (no whitespace).

Parameters:

stream the end of a data buffer containing the html data to be parsed.

length the size, in bytes, of the data buffer to be parsed.

Returns:

NULL on failure, or a managed string containing the printable contents of the nearest html tag on success.

Definition at line 223 of file signatures.c.

References log_pedantic, st_alloc, st_char_get(), st_free(), and st_length_set().

Referenced by mail_discover_insertion_point().

stringer_t* mail_get_boundary (stringer_t * *header*)

Get the boundary string from a message header.

Note:

This function works by parsing the Content-Type header if it exists, falling back to the entire header otherwise. The returned boundary string includes a trailing "--".

Parameters:

header a managed string containing the message header.

Returns:

NULL on failure or a managed string containing the boundary string on success.

Definition at line 613 of file signatures.c.

References content, length, log_pedantic, mail_header_fetch_all(), PLACER, st_char_get(), st_cleanup(), st_data_get(), st_length_get(), st_merge, and st_search_ci().

Referenced by mail_modify_part().

stringer_t* mail_get_chunk (stringer_t * *message*, stringer_t * *boundary*, int_t *chunk*)

Get a specified chunk (mime part) of a multipart mime message.

Parameters:

message a managed string containing the mime message to be parsed.

boundary a managed string containing the boundary used to split the multipart mime message.

chunk the one-index based chunk to be retrieved from the multipart message

Returns:

NULL on failure or a placer containing the specified chunk on success.

Definition at line 546 of file signatures.c.

References `length`, `log_pedantic`, `mm_cmp_cs_eq()`, `PLACER`, `st_char_get()`, `st_length_get()`, and `st_search_cs()`.

Referenced by `mail_modify_part()`.

size_t mail_header_end (stringer_t * *message*)

Get the number of bytes taken up by a mail header.

Parameters:

message a managed string containing the full smtp mail message.

Returns:

0 on failure, or the number of bytes occupied by the mail header.

Definition at line 61 of file `counters.c`.

References `length`, `log_pedantic`, `st_char_get()`, and `st_length_get()`.

Referenced by `mail_add_forward_headers()`, `mail_add_outbound_headers()`, `mail_add_required_headers()`, `mail_create_message()`, `mail_headers()`, `mail_load_header()`, `mail_message()`, `mail_mod_subject()`, `mail_setup_basic()`, and `smtp_check_filters()`.

stringer_t* mail_header_fetch_all (stringer_t * *header*, stringer_t * *key*)

Get all the values that match a named header line in a message header.

Parameters:

header a managed string containing the message header.

key a managed string containing the header name to be searched for.

Returns:

NULL on failure or a managed string containing all the matching header line values separated by newlines.

Definition at line 154 of file `headers.c`.

References `mm_cmp_ci_eq()`, `PLACER`, `st_char_get()`, `st_empty()`, `st_free()`, `st_import()`, `st_length_get()`, and `st_merge`.

Referenced by `imap_fetch_body_mime()`, `imap_search_messages_header()`, `mail_discover_encoding()`, `mail_discover_type()`, `mail_get_boundary()`, `mail_mime_boundary()`, and `smtp_check_filters()`.

stringer_t* mail_header_fetch_cleaned (stringer_t * *header*, stringer_t * *key*)

Fetch the value of a specified line from a mail header, performing whitespace and line cleaning.

Note:

This function handles mail header values that span new lines, although the output excludes line breaks and consecutive space characters.

Parameters:

header a managed string containing the full header of the target mail message.

key a managed string containing the name of the desired header name to be extracted, excluding the trailing ":"

Returns:

NULL on failure, or a managed string containing the cleaned value of the specified mail header line on success.

Definition at line 22 of file headers.c.

References PLACER, st_alloc, st_char_get(), st_cmp_ci_starts(), st_empty(), st_length_get(), and st_length_set().

Referenced by imap_fetch_bodystructure(), imap_fetch_envelope(), imap_search_messages_date(), mail_mime_content_encoding(), mail_mime_content_id(), mail_mime_encoding(), mail_mime_type(), mail_mime_type_group(), mail_mime_type_parameters(), mail_mime_type_sub(), portal_endpoint_messages_list(), and portal_message_header().

placer_t mail_header_pop (stringer_t * header, size_t * position)

Pop the next header line from a mail message header.

Parameters:

header a managed string containing the entire mail message header.

position a pointer to a size_t that contains the tracker index into the entire header for the pop operation, and will also receive the new value of the tracker for the subsequent call, when a new header line is located.

Returns:

pl_null() on error or a placer pointing to the next mail message subject line on success.

Definition at line 243 of file headers.c.

References pl_init(), pl_null(), st_char_get(), st_empty(), and st_length_get().

Referenced by imap_fetch_body_header(), mail_add_forward_headers(), mail_add_outbound_headers(), and mail_mod_subject().

bool_t mail_headers (smtp_message_t * message)

Parse an smtp message header and store the To, From, Date, and Subject header values.

Note:

The found header fields will be stored inside the same smtp message object passed by the caller as input.

Parameters:

message a pointer to a partially populated smtp message object that contains the raw message data to be parsed.

Returns:

true on success or false on failure.

Definition at line 325 of file headers.c.

References smtp_message_t::date, smtp_message_t::from, smtp_message_t::header_length, length, log_pedantic, mail_header_end(), mail_store_header(), mm_cmp_ci_eq(), st_char_get(), st_length_get(), smtp_message_t::subject, smtp_message_t::text, and smtp_message_t::to.

Referenced by mail_add_required_headers(), and mail_create_message().

stringer_t* mail_insert_chunk_base64 (server_t * server, stringer_t * message, stringer_t * part, uint64_t signum, uint64_t sigkey, int_t disposition, int_t type, int_t encoding)

Insert a spam signature training link into a base64-encoded message part.

See also:

mail_discover_insertion_point()

Note:

This is similar to **mail_insert_chunk_text()** except the part has to be decoded and then re-encoded after the training link is inserted.

Parameters:

server the server object of the web server where the teacher application is hosted.
message a managed string containing the base64-encoded message body to be parsed.
part a managed string (placer) containing the part of the message where the signature should be inserted.
signum the spam signature number referenced by the teacher url.
sigkey the spam signature key for client verification in the teacher app.
disposition if 0, the message disposition is "innocent"; otherwise, the disposition specifies "spam".
type the encoding type of the message (MESSAGE_TYPE_HTML or other).
encoding if MESSAGE_ENCODING_QUOTED_PRINTABLE, set the encoding type to qp.

Returns:

NULL on failure or a managed string containing the base64-encoded message with the inserted signature training link on success.

Definition at line 366 of file signatures.c.

References `base64_decode()`, `base64_encode()`, `length`, `log_pedantic`, `mail_build_signature()`, `mail_discover_insertion_point()`, `PLACER`, `st_char_get()`, `st_cleanup()`, `st_data_get()`, `st_free()`, `st_length_get()`, and `st_merge`.

Referenced by `mail_modify_part()`.

stringer_t* mail_insert_chunk_text (server_t * server, stringer_t * message, stringer_t * part, uint64_t signum, uint64_t sigkey, int_t disposition, int_t type, int_t encoding)

Insert a spam signature training link into a plain text message part.

See also:

mail_discover_insertion_point()

Parameters:

server the server object of the web server where the teacher application is hosted.
message a managed string containing the message body to be parsed.
part a managed string (placer) containing the part of the message where the signature should be inserted.
signum the spam signature number referenced by the teacher url.
sigkey the spam signature key for client verification in the teacher app.
disposition if 0, the message disposition is "innocent"; otherwise, the disposition specifies "spam".
type the encoding type of the message (MESSAGE_TYPE_HTML or other).
encoding if MESSAGE_ENCODING_QUOTED_PRINTABLE, set the encoding type to qp.

Returns:

NULL on failure or a managed string containing the message with the inserted signature training link on success.

Definition at line 507 of file signatures.c.

References `log_pedantic`, `mail_build_signature()`, `mail_discover_insertion_point()`, `PLACER`, `st_char_get()`, `st_empty()`, `st_free()`, `st_length_get()`, and `st_merge`.

Referenced by `mail_modify_part()`.

stringer_t* mail_load_header (meta_message_t * meta, meta_user_t * user)

Get the header of a message, checking first in the cache and then on disk.

Note:

The file data is first unencrypted and decompressed, according to the file header flags. When extracted, the header's Subject line is branded with any applicable labels such as JUNK, INFECTED, SPOOFED, BLACKHOLED, PHISHING.

Parameters:

meta the meta message object of the message to be queried.

user the meta user object of the user that owns the message.

Returns:

NULL on failure or a managed string containing the message's header on success.

Definition at line 23 of file load_message.c.

References cache_add(), cache_get(), compress_block_length(), decompress_block_lzo(), decompress_lzo(), ecies_decrypt(), ECIES_PRIVATE_BINARY, FMESSAGE_MAGIC_1, FMESSAGE_MAGIC_2, FMESSAGE_OPT_ENCRYPTED, mail_message_t::header_length, log_pedantic, mail_db_hide_message(), mail_destroy(), mail_header_end(), mail_load_message(), MAIL_MARK_BLACKHOLED, MAIL_MARK_INFECTED, MAIL_MARK_JUNK, MAIL_MARK_PHISHING, MAIL_MARK_SPOOFED, mail_message_path(), mail_mod_subject(), MAIL_STATUS_ENCRYPTED, message_fheader_t, meta_message_t::messagenum, mm_alloc(), ns_free(), OBJECT_MESSAGES, PLACER, serial_increment(), meta_message_t::server, st_char_get(), st_free(), st_import(), st_length_get(), st_length_set(), meta_message_t::status, meta_user_t::storage_privkey, mail_message_t::text, and meta_user_t::usernum.

Referenced by imap_fetch_return_header(), imap_search_messages_date(), imap_search_messages_header(), and portal_endpoint_messages_list().

mail_message_t* mail_load_message (meta_message_t * meta, meta_user_t * user, server_t * server, bool_t parse)

load_message.c

load_message.c

Note:

The mail message will always, at the very least, be compressed using the lzo algorithm; however, on-disk encryption may be enabled. If parsing is enabled, a spam signature training link may be embedded in the message.

Parameters:

meta the meta message object of the message to be loaded from disk.

user the meta user object of the user that owns the requested message.

server the server object of the web server where the spam teacher application is hosted.

parse if true, the header's Subject line is branded with any applicable labels such as JUNK, INFECTED, SPOOFED, BLACKHOLED, PHISHING.

Returns:

NULL on failure or a mail message object containing the retrieved mail message data on success.

Definition at line 238 of file load_message.c.

References cache_add(), cache_get(), compress_import(), decompress_lzo(), ecies_decrypt(), ECIES_PRIVATE_BINARY, meta_user_t::flags, FMESSAGE_MAGIC_1, FMESSAGE_MAGIC_2, FMESSAGE_OPT_ENCRYPTED, mail_message_t::header_length, log_info, log_pedantic, mail_cache_get(), mail_cache_set(), mail_db_hide_message(), MAIL_MARK_BLACKHOLED, MAIL_MARK_INFECTED,

MAIL_MARK_JUNK, MAIL_MARK_PHISHING, MAIL_MARK_SPOOFED, mail_message(), mail_message_path(), mail_mod_subject(), mail_signature_add(), MAIL_STATUS_ENCRYPTED, message_fheader_t, meta_message_t::messagenum, META_USER_ENCRYPT_DATA, mm_free(), ns_free(), OBJECT_MESSAGES, PLACER, serial_increment(), meta_message_t::server, meta_message_t::sigkey, meta_message_t::signum, st_alloc, st_char_get(), st_data_get(), st_free(), st_import(), st_length_set(), meta_message_t::status, meta_user_t::storage_privkey, mail_message_t::text, and meta_user_t::usernum.

Referenced by imap_fetch_message(), imap_fetch_return_message(), imap_fetch_return_mime(), imap_fetch_return_text(), imap_search_messages_body(), imap_search_messages_text(), mail_load_header(), mail_load_message_top(), pop_retr(), and portal_endpoint_messages_load().

mail_message_t* mail_load_message_top (meta_message_t * meta, meta_user_t * user, server_t * server, uint64_t lines, bool_t parse)

Get the top of a mail message, up to a specified maximum number of lines of content.

See also:

mail_load_message()

Parameters:

meta the meta message object of the message to be loaded from disk.

user the meta user object of the user that owns the requested message.

server the server object of the web server where the spam teacher application is hosted.

lines the maximum number of lines to be retrieved from the loaded message.

parse sets the parse parameter passed to **mail_load_message()**.

Returns:

NULL on failure or a mail message object containing the retrieved mail message data (truncated if necessary) on success.

Definition at line 479 of file load_message.c.

References length, mail_load_message(), st_char_get(), st_length_get(), st_length_set(), and mail_message_t::text.

Referenced by pop_top().

mail_message_t* mail_message (stringer_t * text)

objects.c

objects.c

Parameters:

text a pointer to the managed string containing the raw (uncompressed) mail data to be parsed.

Returns:

NULL on failure or a pointer to a newly allocated mail message object with the message on success.

Definition at line 105 of file objects.c.

References mail_message_t::date, mail_message_t::from, mail_message_t::header_length, length, log_pedantic, mail_header_end(), mail_store_header(), mm_alloc(), mm_cmp_ci_eq(), mm_free(), st_char_get(), st_length_get(), mail_message_t::subject, mail_message_t::text, and mail_message_t::to.

Referenced by mail_add_forward_headers(), and mail_load_message().

bool_t mail_message_cleanup (stringer_t ** message)

Clean up the body of a message read in via smtp, enforcing magma.smtp.wrap_line_length.

Note:

This function fixes broken line separators by making sure each `\r` is followed by `\n` and vice versa. All Return-Path: header lines are also removed. New lines are begun whenever the current length of any line reaches the configuration value set in magma.smtp.wrap_line_length. The trailing dot at the end of the smtp DATA command is also stripped.

If the original message ends with `.`, it will have `\n` appended to it.

Parameters:

message a pointer to a managed string that contains the message input, and will also store the cleaned output on success.

Returns:

true on success or false on failure.

Definition at line 36 of file cleanup.c.

References HEAP, JOINTED, length, log_pedantic, magma, MAPPED_T, mm_cmp_ci_eq(), magma_t::smtp, st_alloc_opts(), st_char_get(), st_free(), st_length_get(), st_length_set(), and magma_t::wrap_line_length.

Referenced by smtp_data().

chr_t* mail_message_path (uint64_t *number*, chr_t * *server*)

paths.c

paths.c

Parameters:

number the mail message id.

server the hostname of the server where the message data resides or if NULL, the default server.

Returns:

NULL on failure, or a pointer to a null-terminated string containing the absolute file path of the specified message.

Definition at line 40 of file paths.c.

References magma_t::active, log_pedantic, magma, ns_alloc(), ns_free(), magma_t::root, st_char_get(), st_length_int(), and magma_t::storage.

Referenced by adjust_message_encryption(), mail_copy_message(), mail_load_header(), mail_load_message(), mail_remove_message(), and mail_store_message_data().

stringer_t* mail_mime_boundary (placer_t *header*)

mime.c

mime.c

Note:

This function first scans the value of Content-Type for the boundary, and then the rest of the MIME header.

Parameters:

header a placer pointing to the MIME header to be parsed.

Returns:

NULL on failure, or a pointer to a managed string containing the MIME boundary string on success.

Definition at line 622 of file mime.c.

References `content`, `length`, `log_error`, `log_pedantic`, `mail_header_fetch_all()`, `PLACER`, `st_char_get()`, `st_cleanup()`, `st_length_get()`, `st_merge`, and `st_search_ci()`.

Referenced by `mail_mime_part()`.

`placer_t mail_mime_child (placer_t body, stringer_t * boundary, uint32_t child)`

Get a placer pointing to the specified child inside a MIME body.

Parameters:

body a placer containing the body text to be parsed.

boundary a pointer to a managed string containing the boundary string to split the MIME content.

child the zero-based index of the MIME child to be located in the body text.

Returns:

`pl_null()` on failure, or a placer containing the specified MIME child on success.

Definition at line 754 of file mime.c.

References `length`, `log_pedantic`, `mm_cmp_cs_eq()`, `pl_empty()`, `pl_init()`, `pl_null()`, `st_char_get()`, `st_empty()`, and `st_length_get()`.

Referenced by `mail_mime_split()`.

`stringer_t* mail_mime_content_encoding (placer_t header)`

Get the content encoding type from a mime header.

Note:

This function parses the Content-Transfer-Encoding field of the header, returning "7bit" by default.

Parameters:

header a placer pointing to the mime header to be parsed.

Returns:

a managed string containing the content encoding type value of the header, with "7bit" as default.

Definition at line 236 of file mime.c.

References `mail_header_fetch_cleaned()`, `PLACER`, `st_char_get()`, `st_free()`, `st_import()`, and `st_length_get()`.

Referenced by `imap_fetch_bodystructure()`.

`stringer_t* mail_mime_content_id (placer_t header)`

Get the content id from a mime header.

Note:

This function parses the Content-Id field of the header, returning "7bit" by default.

Parameters:

header a placer pointing to the mime header to be parsed.

Returns:

NULL on failure, or a managed string containing the content id value of the header.

Definition at line 282 of file mime.c.

References mail_header_fetch_cleaned(), PLACER, st_char_get(), st_free(), st_import(), and st_length_get().

Referenced by imap_fetch_bodystructure().

uint32_t mail_mime_count (placer_t *body*, stringer_t * *boundary*)

Count the number of instances of a boundary string inside a MIME body.

Note:

The search is terminated if "--" is found right after the boundary string.

Parameters:

body a placer containing the body text to be parsed.

boundary a pointer to a managed string containing the boundary string for the MIME content.

Returns:

0 on failure, or the number of times the boundary string was located in the MIME body on success.

Definition at line 699 of file mime.c.

References length, log_pedantic, mm_cmp_cs_eq(), pl_empty(), st_char_get(), st_empty(), and st_length_get().

Referenced by mail_mime_split().

stringer_t* mail_mime_encode_part (stringer_t * *data*, stringer_t * *filename*, stringer_t * *boundary*)

Encode a MIME part for a provided block of data (file attachment) with the specified filename.

Note:

This function will look up the media type based on the supplied filename, and use that media type as a determination of whether the content is to be encoded as either quoted-printable or base64.

Parameters:

data a pointer to a managed string containing the body of the data to be encoded.

filename a pointer to a managed string containing the filename of the attachment for which the data was provided.

boundary a pointer to a managed string containing the boundary that will be used to separate the individual MIME parts.

Returns:

NULL on failure, or a pointer to a managed string containing the file attachment encoded as a MIME part on success.

Definition at line 1083 of file mime.c.

References base64_encode(), media_type_t::bin, log_pedantic, mail_mime_get_media_type(), media_type_t::name, qp_encode(), st_empty_out(), st_free(), and st_merge.

Referenced by portal_smtp_create_data().

int_t mail_mime_encoding (placer_t *header*)

Get the encoding type from a MIME header.

Note:

If no encoding type is specified in the header via Content-Transfer-Encoding, 7bit encoding is assumed.

Parameters:

header a placer containing the MIME header to be examined.

Returns:

the MIME encoding type specified by the header, or MESSAGE_ENCODING_UNKNOWN on failure.

Definition at line 575 of file mime.c.

References mail_header_fetch_cleaned(), MESSAGE_ENCODING_7BIT, MESSAGE_ENCODING_8BIT, MESSAGE_ENCODING_BASE64, MESSAGE_ENCODING_QUOTED_PRINTABLE, MESSAGE_ENCODING_UNKNOWN, mm_cmp_ci_eq(), PLACER, st_char_get(), st_free(), and st_length_get().

Referenced by mail_mime_part().

void mail_mime_free (mail_mime_t * *mime*)

Free a mail mime object and its underlying data, and recursively free its children parts.

Parameters:

mime a pointer to the mail mime object to be freed.

Returns:

This function returns no value.

Definition at line 889 of file mime.c.

References ar_field_ptr(), ar_free(), ar_length_get(), mail_mime_t::boundary, mail_mime_t::children, mail_mime_free(), mm_free(), and st_cleanup().

Referenced by mail_destroy(), mail_mime_free(), mail_mime_part(), and mail_mime_update().

stringer_t* mail_mime_generate_boundary (array_t * *parts*)

Generate a MIME boundary string that is unique to a collection of content.

Parameters:

parts a pointer to an array of managed strings containing the MIME children data to be separated by the boundary.

Returns:

NULL on failure, or a pointer to a managed string containing the generated boundary on success.

Definition at line 1016 of file mime.c.

References ar_field_ptr(), ar_length_get(), log_error, log_pedantic, rand_get_uint64(), st_alloc, st_char_get(), st_free(), st_length_set(), and st_search_ci().

Referenced by portal_smtp_create_data().

media_type_t* mail_mime_get_media_type (chr_t * *extension*)

Get the media type for a given file extension.

Note:

If no direct match is found for the content, "application/octet-stream" will be returned.

Parameters:

extension a pointer to a null-terminated string containing the file extension to be looked up, starting with a period.

Returns:

a pointer to a media type object corresponding to the media type of the specified file extension.

Definition at line 106 of file mime.c.

References mm_cmp_cs_eq(), and ns_length_get().

Referenced by mail_mime_encode_part().

stringer_t* mail_mime_get_smtp_envelope (stringer_t * *from*, inx_t * *tos*, inx_t * *ccs*, inx_t * *bccs*, stringer_t * *subject*, stringer_t * *boundary*, bool_t *attached*)

Get smtp envelope data for an outbound message sent by a webmail client.

Parameters:

from a pointer to a managed string containing the sender's address.

tos a pointer to an inx holder containing a collection of managed strings with the email addresses specified in the To: header.

ccs a pointer to an inx holder containing a collection of managed strings with the email addresses specified in the CC: header.

bccs a pointer to an inx holder containing a collection of managed strings with the email addresses specified in the BCC: header.

subject a pointer to a managed string containing the text of the subject line.

boundary a pointer to a managed string containing a boundary string to be used in multipart messages.

attached if true, the envelope is to be created for a mail with attachments (type "multipart/mixed"); if false, only a single part will be sent for the main email body.

Returns:

NULL on failure or a pointer to a managed string containing the smtp envelope data that will be supplied to an smtp relay server at the beginning of the DATA command.

Definition at line 1153 of file mime.c.

References log_error, log_pedantic, NULLER, portal_smtp_merge_headers(), st_cleanup(), st_free(), and st_merge.

Referenced by portal_smtp_create_data().

placer_t mail_mime_header (stringer_t * *part*)

Get a placer pointing to a mime header in a mime part.

Parameters:

part a managed string containing the mime part text to be parsed.

Returns:

a placer pointing to the mime header at the start of the specified mime part.

Definition at line 480 of file mime.c.

References length, pl_init(), st_char_get(), st_data_get(), and st_length_get().

Referenced by mail_mime_part().

mail_mime_t* mail_mime_part (stringer_t * *part*, uint32_t *recursion*)

Parse a block of data into a mail mime object.

Note:

By parsing the specified mime part, this function fills in the content type and encoding of the resulting mail mime object. If the message is multipart, the boundary string is determined and then used to split the body into children; then each child part is passed to **mail_mime_part()** to be parsed likewise, recursively.

Parameters:

part a managed string containing the mime part data to be parsed.

recursion an incremented recursion level tracker for calling this function, to prevent an overflow from occurring.

Returns:

NULL on failure or a pointer to a newly allocated and updated mail mime object parsed from the part data on success.

Definition at line 922 of file mime.c.

References ar_alloc(), ar_append(), ar_field_st(), ar_free(), ar_length_get(), ARRAY_TYPE_POINTER, mail_mime_t::body, mail_mime_t::boundary, mail_mime_t::children, mail_mime_t::encoding, mail_mime_t::entire, mail_mime_t::header, log_pedantic, mail_mime_boundary(), mail_mime_encoding(), mail_mime_free(), mail_mime_header(), mail_mime_part(), MAIL_MIME_RECURSION_LIMIT, mail_mime_split(), mail_mime_type(), MESSAGE_TYPE_MULTI_ALTERNATIVE, MESSAGE_TYPE_MULTI_MIXED, MESSAGE_TYPE_MULTI_RELATED, MESSAGE_TYPE_MULTI_RFC822, MESSAGE_TYPE_MULTI_UNKOWN, mm_alloc(), pl_init(), st_char_get(), st_data_get(), st_empty(), st_length_get(), and mail_mime_t::type.

Referenced by mail_mime_part(), and mail_mime_update().

array_t* mail_mime_split (placer_t *body*, stringer_t * *boundary*)

Split a mime body into an array of children by a boundary string.

Parameters:

body a placer containing the body text to be parsed.

boundary a pointer to a managed string containing the boundary string to split the mime content.

Returns:

NULL on failure, or a pointer to an array of mime children on success.

TODO: This is ugly. Because the array gets a placer pointer we need to free it when done. But that means differentiating between these placers and what were usually passed which will likely stack allocated placers. We could probably just change it to a stringer now that its going to **st_free()**, but that would mean lots of updates all over the place.

Definition at line 846 of file mime.c.

References ar_alloc(), ar_append(), ARRAY_TYPE_STRINGER, FOREIGNDATA, HEAP, JOINED, log_pedantic, mail_mime_child(), mail_mime_count(), pl_set(), placer_t, PLACER_T, st_alloc_opts(), st_empty(), and st_free().

Referenced by mail_mime_part().

int_t mail_mime_type (placer_t header)

Get the content type from a MIME header.

Note:

If no content type is specified in the header via Content-Type, "text/plain" is assumed.

Parameters:

header a placer containing the MIME header to be examined.

Returns:

the MIME content type specified by the header, or MESSAGE_TYPE_UNKNOWN on failure.

Definition at line 519 of file mime.c.

References mail_header_fetch_cleaned(), MESSAGE_TYPE_HTML,
MESSAGE_TYPE_MULTI_ALTERNATIVE, MESSAGE_TYPE_MULTI_MIXED,
MESSAGE_TYPE_MULTI_RELATED, MESSAGE_TYPE_MULTI_RFC822,
MESSAGE_TYPE_MULTI_UNKOWN, MESSAGE_TYPE_PLAIN, MESSAGE_TYPE_UNKNOWN,
mm_cmp_ci_eq(), PLACER, st_char_get(), st_free(), and st_length_get().

Referenced by mail_mime_part().

stringer_t* mail_mime_type_group (placer_t header)

Get the value of the Content-Type header from a mime header.

Parameters:

header a placer pointing to the mime header to be parsed.

Returns:

a managed string containing the content type value of the header, with "text" as the default.

Definition at line 126 of file mime.c.

References mail_header_fetch_cleaned(), PLACER, st_char_get(), st_free(), st_import(), and st_length_get().

Referenced by imap_fetch_bodystructure(), and portal_message_attachments().

array_t* mail_mime_type_parameters (placer_t header)

Get an array of the key/value pairs of parameters passed to the value of the mime Content-Type header.

Note:

The parameters of the Content-Type header value will be examined, and each found parameter will result in the addition of TWO managed strings to the returned array: the first containing the parameter key name, and the second with the parameter value.

Parameters:

header a placer pointing to the mime header to be parsed.

Returns:

NULL on failure, or on success, an array of managed strings structured as the key name followed by the value of each parameter passed in the Content-Type header.

Definition at line 432 of file mime.c.

References `ar_alloc()`, `ar_append()`, `ar_free()`, `ar_length_get()`, `ARRAY_TYPE_STRINGER`, `mail_header_fetch_cleaned()`, `mail_mime_type_parameters_key()`, `mail_mime_type_parameters_value()`, `PLACER`, `placer_t`, `st_free()`, `tok_get_count_st()`, `tok_get_st()`, and `upper_st()`.

Referenced by `imap_fetch_bodystructure()`.

`stringer_t* mail_mime_type_parameters_key (stringer_t * parameter)`

Get the key name of a mime header line parameter.

Parameters:

parameter a managed string containing the complete parameter (key/value pair) of the mime header line.

Returns:

NULL on failure or a managed string containing the mime header parameter key name on success.

Definition at line 326 of file `mime.c`.

References `length`, `st_char_get()`, `st_import()`, and `st_length_get()`.

Referenced by `mail_mime_type_parameters()`.

`stringer_t* mail_mime_type_parameters_value (stringer_t * parameter)`

Get the value of a mime header line parameter.

Parameters:

parameter a managed string containing the complete parameter (key/value pair) of the mime header line.

Returns:

NULL on failure or a managed string containing the mime header parameter value on success.

Definition at line 364 of file `mime.c`.

References `length`, `st_char_get()`, `st_import()`, and `st_length_get()`.

Referenced by `mail_mime_type_parameters()`.

`stringer_t* mail_mime_type_sub (placer_t header)`

Get the subtype of the Content-Type header value from a mime header.

Note:

For example in the case of a Content-Type of 'text/plain', "plain" would be the subtype.

Parameters:

header a placer pointing to the mime header to be parsed.

Returns:

a managed string containing the content subtype of the header, with "plain" as the default.

Definition at line 172 of file `mime.c`.

References `mail_header_fetch_cleaned()`, `PLACER`, `st_char_get()`, `st_free()`, `st_import()`, and `st_length_get()`.

Referenced by `imap_fetch_bodystructure()`, and `portal_message_attachments()`.

int_t mail_mime_update (mail_message_t * *message*)

Re-parse a mail message's data as a mime part, freeing any existing mime part(s) that may have already existed.

Parameters:

message the mail message object containing the message data to be parsed.

Returns:

This function always returns 1.

Definition at line 997 of file mime.c.

References mail_mime_free(), mail_mime_part(), mail_message_t::mime, pl_init(), placer_t, st_char_get(), st_length_get(), and mail_message_t::text.

Referenced by imap_fetch_message(), imap_fetch_return_message(), imap_fetch_return_mime(), imap_search_messages_body(), imap_search_messages_text(), and portal_endpoint_messages_load().

void mail_mod_subject (stringer_t ** *message*, chr_t * *label*)

Prepend text to a Subject header line inside of a mail message.

Note:

If no Subject line is found in the header of the message, one will be created and inserted with the specified label at the end of the header.

Parameters:

message a pointer to the address of a managed string that contains the mail message header or mail message body, and will receive the value of the resulting managed string that will be allocated to hold the modified contents.

label a null-terminated string that will be prepended to the value of the subject header value.

Returns:

This function returns no value.

Definition at line 391 of file headers.c.

References CONSTANT, log_pedantic, mail_header_end(), mail_header_pop(), pl_empty(), PLACER, placer_t, st_char_get(), st_cmp_ci_starts(), st_data_get(), st_free(), st_length_get(), and st_merge.

Referenced by mail_add_forward_headers(), mail_load_header(), mail_load_message(), and smtp_check_filters().

int_t mail_modify_part (server_t * *server*, mail_message_t * *message*, stringer_t * *part*, uint64_t *signum*, uint64_t *sigkey*, int_t *disposition*, int_t *recursion*)

Insert a spam signature training link into a specified part of a mime message.

Warning:

This function may fail to work as expected and still return a value of 1.

Note:

If the mime type is MESSAGE_TYPE_UNKNOWN (binary file), the function returns immediately. If the type is MESSAGE_TYPE_HTML or MESSAGE_TYPE_PLAIN, the training link is inserted normally. For MESSAGE_TYPE_MULTI_RELATED, MESSAGE_TYPE_MULTI_MIXED,

MESSAGE_TYPE_MULTI_UNKOWN the link is inserted in the first mime part. For MESSAGE_TYPE_MULTI_ALTERNATIVE, the link is inserted into each part, for up to 8 times.

Parameters:

server the server object of the web server where the teacher application is hosted.
message the mail message object of the message to be modified.
part a placer pointing to the specified part of the message to be modified.
signum the spam signature number referenced by the teacher url.
sigkey the spam signature key for client verification in the teacher app.
disposition if 0, the message disposition is "innocent"; otherwise, the disposition specifies "spam".
recursion an incremented recursion level tracker for calling this function, to prevent an overflow from occurring.

Returns:

0 on recursion failure, or 1 otherwise.

Definition at line 699 of file signatures.c.

References `length`, `log_pedantic`, `mail_discover_encoding()`, `mail_discover_type()`, `mail_get_boundary()`, `mail_get_chunk()`, `mail_insert_chunk_base64()`, `mail_insert_chunk_text()`, `mail_modify_part()`, `MAIL_SIGNATURES_RECURSION_LIMIT`, `MESSAGE_ENCODING_BASE64`, `MESSAGE_TYPE_HTML`, `MESSAGE_TYPE_MULTI_ALTERNATIVE`, `MESSAGE_TYPE_MULTI_MIXED`, `MESSAGE_TYPE_MULTI_RELATED`, `MESSAGE_TYPE_MULTI_UNKOWN`, `MESSAGE_TYPE_PLAIN`, `MESSAGE_TYPE_UNKNOWN`, `PLACER`, `st_char_get()`, `st_data_get()`, `st_free()`, `st_length_get()`, `mail_message_t::text`, and `type()`.

Referenced by `mail_modify_part()`, and `mail_signature_add()`.

int_t mail_move_message (uint64_t usernum, uint64_t messagenum, uint64_t source, uint64_t target)

Move a message to a new folder in the database.

Parameters:

usenum the numerical id of the user that owns the message.
messagenum the numerical id of the message to be moved.
source the numerical id of the current parent folder of the specified message.
target the numerical id of the folder to which the specified message will be moved.

Returns:

-1 on error, 0 if the message wasn't found, or 1 on success.

Definition at line 307 of file store_message.c.

References `log_error`, `log_pedantic`, `mail_db_update_message_folder()`, `tran_commit()`, `tran_rollback()`, and `tran_start()`.

Referenced by `meta_messages_mover()`.

int_t mail_path_finder (chr_t * string)

Note:

This function is not called from anywhere in the code and may be subject to removal.

Definition at line 18 of file paths.c.

References `length`, and `ns_length_get()`.

**bool_t mail_remove_message (uint64_t *userid*, uint64_t *messageid*, uint32_t *size*,
chr_t * *server*)**

remove_message.c

remove_message.c

LOW: When the reliable job queue is working we can handle unlink errors by creating a job entry which will retry the unlink operation at a later time.

Parameters:

userid the user id to whom the specified mail message belongs.

messageid the target mail message id.

size the size of the message in bytes, to be assessed against the user quota.

server the name of the server on which the mail message resides.

Returns:

true if the message removal succeeds or false on failure.

Definition at line 25 of file remove_message.c.

References log_pedantic, mail_db_delete_message(), mail_message_path(), ns_free(), tran_commit(), tran_rollback(), and tran_start().

Referenced by imap_folder_remove(), imap_message_expunge(), pop_session_destroy(), portal_endpoint_messages_remove(), and smtp_rollout().

void mail_setup_basic (basic_message_t * *message*, stringer_t * *text*)

Note:

This function is not currently referenced by any other code.

Definition at line 34 of file objects.c.

References basic_message_t::date, basic_message_t::from, length, mail_header_end(), mail_store_header(), mm_cmp_ci_eq(), st_char_get(), basic_message_t::subject, basic_message_t::text, and basic_message_t::to.

**void mail_signature_add (mail_message_t * *message*, server_t * *server*, uint64_t *signature*,
uint64_t *sigkey*, int_t *disposition*)**

Insert a spam signature training link into a mail message.

See also:

mail_modify_part()

Parameters:

message the mail message object of the message to be modified.

server the server object of the web server where the teacher application is hosted.

signature the spam signature number referenced by the teacher url.

sigkey the spam signature key for client verification in the teacher app.

disposition if 0, the message disposition is "innocent"; otherwise, the disposition specifies "spam".

Returns:

This function returns no value.

Definition at line 824 of file signatures.c.

References log_pedantic, mail_modify_part(), PLACER, st_char_get(), st_length_get(), st_length_int(), and mail_message_t::text.

Referenced by mail_add_forward_headers(), and mail_load_message().

placer_t mail_store_header (chr_t * *stream*, size_t *length*)

Return a placer pointing to the trimmed value of a mail header.

Note:

This function merely marks a string residing between leading whitespace and a trailing or

.

Parameters:

stream a pointer to a buffer containing the start of the mail header data.

length the length, in bytes, of the mail header field. a placer containing the trimmed value of the mail header line.

Definition at line 298 of file headers.c.

References pl_init().

Referenced by mail_headers(), mail_message(), and mail_setup_basic().

uint64_t mail_store_message (uint64_t *usernum*, stringer_t * *pubkey*, uint64_t *foldernum*, uint32_t * *status*, uint64_t *signum*, uint64_t *sigkey*, stringer_t * *message*)

Store a mail message, with its meta-information in the database, and the contents persisted to disk.

Note:

The stored message is always compressed, but only encrypted if the user's public key is supplied.

Parameters:

usernum the numerical id of the user to which the message belongs.

pubkey if not NULL, a public key that will be used to encrypt the message for the intended user.

foldernum the folder # that will contain the message.

status a pointer to the status flags value for the message, which will be updated if the message is to be encrypted.

signum the spam signature for the message.

sigkey the spam key for the message.

message a managed string containing the raw body of the message.

Returns:

0 on failure, or the newly inserted id of the message in the database on success.

Definition at line 111 of file store_message.c.

References compress_free(), compress_lzo(), compress_total_length(), cryptex_free(), cryptex_total_length(), ecies_encrypt(), ECIES_PUBLIC_BINARY, FMESSAGE_OPT_COMPRESSED, FMESSAGE_OPT_ENCRYPTED, log_error, log_pedantic, mail_db_insert_message(), MAIL_STATUS_ENCRYPTED, mail_store_message_data(), ns_free(), st_length_get(), st_length_int(), tran_commit(), tran_rollback(), and tran_start().

Referenced by imap_append_message(), and smtp_store_message().

magma/objects/mail/mime.c File Reference

Functions used to parse and manipulate MIME messages.

```
#include "magma.h"
```

Functions

- **media_type_t * mail_mime_get_media_type** (chr_t *extension)
- *Get the media type for a given file extension.* **stringer_t * mail_mime_type_group** (placer_t header)
- *Get the value of the Content-Type header from a mime header.* **stringer_t * mail_mime_type_sub** (placer_t header)
- *Get the subtype of the Content-Type header value from a mime header.* **stringer_t * mail_mime_content_encoding** (placer_t header)
- *Get the content encoding type from a mime header.* **stringer_t * mail_mime_content_id** (placer_t header)
- *Get the content id from a mime header.* **stringer_t * mail_mime_type_parameters_key** (stringer_t *parameter)
- *Get the key name of a mime header line parameter.* **stringer_t * mail_mime_type_parameters_value** (stringer_t *parameter)
- *Get the value of a mime header line parameter.* **array_t * mail_mime_type_parameters** (placer_t header)
- *Get an array of the key/value pairs of parameters passed to the value of the mime Content-Type header.* **placer_t mail_mime_header** (stringer_t *part)
- *Get a placer pointing to a mime header in a mime part.* **int_t mail_mime_type** (placer_t header)
- *Get the content type from a MIME header.* **int_t mail_mime_encoding** (placer_t header)
- *Get the encoding type from a MIME header.* **stringer_t * mail_mime_boundary** (placer_t header)
- *Get the boundary from a MIME header.* **uint32_t mail_mime_count** (placer_t body, stringer_t *boundary)
- *Count the number of instances of a boundary string inside a MIME body.* **placer_t mail_mime_child** (placer_t body, stringer_t *boundary, uint32_t child)
- *Get a placer pointing to the specified child inside a MIME body.* **array_t * mail_mime_split** (placer_t body, stringer_t *boundary)
- *Split a mime body into an array of children by a boundary string.* **void mail_mime_free** (mail_mime_t *mime)
- *Free a mail mime object and its underlying data, and recursively free its children parts.* **mail_mime_t * mail_mime_part** (stringer_t *part, uint32_t recursion)
- *Parse a block of data into a mail mime object.* **int_t mail_mime_update** (mail_message_t *message)
- *Re-parse a mail message's data as a mime part, freeing any existing mime part(s) that may have already existed.* **stringer_t * mail_mime_generate_boundary** (array_t *parts)
- *Generate a MIME boundary string that is unique to a collection of content.* **stringer_t * mail_mime_encode_part** (stringer_t *data, stringer_t *filename, stringer_t *boundary)
- *Encode a MIME part for a provided block of data (file attachment) with the specified filename.* **stringer_t * mail_mime_get_smtp_envelope** (stringer_t *from, inx_t *tos, inx_t *ccs, inx_t *bccs, stringer_t *subject, stringer_t *boundary, bool_t attached)

Get smtp envelope data for an outbound message sent by a webmail client.

Variables

- **media_type_t media_types** []

Detailed Description

Functions used to parse and manipulate MIME messages.

Definition in file **mime.c**.

Function Documentation

stringer_t* mail_mime_boundary (placer_t *header*)

Get the boundary from a MIME header.

mime.c

Note:

This function first scans the value of Content-Type for the boundary, and then the rest of the MIME header.

Parameters:

header a placer pointing to the MIME header to be parsed.

Returns:

NULL on failure, or a pointer to a managed string containing the MIME boundary string on success.

Definition at line 622 of file mime.c.

References `content`, `length`, `log_error`, `log_pedantic`, `mail_header_fetch_all()`, `PLACER`, `st_char_get()`, `st_cleanup()`, `st_length_get()`, `st_merge`, and `st_search_ci()`.

Referenced by `mail_mime_part()`.

placer_t mail_mime_child (placer_t *body*, stringer_t * *boundary*, uint32_t *child*)

Get a placer pointing to the specified child inside a MIME body.

Parameters:

body a placer containing the body text to be parsed.

boundary a pointer to a managed string containing the boundary string to split the MIME content.

child the zero-based index of the MIME child to be located in the body text.

Returns:

`pl_null()` on failure, or a placer containing the specified MIME child on success.

Definition at line 754 of file mime.c.

References `length`, `log_pedantic`, `mm_cmp_cs_eq()`, `pl_empty()`, `pl_init()`, `pl_null()`, `st_char_get()`, `st_empty()`, and `st_length_get()`.

Referenced by `mail_mime_split()`.

stringer_t* mail_mime_content_encoding (placer_t *header*)

Get the content encoding type from a mime header.

Note:

This function parses the Content-Transfer-Encoding field of the header, returning "7bit" by default.

Parameters:

header a placer pointing to the mime header to be parsed.

Returns:

a managed string containing the content encoding type value of the header, with "7bit" as default.

Definition at line 236 of file mime.c.

References mail_header_fetch_cleaned(), PLACER, st_char_get(), st_free(), st_import(), and st_length_get().

Referenced by imap_fetch_bodystructure().

stringer_t* mail_mime_content_id (placer_t *header*)

Get the content id from a mime header.

Note:

This function parses the Content-Id field of the header, returning "7bit" by default.

Parameters:

header a placer pointing to the mime header to be parsed.

Returns:

NULL on failure, or a managed string containing the content id value of the header.

Definition at line 282 of file mime.c.

References mail_header_fetch_cleaned(), PLACER, st_char_get(), st_free(), st_import(), and st_length_get().

Referenced by imap_fetch_bodystructure().

uint32_t mail_mime_count (placer_t *body*, stringer_t * *boundary*)

Count the number of instances of a boundary string inside a MIME body.

Note:

The search is terminated if "--" is found right after the boundary string.

Parameters:

body a placer containing the body text to be parsed.

boundary a pointer to a managed string containing the boundary string for the MIME content.

Returns:

0 on failure, or the number of times the boundary string was located in the MIME body on success.

Definition at line 699 of file mime.c.

References length, log_pedantic, mm_cmp_cs_eq(), pl_empty(), st_char_get(), st_empty(), and st_length_get().

Referenced by mail_mime_split().

stringer_t* mail_mime_encode_part (stringer_t * *data*, stringer_t * *filename*, stringer_t * *boundary*)

Encode a MIME part for a provided block of data (file attachment) with the specified filename.

Note:

This function will look up the media type based on the supplied filename, and use that media type as a determination of whether the content is to be encoded as either quoted-printable or base64.

Parameters:

data a pointer to a managed string containing the body of the data to be encoded.

filename a pointer to a managed string containing the filename of the attachment for which the data was provided.
boundary a pointer to a managed string containing the boundary that will be used to separate the individual MIME parts.

Returns:

NULL on failure, or a pointer to a managed string containing the file attachment encoded as a MIME part on success.

Definition at line 1083 of file mime.c.

References `base64_encode()`, `media_type_t::bin`, `log_pedantic`, `mail_mime_get_media_type()`, `media_type_t::name`, `qp_encode()`, `st_empty_out()`, `st_free()`, and `st_merge`.

Referenced by `portal_smtp_create_data()`.

int_t mail_mime_encoding (placer_t header)

Get the encoding type from a MIME header.

Note:

If no encoding type is specified in the header via Content-Transfer-Encoding, 7bit encoding is assumed.

Parameters:

header a placer containing the MIME header to be examined.

Returns:

the MIME encoding type specified by the header, or MESSAGE_ENCODING_UNKNOWN on failure.

Definition at line 575 of file mime.c.

References `mail_header_fetch_cleaned()`, `MESSAGE_ENCODING_7BIT`, `MESSAGE_ENCODING_8BIT`, `MESSAGE_ENCODING_BASE64`, `MESSAGE_ENCODING_QUOTED_PRINTABLE`, `MESSAGE_ENCODING_UNKNOWN`, `mm_cmp_ci_eq()`, `PLACER`, `st_char_get()`, `st_free()`, and `st_length_get()`.

Referenced by `mail_mime_part()`.

void mail_mime_free (mail_mime_t * mime)

Free a mail mime object and its underlying data, and recursively free its children parts.

Parameters:

mime a pointer to the mail mime object to be freed.

Returns:

This function returns no value.

Definition at line 889 of file mime.c.

References `ar_field_ptr()`, `ar_free()`, `ar_length_get()`, `mail_mime_t::boundary`, `mail_mime_t::children`, `mail_mime_free()`, `mm_free()`, and `st_cleanup()`.

Referenced by `mail_destroy()`, `mail_mime_free()`, `mail_mime_part()`, and `mail_mime_update()`.

stringer_t* mail_mime_generate_boundary (array_t * parts)

Generate a MIME boundary string that is unique to a collection of content.

Parameters:

parts a pointer to an array of managed strings containing the MIME children data to be separated by the boundary.

Returns:

NULL on failure, or a pointer to a managed string containing the generated boundary on success.

Definition at line 1016 of file mime.c.

References `ar_field_ptr()`, `ar_length_get()`, `log_error`, `log_pedantic`, `rand_get_uint64()`, `st_alloc`, `st_char_get()`, `st_free()`, `st_length_set()`, and `st_search_ci()`.

Referenced by `portal_smtp_create_data()`.

media_type_t* mail_mime_get_media_type (chr_t * *extension*)

Get the media type for a given file extension.

Note:

If no direct match is found for the content, "application/octet-stream" will be returned.

Parameters:

extension a pointer to a null-terminated string containing the file extension to be looked up, starting with a period.

Returns:

a pointer to a media type object corresponding to the media type of the specified file extension.

Definition at line 106 of file mime.c.

References `mm_cmp_cs_eq()`, and `ns_length_get()`.

Referenced by `mail_mime_encode_part()`.

stringer_t* mail_mime_get_smtp_envelope (stringer_t * *from*, inx_t * *tos*, inx_t * *ccs*, inx_t * *bccs*, stringer_t * *subject*, stringer_t * *boundary*, bool_t *attached*)

Get smtp envelope data for an outbound message sent by a webmail client.

Parameters:

from a pointer to a managed string containing the sender's address.

tos a pointer to an inx holder containing a collection of managed strings with the email addresses specified in the To: header.

ccs a pointer to an inx holder containing a collection of managed strings with the email addresses specified in the CC: header.

bccs a pointer to an inx holder containing a collection of managed strings with the email addresses specified in the BCC: header.

subject a pointer to a managed string containing the text of the subject line.

boundary a pointer to a managed string containing a boundary string to be used in multipart messages.

attached if true, the envelope is to be created for a mail with attachments (type "multipart/mixed"); if false, only a single part will be sent for the main email body.

Returns:

NULL on failure or a pointer to a managed string containing the smtp envelope data that will be supplied to an smtp relay server at the beginning of the DATA command.

Definition at line 1153 of file mime.c.

References `log_error`, `log_pedantic`, `NULLER`, `portal_smtp_merge_headers()`, `st_cleanup()`, `st_free()`, and `st_merge`.

Referenced by `portal_smtp_create_data()`.

placer_t mail_mime_header (stringer_t * *part*)

Get a placer pointing to a mime header in a mime part.

Parameters:

part a managed string containing the mime part text to be parsed.

Returns:

a placer pointing to the mime header at the start of the specified mime part.

Definition at line 480 of file mime.c.

References `length`, `pl_init()`, `st_char_get()`, `st_data_get()`, and `st_length_get()`.

Referenced by `mail_mime_part()`.

mail_mime_t* mail_mime_part (stringer_t * *part*, uint32_t *recursion*)

Parse a block of data into a mail mime object.

Note:

By parsing the specified mime part, this function fills in the content type and encoding of the resulting mail mime object. If the message is multipart, the boundary string is determined and then used to split the body into children; then each child part is passed to **mail_mime_part()** to be parsed likewise, recursively.

Parameters:

part a managed string containing the mime part data to be parsed.

recursion an incremented recursion level tracker for calling this function, to prevent an overflow from occurring.

Returns:

NULL on failure or a pointer to a newly allocated and updated mail mime object parsed from the part data on success.

Definition at line 922 of file mime.c.

References `ar_alloc()`, `ar_append()`, `ar_field_st()`, `ar_free()`, `ar_length_get()`, `ARRAY_TYPE_POINTER`, `mail_mime_t::body`, `mail_mime_t::boundary`, `mail_mime_t::children`, `mail_mime_t::encoding`, `mail_mime_t::entire`, `mail_mime_t::header`, `log_pedantic`, `mail_mime_boundary()`, `mail_mime_encoding()`, `mail_mime_free()`, `mail_mime_header()`, `mail_mime_part()`, `MAIL_MIME_RECURSION_LIMIT`, `mail_mime_split()`, `mail_mime_type()`, `MESSAGE_TYPE_MULTI_ALTERNATIVE`, `MESSAGE_TYPE_MULTI_MIXED`, `MESSAGE_TYPE_MULTI_RELATED`, `MESSAGE_TYPE_MULTI_RFC822`, `MESSAGE_TYPE_MULTI_UNKOWN`, `mm_alloc()`, `pl_init()`, `st_char_get()`, `st_data_get()`, `st_empty()`, `st_length_get()`, and `mail_mime_t::type`.

Referenced by `mail_mime_part()`, and `mail_mime_update()`.

array_t* mail_mime_split (placer_t *body*, stringer_t * *boundary*)

Split a mime body into an array of children by a boundary string.

Parameters:

body a placer containing the body text to be parsed.

boundary a pointer to a managed string containing the boundary string to split the mime content.

Returns:

NULL on failure, or a pointer to an array of mime children on success.

TODO: This is ugly. Because the array gets a placer pointer we need to free it when done. But that means differentiating between these placers and what were usually passed which will likely stack allocated placers. We could probably just change it to a stringer now that its going to **st_free()**, but that would mean lots of updates all over the place.

Definition at line 846 of file mime.c.

References `ar_alloc()`, `ar_append()`, `ARRAY_TYPE_STRINGER`, `FOREIGNDATA`, `HEAP`, `JOINTED`, `log_pedantic`, `mail_mime_child()`, `mail_mime_count()`, `pl_set()`, `placer_t`, `PLACER_T`, `st_alloc_opts()`, `st_empty()`, and `st_free()`.

Referenced by `mail_mime_part()`.

int_t mail_mime_type (placer_t header)

Get the content type from a MIME header.

Note:

If no content type is specified in the header via Content-Type, "text/plain" is assumed.

Parameters:

header a placer containing the MIME header to be examined.

Returns:

the MIME content type specified by the header, or `MESSAGE_TYPE_UNKNOWN` on failure.

Definition at line 519 of file mime.c.

References `mail_header_fetch_cleaned()`, `MESSAGE_TYPE_HTML`, `MESSAGE_TYPE_MULTI_ALTERNATIVE`, `MESSAGE_TYPE_MULTI_MIXED`, `MESSAGE_TYPE_MULTI_RELATED`, `MESSAGE_TYPE_MULTI_RFC822`, `MESSAGE_TYPE_MULTI_UNKOWN`, `MESSAGE_TYPE_PLAIN`, `MESSAGE_TYPE_UNKNOWN`, `mm_cmp_ci_eq()`, `PLACER`, `st_char_get()`, `st_free()`, and `st_length_get()`.

Referenced by `mail_mime_part()`.

stringer_t* mail_mime_type_group (placer_t header)

Get the value of the Content-Type header from a mime header.

Parameters:

header a placer pointing to the mime header to be parsed.

Returns:

a managed string containing the content type value of the header, with "text" as the default.

Definition at line 126 of file mime.c.

References `mail_header_fetch_cleaned()`, `PLACER`, `st_char_get()`, `st_free()`, `st_import()`, and `st_length_get()`.

Referenced by `imap_fetch_bodystructure()`, and `portal_message_attachments()`.

array_t* mail_mime_type_parameters (placer_t *header*)

Get an array of the key/value pairs of parameters passed to the value of the mime Content-Type header.

Note:

The parameters of the Content-Type header value will be examined, and each found parameter will result in the addition of TWO managed strings to the returned array: the first containing the parameter key name, and the second with the parameter value.

Parameters:

header a placer pointing to the mime header to be parsed.

Returns:

NULL on failure, or on success, an array of managed strings structured as the key name followed by the value of each parameter passed in the Content-Type header.

Definition at line 432 of file mime.c.

References `ar_alloc()`, `ar_append()`, `ar_free()`, `ar_length_get()`, `ARRAY_TYPE_STRINGER`, `mail_header_fetch_cleaned()`, `mail_mime_type_parameters_key()`, `mail_mime_type_parameters_value()`, `PLACER`, `placer_t`, `st_free()`, `tok_get_count_st()`, `tok_get_st()`, and `upper_st()`.

Referenced by `imap_fetch_bodystructure()`.

stringer_t* mail_mime_type_parameters_key (stringer_t * *parameter*)

Get the key name of a mime header line parameter.

Parameters:

parameter a managed string containing the complete parameter (key/value pair) of the mime header line.

Returns:

NULL on failure or a managed string containing the mime header parameter key name on success.

Definition at line 326 of file mime.c.

References `length`, `st_char_get()`, `st_import()`, and `st_length_get()`.

Referenced by `mail_mime_type_parameters()`.

stringer_t* mail_mime_type_parameters_value (stringer_t * *parameter*)

Get the value of a mime header line parameter.

Parameters:

parameter a managed string containing the complete parameter (key/value pair) of the mime header line.

Returns:

NULL on failure or a managed string containing the mime header parameter value on success.

Definition at line 364 of file mime.c.

References `length`, `st_char_get()`, `st_import()`, and `st_length_get()`.

Referenced by `mail_mime_type_parameters()`.

stringer_t* mail_mime_type_sub (placer_t header)

Get the subtype of the Content-Type header value from a mime header.

Note:

For example in the case of a Content-Type of 'text/plain', "plain" would be the subtype.

Parameters:

header a placer pointing to the mime header to be parsed.

Returns:

a managed string containing the content subtype of the header, with "plain" as the default.

Definition at line 172 of file mime.c.

References mail_header_fetch_cleaned(), PLACER, st_char_get(), st_free(), st_import(), and st_length_get().

Referenced by imap_fetch_bodystructure(), and portal_message_attachments().

int_t mail_mime_update (mail_message_t * message)

Re-parse a mail message's data as a mime part, freeing any existing mime part(s) that may have already existed.

Parameters:

message the mail message object containing the message data to be parsed.

Returns:

This function always returns 1.

Definition at line 997 of file mime.c.

References mail_mime_free(), mail_mime_part(), mail_message_t::mime, pl_init(), placer_t, st_char_get(), st_length_get(), and mail_message_t::text.

Referenced by imap_fetch_message(), imap_fetch_return_message(), imap_fetch_return_mime(), imap_search_messages_body(), imap_search_messages_text(), and portal_endpoint_messages_load().

Variable Documentation

media_type_t media_types[]

Definition at line 15 of file mime.c.

magma/objects/mail/objects.c File Reference

Functions used to interface with and manage message data.

```
#include "magma.h"
```

Functions

- void **mail_destroy_message** (smtp_message_t *message)
- *Destroy an smtp message and free all of its underlying data.* void **mail_setup_basic** (basic_message_t *message, stringer_t *text)
- void **mail_destroy** (mail_message_t *message)
- *Free a mail message and all its underlying data.* mail_message_t * **mail_message** (stringer_t *text)
- *Parse a raw mail data string and return a new mail message object containing the processed message.* smtp_message_t * **mail_create_message** (stringer_t *text)

Create an smtp message object out of a buffer of raw data supplied via the smtp DATA command.

Detailed Description

Functions used to interface with and manage message data.

Definition in file **objects.c**.

Function Documentation

smtp_message_t* mail_create_message (stringer_t * text)

Create an smtp message object out of a buffer of raw data supplied via the smtp DATA command.

Note:

This function will also generate a random message ID.

Parameters:

text a pointer to a managed string containing the smtp data to be parsed.

Returns:

NULL on failure or a pointer to the newly initialized smtp message object wrapping the data on success.

Definition at line 177 of file objects.c.

References smtp_message_t::header_length, smtp_message_t::id, log_pedantic, mail_header_end(), mail_headers(), mm_alloc(), mm_free(), rand_choices(), st_free(), and smtp_message_t::text.

Referenced by smtp_data().

void mail_destroy (mail_message_t * message)

Free a mail message and all its underlying data.

Parameters:

message a pointer to the mail message object to be freed.

Returns:

This function returns no value.

Definition at line 85 of file objects.c.

References mail_mime_free(), mail_message_t::mime, mm_free(), st_cleanup(), and mail_message_t::text.

Referenced by imap_fetch_body(), imap_fetch_message(), imap_fetch_return_header(), imap_fetch_return_message(), imap_fetch_return_mime(), imap_fetch_return_text(), imap_search_messages(), mail_add_forward_headers(), mail_load_header(), pop_retr(), pop_top(), and portal_endpoint_messages_load().

void mail_destroy_message (smtp_message_t * *message*)

Destroy an smtp message and free all of its underlying data.

Parameters:

message a pointer to the smtp message to be destroyed.

Returns:

This function returns no value.

Definition at line 20 of file objects.c.

References smtp_message_t::id, mm_free(), st_cleanup(), and smtp_message_t::text.

Referenced by smtp_data(), smtp_session_destroy(), and smtp_session_reset().

mail_message_t* mail_message (stringer_t * *text*)

Parse a raw mail data string and return a new mail message object containing the processed message.
objects.c

Parameters:

text a pointer to the managed string containing the raw (uncompressed) mail data to be parsed.

Returns:

NULL on failure or a pointer to a newly allocated mail message object with the message on success.

Definition at line 105 of file objects.c.

References mail_message_t::date, mail_message_t::from, mail_message_t::header_length, length, log_pedantic, mail_header_end(), mail_store_header(), mm_alloc(), mm_cmp_ci_eq(), mm_free(), st_char_get(), st_length_get(), mail_message_t::subject, mail_message_t::text, and mail_message_t::to.

Referenced by mail_add_forward_headers(), and mail_load_message().

void mail_setup_basic (basic_message_t * *message*, stringer_t * *text*)**Note:**

This function is not currently referenced by any other code.

Definition at line 34 of file objects.c.

References basic_message_t::date, basic_message_t::from, length, mail_header_end(), mail_store_header(), mm_cmp_ci_eq(), st_char_get(), basic_message_t::subject, basic_message_t::text, and basic_message_t::to.

magma/objects/objects.c File Reference

Functions used for managing objects.

```
#include "magma.h"
```

Functions

- **bool_t obj_cache_start** (void)
- *Initialize the object cache for all active user objects and web sessions.* void **obj_cache_stop** (void)
- *Stop the object cache and free all active user and web session objects.* void **obj_cache_prune** (void)

Variables

- **object_cache_t** objects

Detailed Description

Functions used for managing objects.

Definition in file **objects.c**.

Function Documentation

void obj_cache_prune (void)

Definition at line 59 of file objects.c.

References `count`, `inx_count()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_key_active()`, `inx_cursor_reset()`, `inx_cursor_value_next()`, `inx_delete()`, `inx_lock_read()`, `inx_lock_write()`, `inx_unlock()`, `meta_user_ref_stamp()`, `meta_user_ref_total()`, `sess_ref_stamp()`, `sess_ref_total()`, `object_cache_t::sessions`, `stats_adjust_by_name()`, `stats_set_by_name()`, and `object_cache_t::users`.

Referenced by `process_maint()`.

bool_t obj_cache_start (void)

Initialize the object cache for all active user objects and web sessions.

objects.c

Returns:

true on success or false on failure.

Definition at line 24 of file objects.c.

References `inx_alloc()`, `log_critical`, `M_INX_HASHED`, `M_INX_LOCK_MANUAL`, `meta_user_destroy()`, `sess_destroy()`, `object_cache_t::sessions`, and `object_cache_t::users`.

Referenced by `process_start()`.

void obj_cache_stop (void)

Stop the object cache and free all active user and web session objects.

Returns:

This function returns no value.

Definition at line 43 of file objects.c.

References `inx_free()`, `object_cache_t::sessions`, and `object_cache_t::users`.

Referenced by `process_stop()`.

Variable Documentation

object_cache_t objects

```
Initial value: {  
    .users = NULL,  
    .sessions = NULL  
}
```

Definition at line 15 of file objects.c.

Referenced by `meta_get()`, `meta_remove()`, `meta_user_prune()`, `sess_create()`, and `sess_get()`.

magma/objects/mail/parsing.c File Reference

Functions used to parse mail messages and extract information from them.

#include "magma.h"

Functions

- **placer_t * mail_domain_get** (stringer_t *address, placer_t *output)
- *Get the domain portion of an email address.* **stringer_t * mail_extract_address** (stringer_t *address)

Extract a valid email address from a From: header value.

Detailed Description

Functions used to parse mail messages and extract information from them.

Definition in file **parsing.c**.

Function Documentation

placer_t* mail_domain_get (stringer_t * *address*, placer_t * *output*)

Get the domain portion of an email address.

Parameters:

address a managed string containing the email address to be parsed.

output a pointer to a placer to receive the domain portion of the email address.

Returns:

NULL on failure or a pointer to the output of the domain extraction on success.

Definition at line 21 of file parsing.c.

References tok_get_count_st(), and tok_get_st().

Referenced by smtp_check_authorized_from(), and spf_check().

stringer_t* mail_extract_address (stringer_t * *address*)

Extract a valid email address from a From: header value.

parsing.c

Note:

The preference of this function is first to try to find the email addressed enclosed in <>. No valid email address may be enclosed in () or "".

Parameters:

address a managed string containing the buffer from which the email address will be extracted.

Returns:

NULL on failure or a managed string containing the email address on success.

Definition at line 37 of file parsing.c.

References `comment`, `length`, `log_pedantic`, `st_alloc`, `st_char_get()`, `st_data_get()`, `st_empty_out()`, and `st_length_set()`.

Referenced by `smtp_data_outbound()`.

magma/objects/mail/paths.c File Reference

Functions for mapping mail messages to their persistent file storage paths.

```
#include "magma.h"
```

Functions

- **int_t mail_path_finder** (chr_t *string)
- **chr_t * mail_message_path** (uint64_t number, chr_t *server)
- *Return the fully qualified local file path of a stored mail message for a specified message number and server.*
- **bool_t mail_create_directory** (uint64_t number, chr_t *server)

Create the on-disk directory structure necessary to hold a given message's file data.

Detailed Description

Functions for mapping mail messages to their persistent file storage paths.

Definition in file **paths.c**.

Function Documentation

bool_t mail_create_directory (uint64_t *number*, chr_t * *server*)

Create the on-disk directory structure necessary to hold a given message's file data.

Parameters:

number the mail message id.

server the hostname of the server where the message data resides or if NULL, the default server.

Returns:

true on success or false on failure.

Definition at line 71 of file paths.c.

References magma_t::active, log_error, magma, magma_t::root, st_char_get(), st_length_int(), and magma_t::storage.

Referenced by mail_copy_message(), and mail_store_message_data().

chr_t* mail_message_path (uint64_t *number*, chr_t * *server*)

Return the fully qualified local file path of a stored mail message for a specified message number and server.

paths.c

Parameters:

number the mail message id.

server the hostname of the server where the message data resides or if NULL, the default server.

Returns:

NULL on failure, or a pointer to a null-terminated string containing the absolute file path of the specified message.

Definition at line 40 of file paths.c.

References magma_t::active, log_pedantic, magma, ns_alloc(), ns_free(), magma_t::root, st_char_get(), st_length_int(), and magma_t::storage.

Referenced by adjust_message_encryption(), mail_copy_message(), mail_load_header(), mail_load_message(), mail_remove_message(), and mail_store_message_data().

int_t mail_path_finder (chr_t * *string*)

Note:

This function is not called from anywhere in the code and may be subject to removal.

Definition at line 18 of file paths.c.

References length, and ns_length_get().

magma/objects/mail/remove_message.c File Reference

Functions used to delete messages.

```
#include "magma.h"
```

Functions

- **bool_t mail_remove_message** (uint64_t usernum, uint64_t messagenum, uint32_t size, **chr_t** *server)
Remove a specified mail message from both the database and storage.
-

Detailed Description

Functions used to delete messages.

Definition in file **remove_message.c**.

Function Documentation

bool_t mail_remove_message (uint64_t *usernum*, uint64_t *messagenum*, uint32_t *size*,
chr_t * *server*)

Remove a specified mail message from both the database and storage.

remove_message.c

LOW: When the reliable job queue is working we can handle unlink errors by creating a job entry which will retry the unlink operation at a later time.

Parameters:

usernum the user id to whom the specified mail message belongs.

messagenum the target mail message id.

size the size of the message in bytes, to be assessed against the user quota.

server the name of the server on which the mail message resides.

Returns:

true if the message removal succeeds or false on failure.

Definition at line 25 of file remove_message.c.

References `log_pedantic`, `mail_db_delete_message()`, `mail_message_path()`, `ns_free()`, `tran_commit()`, `tran_rollback()`, and `tran_start()`.

Referenced by `imap_folder_remove()`, `imap_message_expunge()`, `pop_session_destroy()`, `portal_endpoint_messages_remove()`, and `smtp_rollout()`.

magma/objects/mail/signatures.c File Reference

Functions used to insert signatures into mail messages.

```
#include "magma.h"
```

Functions

- **stringer_t * mail_build_signature** (**server_t** *server, **int_t** content_type, **int_t** content_encoding, **uint64_t** signum, **uint64_t** sigkey, **int_t** disposition)
- *Build a spam text signature for insertion into a message body.* **int_t mail_discover_type** (**stringer_t** *header)
- *Get the value of the Content-Type header in a mail message header.* **int_t mail_discover_encoding** (**stringer_t** *header)
- *Get the value of the Content-Transfer-Encoding header in a mail message header.* **stringer_t * mail_extract_tag** (**chr_t** *stream, **size_t** length)
- *Extract an html tag from a data buffer, searching backwards.* **size_t mail_discover_insertion_point** (**stringer_t** *message, **stringer_t** *part, **int_t** type)
- *Find the position in a mail message where a custom message can be inserted.* **stringer_t * mail_insert_chunk_base64** (**server_t** *server, **stringer_t** *message, **stringer_t** *part, **uint64_t** signum, **uint64_t** sigkey, **int_t** disposition, **int_t** type, **int_t** encoding)
- *Insert a spam signature training link into a base64-encoded message part.* **stringer_t * mail_insert_chunk_text** (**server_t** *server, **stringer_t** *message, **stringer_t** *part, **uint64_t** signum, **uint64_t** sigkey, **int_t** disposition, **int_t** type, **int_t** encoding)
- *Insert a spam signature training link into a plain text message part.* **stringer_t * mail_get_chunk** (**stringer_t** *message, **stringer_t** *boundary, **int_t** chunk)
- *Get a specified chunk (mime part) of a multipart mime message.* **stringer_t * mail_get_boundary** (**stringer_t** *header)
- *Get the boundary string from a message header.* **int_t mail_modify_part** (**server_t** *server, **mail_message_t** *message, **stringer_t** *part, **uint64_t** signum, **uint64_t** sigkey, **int_t** disposition, **int_t** recursion)
- *Insert a spam signature training link into a specified part of a mime message.* **void mail_signature_add** (**mail_message_t** *message, **server_t** *server, **uint64_t** signum, **uint64_t** sigkey, **int_t** disposition)

Insert a spam signature training link into a mail message.

Detailed Description

Functions used to insert signatures into mail messages.

Definition in file **signatures.c**.

Function Documentation

stringer_t* mail_build_signature (**server_t** * server, **int_t** content_type, **int_t** content_encoding, **uint64_t** signum, **uint64_t** sigkey, **int_t** disposition)

Build a spam text signature for insertion into a message body.

signatures.c

Parameters:

server the server object of the web server where the teacher application is hosted.

type MESSAGE_TYPE_HTML to generate an html-based signature, or any other value for plain text.

content_encoding if MESSAGE_ENCODING_QUOTED_PRINTABLE, set the encoding type to qp.

signum the spam signature number referenced by the teacher url.
sigkey the spam signature key for client verification in the teacher app.
disposition if 0, the message disposition is "innocent"; otherwise, the disposition specifies "spam".

Returns:

NULL on failure, or a newly allocated managed string containing the desired mail signature on success.
Definition at line 25 of file signatures.c.

References `server_t::domain`, `length`, `log_pedantic`, `MESSAGE_ENCODING_QUOTED_PRINTABLE`, `MESSAGE_TYPE_HTML`, `qp_encode()`, `st_alloc`, `st_char_get()`, `st_cleanup()`, `st_free()`, `st_length_get()`, `st_merge`, `st_sprint()`, and `uint64_digits()`.

Referenced by `mail_insert_chunk_base64()`, and `mail_insert_chunk_text()`.

int_t mail_discover_encoding (stringer_t * header)

Get the value of the Content-Transfer-Encoding header in a mail message header.

Note:

Possible return values include `MESSAGE_ENCODING_QUOTED_PRINTABLE`, `MESSAGE_ENCODING_BASE64`, `MESSAGE_ENCODING_8BIT`, and `MESSAGE_ENCODING_7BIT`.

Parameters:

header a managed string containing the mail message header to be parsed.

Returns:

the `MESSAGE_ENCODING` code of the mail transfer encoding type, or `MESSAGE_ENCODING_7BIT` by default.

Definition at line 176 of file signatures.c.

References `content`, `mail_header_fetch_all()`, `MESSAGE_ENCODING_7BIT`, `MESSAGE_ENCODING_8BIT`, `MESSAGE_ENCODING_BASE64`, `MESSAGE_ENCODING_QUOTED_PRINTABLE`, `MESSAGE_ENCODING_UNKNOWN`, `mm_cmp_ci_eq()`, `PLACER`, `st_char_get()`, `st_free()`, and `st_length_get()`.

Referenced by `mail_modify_part()`.

size_t mail_discover_insertion_point (stringer_t * message, stringer_t * part, int_t type)

Find the position in a mail message where a custom message can be inserted.

Note:

The *part* parameter is expected to be a `placer` pointing into the contents of message. If the encoding type is not html, the insertion point is determined to be at the end of the part. If the encoding type is html, the following rules are followed: 1. Scanning backwards from the end of the message, skip trailing whitespace get the next html tag. 2. If that tag is NOT `</html>` or `</body>`, insert the signature AFTER it. 3. If that tag IS `</html>` or `</body>`, insert the signature right before it closes.

Parameters:

message a managed string containing the mail message body to be parsed.

part a managed string (`placer`) containing the part of the message where the custom message should be inserted.

type the encoding type of the message (`MESSAGE_TYPE_HTML` or other).

Returns:

the zero-based index of the position in the specified message where the custom message can be inserted.
 Definition at line 301 of file signatures.c.

References `length`, `mail_extract_tag()`, `MESSAGE_TYPE_HTML`, `PLACER`, `st_char_get()`, `st_cmp_ci_eq()`, `st_data_get()`, `st_free()`, and `st_length_get()`.

Referenced by `mail_insert_chunk_base64()`, and `mail_insert_chunk_text()`.

int_t mail_discover_type (stringer_t * header)

Get the value of the Content-Type header in a mail message header.

Note:

Possible return values include `MESSAGE_TYPE_MULTI_ALTERNATIVE`, `MESSAGE_TYPE_MULTI_RELATED`, `MESSAGE_TYPE_MULTI_MIXED`, `MESSAGE_TYPE_HTML`, `MESSAGE_TYPE_MULTI_UNKOWN`, and `MESSAGE_TYPE_PLAIN`.

Parameters:

header a managed string containing the mail message header to be parsed.

Returns:

the `MESSAGE_TYPE` code of the mail content type, or `MESSAGE_TYPE_PLAIN` by default.
 Definition at line 125 of file signatures.c.

References `content`, `mail_header_fetch_all()`, `MESSAGE_TYPE_HTML`, `MESSAGE_TYPE_MULTI_ALTERNATIVE`, `MESSAGE_TYPE_MULTI_MIXED`, `MESSAGE_TYPE_MULTI_RELATED`, `MESSAGE_TYPE_MULTI_UNKOWN`, `MESSAGE_TYPE_PLAIN`, `mm_cmp_ci_eq()`, `PLACER`, `st_char_get()`, `st_free()`, and `st_length_get()`.

Referenced by `mail_modify_part()`.

stringer_t* mail_extract_tag (chr_t * stream, size_t length)

Extract an html tag from a data buffer, searching backwards.

Warning:

Remember that this function takes the end and not the start of a buffer!

Note:

The extracted tag contains the enclosing brackets and only printable characters (no whitespace).

Parameters:

stream the end of a data buffer containing the html data to be parsed.
length the size, in bytes, of the data buffer to be parsed.

Returns:

NULL on failure, or a managed string containing the printable contents of the nearest html tag on success.
 Definition at line 223 of file signatures.c.

References `log_pedantic`, `st_alloc`, `st_char_get()`, `st_free()`, and `st_length_set()`.

Referenced by `mail_discover_insertion_point()`.

stringer_t* mail_get_boundary (stringer_t * header)

Get the boundary string from a message header.

Note:

This function works by parsing the Content-Type header if it exists, falling back to the entire header otherwise. The returned boundary string includes a trailing "--".

Parameters:

header a managed string containing the message header.

Returns:

NULL on failure or a managed string containing the boundary string on success.

Definition at line 613 of file signatures.c.

References `content`, `length`, `log_pedantic`, `mail_header_fetch_all()`, `PLACER`, `st_char_get()`, `st_cleanup()`, `st_data_get()`, `st_length_get()`, `st_merge`, and `st_search_ci()`.

Referenced by `mail_modify_part()`.

stringer_t* mail_get_chunk (stringer_t * *message*, stringer_t * *boundary*, int_t *chunk*)

Get a specified chunk (mime part) of a multipart mime message.

Parameters:

message a managed string containing the mime message to be parsed.

boundary a managed string containing the boundary used to split the multipart mime message.

chunk the one-index based chunk to be retrieved from the multipart message

Returns:

NULL on failure or a placer containing the specified chunk on success.

Definition at line 546 of file signatures.c.

References `length`, `log_pedantic`, `mm_cmp_cs_eq()`, `PLACER`, `st_char_get()`, `st_length_get()`, and `st_search_cs()`.

Referenced by `mail_modify_part()`.

stringer_t* mail_insert_chunk_base64 (server_t * *server*, stringer_t * *message*, stringer_t * *part*, uint64_t *signum*, uint64_t *sigkey*, int_t *disposition*, int_t *type*, int_t *encoding*)

Insert a spam signature training link into a base64-encoded message part.

See also:

`mail_discover_insertion_point()`

Note:

This is similar to `mail_insert_chunk_text()` except the part has to be decoded and then re-encoded after the training link is inserted.

Parameters:

server the server object of the web server where the teacher application is hosted.

message a managed string containing the base64-encoded message body to be parsed.

part a managed string (placer) containing the part of the message where the signature should be inserted.

signum the spam signature number referenced by the teacher url.

sigkey the spam signature key for client verification in the teacher app.

disposition if 0, the message disposition is "innocent"; otherwise, the disposition specifies "spam".
type the encoding type of the message (MESSAGE_TYPE_HTML or other).
encoding if MESSAGE_ENCODING_QUOTED_PRINTABLE, set the encoding type to qp.

Returns:

NULL on failure or a managed string containing the base64-encoded message with the inserted signature training link on success.

Definition at line 366 of file signatures.c.

References base64_decode(), base64_encode(), length, log_pedantic, mail_build_signature(), mail_discover_insertion_point(), PLACER, st_char_get(), st_cleanup(), st_data_get(), st_free(), st_length_get(), and st_merge.

Referenced by mail_modify_part().

stringer_t* mail_insert_chunk_text (server_t * server, stringer_t * message, stringer_t * part, uint64_t signum, uint64_t sigkey, int_t disposition, int_t type, int_t encoding)

Insert a spam signature training link into a plain text message part.

See also:

mail_discover_insertion_point()

Parameters:

server the server object of the web server where the teacher application is hosted.
message a managed string containing the message body to be parsed.
part a managed string (placer) containing the part of the message where the signature should be inserted.
signum the spam signature number referenced by the teacher url.
sigkey the spam signature key for client verification in the teacher app.
disposition if 0, the message disposition is "innocent"; otherwise, the disposition specifies "spam".
type the encoding type of the message (MESSAGE_TYPE_HTML or other).
encoding if MESSAGE_ENCODING_QUOTED_PRINTABLE, set the encoding type to qp.

Returns:

NULL on failure or a managed string containing the message with the inserted signature training link on success.

Definition at line 507 of file signatures.c.

References log_pedantic, mail_build_signature(), mail_discover_insertion_point(), PLACER, st_char_get(), st_empty(), st_free(), st_length_get(), and st_merge.

Referenced by mail_modify_part().

int_t mail_modify_part (server_t * server, mail_message_t * message, stringer_t * part, uint64_t signum, uint64_t sigkey, int_t disposition, int_t recursion)

Insert a spam signature training link into a specified part of a mime message.

Warning:

This function may fail to work as expected and still return a value of 1.

Note:

If the mime type is MESSAGE_TYPE_UNKNOWN (binary file), the function returns immediately. If the type is MESSAGE_TYPE_HTML or MESSAGE_TYPE_PLAIN, the training link is inserted normally. For MESSAGE_TYPE_MULTI_RELATED, MESSAGE_TYPE_MULTI_MIXED,

MESSAGE_TYPE_MULTI_UNKOWN the link is inserted in the first mime part. For MESSAGE_TYPE_MULTI_ALTERNATIVE, the link is inserted into each part, for up to 8 times.

Parameters:

server the server object of the web server where the teacher application is hosted.
message the mail message object of the message to be modified.
part a placer pointing to the specified part of the message to be modified.
signum the spam signature number referenced by the teacher url.
sigkey the spam signature key for client verification in the teacher app.
disposition if 0, the message disposition is "innocent"; otherwise, the disposition specifies "spam".
recursion an incremented recursion level tracker for calling this function, to prevent an overflow from occurring.

Returns:

0 on recursion failure, or 1 otherwise.

Definition at line 699 of file signatures.c.

References `length`, `log_pedantic`, `mail_discover_encoding()`, `mail_discover_type()`, `mail_get_boundary()`, `mail_get_chunk()`, `mail_insert_chunk_base64()`, `mail_insert_chunk_text()`, `mail_modify_part()`, `MAIL_SIGNATURES_RECURSION_LIMIT`, `MESSAGE_ENCODING_BASE64`, `MESSAGE_TYPE_HTML`, `MESSAGE_TYPE_MULTI_ALTERNATIVE`, `MESSAGE_TYPE_MULTI_MIXED`, `MESSAGE_TYPE_MULTI_RELATED`, `MESSAGE_TYPE_MULTI_UNKOWN`, `MESSAGE_TYPE_PLAIN`, `MESSAGE_TYPE_UNKNOWN`, `PLACER`, `st_char_get()`, `st_data_get()`, `st_free()`, `st_length_get()`, `mail_message_t::text`, and `type()`.

Referenced by `mail_modify_part()`, and `mail_signature_add()`.

```
void mail_signature_add (mail_message_t * message, server_t * server, uint64_t signum, uint64_t sigkey, int_t disposition)
```

Insert a spam signature training link into a mail message.

See also:

`mail_modify_part()`

Parameters:

message the mail message object of the message to be modified.
server the server object of the web server where the teacher application is hosted.
signum the spam signature number referenced by the teacher url.
sigkey the spam signature key for client verification in the teacher app.
disposition if 0, the message disposition is "innocent"; otherwise, the disposition specifies "spam".

Returns:

This function returns no value.

Definition at line 824 of file signatures.c.

References `log_pedantic`, `mail_modify_part()`, `PLACER`, `st_char_get()`, `st_length_get()`, `st_length_int()`, and `mail_message_t::text`.

Referenced by `mail_add_forward_headers()`, and `mail_load_message()`.

magma/objects/mail/store_message.c File Reference

Functions used to store and copy mail message data.

```
#include "magma.h"
```

Functions

- **bool_t mail_store_message_data** (uint64_t messagenum, uint8_t fflags, void *data, size_t data_len, chr_t **pathptr)
- *Persist a message's data to disk.* uint64_t **mail_store_message** (uint64_t usernum, stringer_t *pubkey, uint64_t foldernum, uint32_t *status, uint64_t signum, uint64_t sigkey, stringer_t *message)
- *Store a mail message, with its meta-information in the database, and the contents persisted to disk.* uint64_t **mail_copy_message** (uint64_t usernum, uint64_t original, chr_t *server, uint32_t size, uint64_t foldernum, uint32_t status, uint64_t signum, uint64_t sigkey, uint64_t created)
- *Create a copy of a mail message, with a new entry in the database and a hard link to the message contents on disk.* int_t **mail_move_message** (uint64_t usernum, uint64_t messagenum, uint64_t source, uint64_t target)

Move a message to a new folder in the database.

Detailed Description

Functions used to store and copy mail message data.

Definition in file **store_message.c**.

Function Documentation

uint64_t mail_copy_message (uint64_t usernum, uint64_t original, chr_t * server, uint32_t size, uint64_t foldernum, uint32_t status, uint64_t signum, uint64_t sigkey, uint64_t created)

Create a copy of a mail message, with a new entry in the database and a hard link to the message contents on disk.

store_message.c

Parameters:

usenum the numerical id of the user to whom the mail message belongs.

original the numerical id of the mail message to be copied.

server a pointer to a null-terminated string containing the name of the server where the message contents are stored.

size the size, in bytes, of the mail message to be copied.

foldernum the numerical id of the folder to become the parent folder of the message copy.

signum the spam signature for the message.

sigkey the spam key for the message.

created the UNIX timestamp of when the message was created.

Returns:

0 on failure, or the ID of the copy of the mail message in the database on success.

Definition at line 222 of file store_message.c.

References log_error, log_pedantic, mail_create_directory(), mail_db_insert_duplicate_message(), mail_message_path(), ns_free(), tran_commit(), tran_rollback(), and tran_start().

Referenced by `imap_message_copier()`, and `meta_messages_copier()`.

`int_t mail_move_message (uint64_t usernum, uint64_t messagenum, uint64_t source, uint64_t target)`

Move a message to a new folder in the database.

Parameters:

usenum the numerical id of the user that owns the message.

messagenum the numerical id of the message to be moved.

source the numerical id of the current parent folder of the specified message.

target the numerical id of the folder to which the specified message will be moved.

Returns:

-1 on error, 0 if the message wasn't found, or 1 on success.

Definition at line 307 of file `store_message.c`.

References `log_error`, `log_pedantic`, `mail_db_update_message_folder()`, `tran_commit()`, `tran_rollback()`, and `tran_start()`.

Referenced by `meta_messages_mover()`.

`uint64_t mail_store_message (uint64_t usernum, stringer_t * pubkey, uint64_t foldernum, uint32_t * status, uint64_t signum, uint64_t sigkey, stringer_t * message)`

Store a mail message, with its meta-information in the database, and the contents persisted to disk.

Note:

The stored message is always compressed, but only encrypted if the user's public key is supplied.

Parameters:

usenum the numerical id of the user to which the message belongs.

pubkey if not NULL, a public key that will be used to encrypt the message for the intended user.

foldernum the folder # that will contain the message.

status a pointer to the status flags value for the message, which will be updated if the message is to be encrypted.

signum the spam signature for the message.

sigkey the spam key for the message.

message a managed string containing the raw body of the message.

Returns:

0 on failure, or the newly inserted id of the message in the database on success.

Definition at line 111 of file `store_message.c`.

References `compress_free()`, `compress_lzo()`, `compress_total_length()`, `cryptex_free()`, `cryptex_total_length()`, `ecies_encrypt()`, `ECIES_PUBLIC_BINARY`, `FMMESSAGE_OPT_COMPRESSED`, `FMMESSAGE_OPT_ENCRYPTED`, `log_error`, `log_pedantic`, `mail_db_insert_message()`, `MAIL_STATUS_ENCRYPTED`, `mail_store_message_data()`, `ns_free()`, `st_length_get()`, `st_length_int()`, `tran_commit()`, `tran_rollback()`, and `tran_start()`.

Referenced by `imap_append_message()`, and `smtp_store_message()`.

bool_t mail_store_message_data (uint64_t *messagenum*, uint8_t *fflags*, void * *data*, size_t *data_len*, chr_t ** *pathptr*)

Persist a message's data to disk.

Parameters:

messagenum the numerical id of the message that will be associated with the data.

data a pointer to a buffer containing the message's data.

fflags the status flags to be stored in the message's on-disk file header.

data_len the size, in bytes, of the message's data.

pathptr if not NULL, the address of a pointer to a string that will receive a copy of the message's on-disk data.

Returns:

true if the storage operation succeeded, or false on failure.

Definition at line 25 of file store_message.c.

References FMESSAGE_MAGIC_1, FMESSAGE_MAGIC_2, log_error, mail_create_directory(),
mail_message_path(), message_fheader_t, and ns_free().

Referenced by mail_store_message().

magma/objects/messages/messages.h File Reference

Mail message operations.

Defines

- `#define FMESSAGE_MAGIC_1 0x17`
- `#define FMESSAGE_MAGIC_2 0x76`
- `#define FMESSAGE_OPT_COMPRESSED 0x1`
- `#define FMESSAGE_OPT_ENCRYPTED 0x2`

Functions

- `struct __attribute__((__packed__))`
- `struct __attribute__((packed))`
- `message_t * message_alloc (uint64_t messagenum, uint64_t created, uint64_t signature, uint64_t key, uint64_t flags, stringer_t *server, size_t size)`
- *messages.c* void `message_free (message_t *message)`
- *Free a message object and its underlying data.* `inx_t * messages_update (uint64_t usernum)`
- *Fetch all of a user's message folders and their child messages from the database.* `meta_message_t * meta_message_by_number (inx_t *messages, uint64_t number)`
- *meta.c* `meta_message_t * meta_message_dupe (meta_message_t *message)`
- *Duplicate a meta message object (along with a deep copy of its tags).* void `meta_message_free (meta_message_t *message)`
- *Free a meta message object (and its tags).* `bool_t meta_messages_copier (meta_user_t *user, meta_message_t *message, uint64_t target, uint64_t *outnum, bool_t sequences, META_LOCK_STATUS locked)`
- `bool_t meta_messages_login_update (meta_user_t *user, META_LOCK_STATUS locked)`
- *Build a user's messages collection if it is empty, or needs to be refreshed (see note).* `int_t meta_messages_mover (meta_user_t *user, meta_message_t *message, uint64_t target, bool_t lookup, bool_t sequences, META_LOCK_STATUS locked)`
- `int_t meta_messages_update (meta_user_t *user, META_LOCK_STATUS locked)`
- *Refresh and resquence a user's message collection if it is stale.* void `meta_messages_update_sequences (inx_t *folders, inx_t *messages)`
- *Update the sequence numbers of a series of messages, each re-indexed by their containing folder.* `bool_t messages_fetch (uint64_t usernum, message_folder_t *folder)`

datatier.c Variables

- `message_t`
- `message_fheader_t`

Detailed Description

Mail message operations.

Definition in file `messages.h`.

Define Documentation

#define FMESSAGE_MAGIC_1 0x17

Definition at line 53 of file messages.h.

Referenced by `adjust_message_encryption()`, `mail_load_header()`, `mail_load_message()`, and `mail_store_message_data()`.

#define FMESSAGE_MAGIC_2 0x76

Definition at line 54 of file messages.h.

Referenced by `adjust_message_encryption()`, `mail_load_header()`, `mail_load_message()`, and `mail_store_message_data()`.

#define FMESSAGE_OPT_COMPRESSED 0x1

Definition at line 56 of file messages.h.

Referenced by `mail_store_message()`.

#define FMESSAGE_OPT_ENCRYPTED 0x2

Definition at line 57 of file messages.h.

Referenced by `adjust_message_encryption()`, `mail_load_header()`, `mail_load_message()`, and `mail_store_message()`.

Function Documentation

struct __attribute__((packed)) [read]

Definition at line 60 of file messages.h.

References `__attribute__::flags`.

struct __attribute__((__packed__)) [read]

LOW: Update the Messages table columns so they match the tank mail message header fields. Store individual header/body lengths and the compressed/uncompressed hash values; then update the message type to store and use the new information.

Definition at line 19 of file messages.h.

References `__attribute__::body`, `__attribute__::created`, `__attribute__::data`, `__attribute__::flags`, `__attribute__::key`, `lock`, `__attribute__::message`, and `status`.

**message_t* message_alloc (uint64_t *messagenum*, uint64_t *created*, uint64_t *signature*,
uint64_t *key*, uint64_t *flags*, stringer_t* *server*, size_t *size*)**

messages.c

messages.c

Parameters:

messageum the numerical message id of the new message.
created the message creation timestamp.
signature the message's spam filter signature number.
key the message's spam filter access key.
flags the message's flags value.
server a managed string containing the name of the server where the message will be stored.
size the size, in bytes, of the raw message data.

Returns:

NULL on failure, or the newly allocated message object on success.

Definition at line 65 of file messages.c.

References `align()`, `FOREIGNDATA`, `JOINTED`, `log_pedantic`, `message_t`, `mm_alloc()`, `mm_copy()`, `mm_free()`, `PLACER_T`, `placer_t`, `rwlock_init()`, `st_data_get()`, `st_length_get()`, and `STACK`.

Referenced by `messages_fetch()`.

void message_free (message_t * message)

Free a message object and its underlying data.

Returns:

This function returns no value.

Definition at line 44 of file messages.c.

References `mm_cleanup()`, `rwlock_destroy()`, and `st_cleanup()`.

Referenced by `message_folder_alloc()`, and `messages_fetch()`.

bool_t messages_fetch (uint64_t usernum, message_folder_t * folder)

datatier.c

datatier.c

Parameters:

usernum the numerical id of the user that owns the folder.
folder a pointer to the message folder object to be populated.

Returns:

true on success or false on failure.

Definition at line 21 of file datatier.c.

References `inx_insert()`, `log_info`, `log_pedantic`, `M_TYPE_UINT64`, `message_alloc()`, `message_free()`, `message_t`, `mm_wipe()`, `PLACER`, `res_field_block()`, `res_field_length()`, `res_field_uint32()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `stmt_get_result()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `messages_update()`.

inx_t* messages_update (uint64_t usernum)

Fetch all of a user's message folders and their child messages from the database.

Parameters:

usernum the numerical id of the target user.

Returns:

NULL on failure or an `inx` object holding all the retrieved and populated message folder objects on success.
Definition at line 101 of file `messages.c`.

References `inx_cleanup()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_free()`, `log_pedantic`, `M_FOLDER_MESSAGES`, `magma_folder_fetch()`, and `messages_fetch()`.

Referenced by `meta_message_folders_update()`.

`meta_message_t* meta_message_by_number (inx_t * messages, uint64_t number)`

`meta.c`

`meta.c`

Parameters:

messages a pointer to the messages collection to be searched.

number the number of the message to be retrieved.

Returns:

NULL on failure, or a pointer to the meta message object of the matching mail message.
Definition at line 60 of file `meta.c`.

References `inx_cursor_alloc()`, `inx_cursor_free()`, and `inx_cursor_value_next()`.

`meta_message_t* meta_message_dupe (meta_message_t * message)`

Duplicate a meta message object (along with a deep copy of its tags).

Parameters:

message the meta message object to be cloned.

Returns:

NULL on failure or a pointer to the duplicated meta message object on success.
Definition at line 39 of file `meta.c`.

References `ar_dupe()`, `mm_dupe()`, and `meta_message_t::tags`.

Referenced by `imap_duplicate_messages()`, `imap_message_copier()`, `imap_search_messages()`, and `meta_messages_copier()`.

`void meta_message_free (meta_message_t * message)`

Free a meta message object (and its tags).

Returns:

This function returns no value.
Definition at line 20 of file `meta.c`.

References `ar_free()`, `mm_free()`, and `meta_message_t::tags`.

Referenced by `imap_duplicate_messages()`, `imap_search_messages()`, and `meta_data_fetch_messages()`.

bool_t meta_messages_copier (meta_user_t * user, meta_message_t * message, uint64_t target, uint64_t * outnum, bool_t sequences, META_LOCK_STATUS locked)

Parameters:

Make a copy of a mail message, in the database and in memory.

See also:

mail_copy_message()

Note:

The message copy will not have the deleted, hidden, or recent flags set in the database, but it will have the recent flag set in memory.

Parameters:

user a pointer to the meta user object to whom the message belong.

message the meta message object of the message to be copied.

target the numerical id of the folder to which the message will be copied.

outnum a pointer to an unsigned 64-bit integer that will receive the value of the numerical id of the message copy.

sequences if true, resequence the message folders after the copy has been made.

locked if set to META_NEED_LOCK, lock the specified meta user object for the duration of the request.

Returns:

true on success or false on failure.

Definition at line 244 of file meta.c.

References meta_message_t::created, meta_user_t::folders, inx_insert(), log_error, log_pedantic, M_TYPE_UINT64, mail_copy_message(), MAIL_STATUS_DELETED, MAIL_STATUS_HIDDEN, MAIL_STATUS_RECENT, meta_message_t::messagenum, meta_user_t::messages, meta_message_dupe(), meta_messages_update_sequences(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), mm_free(), meta_message_t::server, meta_message_t::sigkey, meta_message_t::signum, meta_message_t::size, meta_message_t::status, status, multi_t::u64, meta_user_t::usernum, and multi_t::val.

Referenced by portal_endpoint_messages_copy().

bool_t meta_messages_login_update (meta_user_t * user, META_LOCK_STATUS locked)

Build a user's messages collection if it is empty, or needs to be refreshed (see note).

Note:

This function will fetch and sequence the user's messages from the database if the meta user object has no messages, or if the user has one or fewer pop sessions open and the messages serial number was stale.

Parameters:

user a pointer to the meta user object of the user requesting the messages update.

locked if set to META_NEED_LOCK, lock the specified meta user object for the duration of the request.

Returns:

true if no update was necessary or if the update was successful, or false otherwise.

Definition at line 131 of file meta.c.

References meta_user_t::folders, meta_user_t::messages, meta_data_fetch_messages(), meta_messages_update_sequences(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), OBJECT_MESSAGES, meta_user_t::pop, meta_user_t::refs, serial_get(), serial_increment(), meta_user_t::serials, and meta_user_t::usernum.

Referenced by pop_pass().

int_t meta_messages_mover (meta_user_t * user, meta_message_t * message, uint64_t target, bool_t lookup, bool_t sequences, META_LOCK_STATUS locked)

Parameters:

Move a mail message to another folder.

See also:

mail_move_message()

Note:

The message will receive a copy of the recent flag in memory.

Parameters:

user a pointer to the meta user object to whom the message belong.

message the meta message object of the message to be moved.

target the numerical id of the folder to which the message will be copied.

lookup if set, verify the existence of the moved message in the user's messages.

sequences if true, resequence the message folders after the copy has been made.

locked if set to META_NEED_LOCK, lock the specified meta user object for the duration of the request.

Returns:

true on success or false on failure.

BUG: Currently we preserve the original message number which means when the target folder sequences are updated the moved message can appear anywhere in the list (since its based on message number). Inserting messages into the sequence list in this fashion will probably break clients. We may want to duplicate the row which would generate a new message number causing the newly added message to appear at the end of the folder listing. On the other hand POP ignores folders completely, so maybe it won't be quite so bad.

Definition at line 322 of file meta.c.

References meta_message_t::foldernum, meta_user_t::folders, inx_find(), log_pedantic, M_TYPE_UINT64, mail_move_message(), MAIL_STATUS_RECENT, meta_message_t::messagenum, meta_user_t::messages, meta_messages_update_sequences(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), meta_message_t::status, multi_t::u64, meta_user_t::usernum, and multi_t::val.

Referenced by portal_endpoint_messages_move().

int_t meta_messages_update (meta_user_t * user, META_LOCK_STATUS locked)

Refresh and resquence a user's message collection if it is stale.

Note:

The user's messages will only be updated if they are empty or if they are out of sync and the user has no open pop sessions.

See also:

meta_data_fetch_messages()

Parameters:

user a pointer to the meta user object requesting the messages update.

locked if set to META_NEED_LOCK, lock the specified meta user object for the duration of the request.

Returns:

-1 on failure, or 1 on success.

Definition at line 185 of file meta.c.

References meta_user_t::folders, meta_user_t::messages, meta_data_fetch_messages(), meta_messages_update_sequences(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), OBJECT_MESSAGES, meta_user_t::pop, meta_user_t::refs, serial_get(), serial_increment(), meta_user_t::serials, and meta_user_t::usernum.

Referenced by imap_session_update(), meta_get(), and sess_update().

void meta_messages_update_sequences (inx_t * *folders*, inx_t * *messages*)

Update the sequence numbers of a series of messages, each re-indexed by their containing folder.

Note:

All messages will be sequenced incrementally per folder, starting with a value of 1.

Parameters:

folders an inx holder containing all of the user's meta folder records.

messages an inx folder containing all the messages to be re-sequenced.

Returns:

This function returns no value.

Definition at line 89 of file meta.c.

References meta_folder_t::foldernum, meta_message_t::foldernum, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), and meta_message_t::sequencenum.

Referenced by imap_append_message(), imap_close(), imap_expunge(), imap_message_copier(), meta_folders_update(), meta_messages_copier(), meta_messages_login_update(), meta_messages_mover(), meta_messages_update(), pop_session_destroy(), portal_endpoint_messages_copy(), portal_endpoint_messages_move(), and portal_endpoint_messages_remove().

Variable Documentation

message_fheader_t

Definition at line 65 of file messages.h.

Referenced by adjust_message_encryption(), mail_load_header(), mail_load_message(), and mail_store_message_data().

message_t

Definition at line 51 of file messages.h.

Referenced by message_alloc(), and messages_fetch().

magma/objects/messages/meta.c File Reference

Functions to interface with the deprecated **meta_message_t** structure.

```
#include "magma.h"
```

Functions

- void **meta_message_free** (**meta_message_t** *message)
 - *Free a meta message object (and its tags).* **meta_message_t** * **meta_message_dupe** (**meta_message_t** *message)
 - *Duplicate a meta message object (along with a deep copy of its tags).* **meta_message_t** * **meta_message_by_number** (**inx_t** *messages, uint64_t number)
 - *Retrieve a message by number.* void **meta_messages_update_sequences** (**inx_t** *folders, **inx_t** *messages)
 - *Update the sequence numbers of a series of messages, each re-indexed by their containing folder.* **bool_t** **meta_messages_login_update** (**meta_user_t** *user, **META_LOCK_STATUS** locked)
 - *Build a user's messages collection if it is empty, or needs to be refreshed (see note).* **int_t** **meta_messages_update** (**meta_user_t** *user, **META_LOCK_STATUS** locked)
 - *Refresh and resquence a user's message collection if it is stale.* **bool_t** **meta_messages_copier** (**meta_user_t** *user, **meta_message_t** *message, uint64_t target, uint64_t outnum, **bool_t** sequences, **META_LOCK_STATUS** locked)
 - **int_t** **meta_messages_mover** (**meta_user_t** *user, **meta_message_t** *message, uint64_t target, **bool_t** lookup, **bool_t** sequences, **META_LOCK_STATUS** locked)
-

Detailed Description

Functions to interface with the deprecated **meta_message_t** structure.

Definition in file **meta.c**.

Function Documentation

meta_message_t* **meta_message_by_number** (**inx_t** * *messages*, uint64_t *number*)

Retrieve a message by number.

meta.c

Parameters:

messages a pointer to the messages collection to be searched.

number the number of the message to be retrieved.

Returns:

NULL on failure, or a pointer to the meta message object of the matching mail message.

Definition at line 60 of file meta.c.

References **inx_cursor_alloc()**, **inx_cursor_free()**, and **inx_cursor_value_next()**.

meta_message_t* **meta_message_dupe** (**meta_message_t** * *message*)

Duplicate a meta message object (along with a deep copy of its tags).

Parameters:

message the meta message object to be cloned.

Returns:

NULL on failure or a pointer to the duplicated meta message object on success.

Definition at line 39 of file meta.c.

References `ar_dupe()`, `mm_dupe()`, and `meta_message_t::tags`.

Referenced by `imap_duplicate_messages()`, `imap_message_copier()`, `imap_search_messages()`, and `meta_messages_copier()`.

void meta_message_free (meta_message_t * *message*)

Free a meta message object (and its tags).

Returns:

This function returns no value.

Definition at line 20 of file meta.c.

References `ar_free()`, `mm_free()`, and `meta_message_t::tags`.

Referenced by `imap_duplicate_messages()`, `imap_search_messages()`, and `meta_data_fetch_messages()`.

bool_t meta_messages_copier (meta_user_t * *user*, meta_message_t * *message*, uint64_t *target*, uint64_t * *outnum*, bool_t *sequences*, META_LOCK_STATUS *locked*)**Parameters:**

Make a copy of a mail message, in the database and in memory.

See also:

`mail_copy_message()`

Note:

The message copy will not have the deleted, hidden, or recent flags set in the database, but it will have the recent flag set in memory.

Parameters:

user a pointer to the meta user object to whom the message belong.

message the meta message object of the message to be copied.

target the numerical id of the folder to which the message will be copied.

outnum a pointer to an unsigned 64-bit integer that will receive the value of the numerical id of the message copy.

sequences if true, resequence the message folders after the copy has been made.

locked if set to `META_NEED_LOCK`, lock the specified meta user object for the duration of the request.

Returns:

true on success or false on failure.

Definition at line 244 of file meta.c.

References `meta_message_t::created`, `meta_user_t::folders`, `inx_insert()`, `log_error`, `log_pedantic`, `M_TYPE_UINT64`, `mail_copy_message()`, `MAIL_STATUS_DELETED`, `MAIL_STATUS_HIDDEN`, `MAIL_STATUS_RECENT`, `meta_message_t::messagenum`, `meta_user_t::messages`, `meta_message_dupe()`, `meta_messages_update_sequences()`, `META_NEED_LOCK`, `meta_user_unlock()`, `meta_user_wlock()`,

mm_free(), meta_message_t::server, meta_message_t::sigkey, meta_message_t::signum, meta_message_t::size, meta_message_t::status, status, multi_t::u64, meta_user_t::usernum, and multi_t::val.

Referenced by portal_endpoint_messages_copy().

bool_t meta_messages_login_update (meta_user_t * user, META_LOCK_STATUS locked)

Build a user's messages collection if it is empty, or needs to be refreshed (see note).

Note:

This function will fetch and sequence the user's messages from the database if the meta user object has no messages, or if the user has one or fewer pop sessions open and the messages serial number was stale.

Parameters:

user a pointer to the meta user object of the user requesting the messages update.

locked if set to META_NEED_LOCK, lock the specified meta user object for the duration of the request.

Returns:

true if no update was necessary or if the update was successful, or false otherwise.

Definition at line 131 of file meta.c.

References meta_user_t::folders, meta_user_t::messages, meta_data_fetch_messages(), meta_messages_update_sequences(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), OBJECT_MESSAGES, meta_user_t::pop, meta_user_t::refs, serial_get(), serial_increment(), meta_user_t::serials, and meta_user_t::usernum.

Referenced by pop_pass().

int_t meta_messages_mover (meta_user_t * user, meta_message_t * message, uint64_t target, bool_t lookup, bool_t sequences, META_LOCK_STATUS locked)

Parameters:

Move a mail message to another folder.

See also:

mail_move_message()

Note:

The message will receive a copy of the recent flag in memory.

Parameters:

user a pointer to the meta user object to whom the message belong.

message the meta message object of the message to be moved.

target the numerical id of the folder to which the message will be copied.

lookup if set, verify the existence of the moved message in the user's messages.

sequences if true, resequence the message folders after the copy has been made.

locked if set to META_NEED_LOCK, lock the specified meta user object for the duration of the request.

Returns:

true on success or false on failure.

BUG: Currently we preserve the original message number which means when the target folder sequences are updated the moved message can appear anywhere in the list (since its based on message number). Inserting messages into the sequence list in this fashion will probably break clients. We may want to duplicate the row which would generate a new message number causing the newly added message to appear at the end of the folder listing. On the other hand POP ignores folders completely, so maybe it won't be quite so bad.

Definition at line 322 of file meta.c.

References meta_message_t::foldernum, meta_user_t::folders, inx_find(), log_pedantic, M_TYPE_UINT64, mail_move_message(), MAIL_STATUS_RECENT, meta_message_t::messagenum, meta_user_t::messages, meta_messages_update_sequences(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), meta_message_t::status, multi_t::u64, meta_user_t::usernum, and multi_t::val.

Referenced by portal_endpoint_messages_move().

int_t meta_messages_update (meta_user_t * user, META_LOCK_STATUS locked)

Refresh and resquence a user's message collection if it is stale.

Note:

The user's messages will only be updated if they are empty or if they are out of sync and the user has no open pop sessions.

See also:

meta_data_fetch_messages()

Parameters:

user a pointer to the meta user object requesting the messages update.

locked if set to META_NEED_LOCK, lock the specified meta user object for the duration of the request.

Returns:

-1 on failure, or 1 on success.

Definition at line 185 of file meta.c.

References meta_user_t::folders, meta_user_t::messages, meta_data_fetch_messages(), meta_messages_update_sequences(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), OBJECT_MESSAGES, meta_user_t::pop, meta_user_t::refs, serial_get(), serial_increment(), meta_user_t::serials, and meta_user_t::usernum.

Referenced by imap_session_update(), meta_get(), and sess_update().

void meta_messages_update_sequences (inx_t * folders, inx_t * messages)

Update the sequence numbers of a series of messages, each re-indexed by their containing folder.

Note:

All messages will be sequenced incrementally per folder, starting with a value of 1.

Parameters:

folders an inx holder containing all of the user's meta folder records.

messages an inx folder containing all the messages to be re-sequenced.

Returns:

This function returns no value.

Definition at line 89 of file meta.c.

References meta_folder_t::foldernum, meta_message_t::foldernum, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), and meta_message_t::sequencenum.

Referenced by imap_append_message(), imap_close(), imap_expunge(), imap_message_copier(), meta_folders_update(), meta_messages_copier(), meta_messages_login_update(), meta_messages_mover(),

meta_messages_update(), pop_session_destroy(), portal_endpoint_messages_copy(),
portal_endpoint_messages_move(), and portal_endpoint_messages_remove().

magma/objects/neue/credentials.c File Reference

```
#include "magma.h"
```

Functions

- **stringer_t * credential_address (stringer_t *s)**
- *LOW: Figure out how to result a credential_address into a credential_username using the Mailboxes table.*
stringer_t * credential_username (stringer_t *s)
- *Get the valid credential username portion of a fully qualified user address.* void **credential_free (credential_t *cred)**
- *Free a user credential.* **credential_t * credential_alloc_mail (stringer_t *address)**
- *Get a user mail credential for anonymous use within the mail subsystem.* **credential_t * credential_alloc_auth (stringer_t *username, stringer_t *password)**

Construct a user credential object from supplied username and password.

Function Documentation

stringer_t* credential_address (stringer_t * s)

LOW: Figure out how to result a credential_address into a credential_username using the Mailboxes table.

credentials.c

LOW: Add a function for detecting potentially illegal username/address sequences. Valid usernames must start with an alpha character, end with an alphanumeric character and not user consecutive underscores. If present, the domain portion of the username must follow the applicable standard for the TLD being used. Process a user supplied credential address to ensures it only contains valid characters.

Parameters:

s a managed string containing the user's credential address to be processed, with an optional domain suffix.

Note:

This function duplicates the input address string, with all characters converted to lowercase, and whitespace removed. '.' and '-' are also converted to '_' in the username, and if there is a '+' in the username portion of the credential address, all subsequent characters in that username will be ignored.

Returns:

NULL on failure, or a managed string containing the validated credential address on success.

Definition at line 30 of file credentials.c.

References `chr_whitespace()`, `CONTIGUOUS`, `lower_chr()`, `MANAGED_T`, `PLACER`, `SECURE`, `st_alloc_opts()`, `st_char_get()`, `st_empty_out()`, `st_length_get()`, and `st_length_set()`.

Referenced by `credential_alloc_auth()`, `credential_alloc_mail()`, `credential_username()`, and `pop_user()`.

credential_t* credential_alloc_auth (stringer_t * username, stringer_t * password)

Construct a user credential object from supplied username and password.

Note:

The credential's auth.key field becomes a single pass hash of the password, while auth.password is created from a three-time hash.

Parameters:

username the input username.

password the plaintext password of the user.

Returns:

NULL on failure, or a pointer to the requested user's auth credentials on success.

Definition at line 207 of file credentials.c.

References `credential_t::auth`, `BLOCK_T`, `CONTIGUOUS`, `credential_address()`, `CREDENTIAL_AUTH`, `credential_free()`, `credential_username()`, `digest_sha512()`, `magma_t::domain`, `hex_encode_st()`, `JOINTED`, `magma`, `MANAGED_T`, `mm_alloc()`, `PLACER`, `magma_t::salt`, `magma_t::secure`, `SECURE`, `st_alloc_opts()`, `st_cleanup()`, `st_copy_in()`, `st_data_get()`, `st_empty()`, `st_free()`, `st_length_get()`, `st_merge_opts()`, `st_search_cs()`, `magma_t::system`, and `credential_t::type`.

Referenced by `imap_login()`, `pop_pass()`, `portal_endpoint_auth()`, `register_data_insert_user()`, `smtp_auth_login()`, `smtp_auth_plain()`, and `teacher_process()`.

credential_t* credential_alloc_mail (stringer_t * address)

Get a user mail credential for anonymous use within the mail subsystem.

Note:

This function is used by the smtp service to fetch the smtp inbound preferences of a recipient user.

Parameters:

address a managed string containing the email address of the user account.

Returns:

NULL on failure, or a pointer to the requested user's mail credentials on success.

Definition at line 168 of file credentials.c.

References `credential_t::auth`, `credential_address()`, `credential_free()`, `CREDENTIAL_MAIL`, `FOREIGNDATA`, `JOINTED`, `credential_t::mail`, `mm_alloc()`, `PLACER`, `PLACER_T`, `SECURE`, `st_alloc_opts()`, `st_data_get()`, `st_data_set()`, `st_length_get()`, `st_length_set()`, `st_search_cs()`, and `credential_t::type`.

Referenced by `smtp_rcpt_to()`.

void credential_free (credential_t * cred)

Free a user credential.

Parameters:

cred a pointer to the user credential object to be freed.

Returns:

This function returns no value.

Definition at line 140 of file credentials.c.

References `credential_t::auth`, `CREDENTIAL_AUTH`, `CREDENTIAL_MAIL`, `credential_t::mail`, `mm_free()`, `st_cleanup()`, and `credential_t::type`.

Referenced by `credential_alloc_auth()`, `credential_alloc_mail()`, `imap_login()`, `pop_pass()`, `register_data_insert_user()`, `sess_destroy()`, `smtp_auth_login()`, `smtp_auth_plain()`, `smtp_rcpt_to()`, and `teacher_process()`.

stringer_t* credential_username (stringer_t * s)

Get the valid credential username portion of a fully qualified user address.

Parameters:

s a managed string containing the user's credential address to be processed, with an optional domain suffix.

Returns:

NULL on failure, or a managed string containing the credential username portion of the supplied address.

Definition at line 115 of file credentials.c.

References `credential_address()`, `magma_t::domain`, `magma`, `PLACER`, `st_char_get()`, `st_cmp_cs_eq()`, `st_length_get()`, `st_length_set()`, `st_search_cs()`, and `magma_t::system`.

Referenced by `credential_alloc_auth()`, and `pop_pass()`.

magma/objects/neue/neue.c File Reference

Neue entry points.

```
#include "magma.h"
```

Functions

- void **neue_rlock** (**neue_t** *neue)
 - *This function is not called from anywhere and may be removed.* void **neue_wlock** (**neue_t** *neue)
 - *This function is not called from anywhere and may be removed.* void **neue_unlock** (**neue_t** *neue)
- This function is not called from anywhere and may be removed.*
-

Detailed Description

Neue entry points.

Definition in file **neue.c**.

Function Documentation

void neue_rlock (neue_t * *neue*)

This function is not called from anywhere and may be removed.

Definition at line 18 of file neue.c.

References neue_t::lock, and rwlock_lock_read().

void neue_unlock (neue_t * *neue*)

This function is not called from anywhere and may be removed.

Definition at line 43 of file neue.c.

References neue_t::lock, and rwlock_unlock().

void neue_wlock (neue_t * *neue*)

This function is not called from anywhere and may be removed.

Definition at line 30 of file neue.c.

References neue_t::lock, and rwlock_lock_write().

magma/objects/neue/neue.h File Reference

Data Structures

- struct **credential_t**
- struct **neue_t**

Enumerations

- enum { **M_NEUE_FLAGS_XXXX** = 1, **M_NEUE_FLAGS_YYYY** = 2 }

Functions

- **stringer_t * credential_address** (**stringer_t *s**)
- *credentials.c* **credential_t * credential_alloc_auth** (**stringer_t *username**, **stringer_t *password**)
- *Construct a user credential object from supplied username and password.* **credential_t * credential_alloc_mail** (**stringer_t *address**)
- *Get a user mail credential for anonymous use within the mail subsystem.* void **credential_free** (**credential_t *cred**)
- *Free a user credential.* **stringer_t * credential_username** (**stringer_t *s**)

Get the valid credential username portion of a fully qualified user address.

Enumeration Type Documentation

anonymous enum

Enumerator:

M_NEUE_FLAGS_XXXX **M_NEUE_FLAGS_XXXX.**
M_NEUE_FLAGS_YYYY **M_NEUE_FLAGS_YYYY.**

Definition at line 16 of file neue.h.

Function Documentation

stringer_t* credential_address (**stringer_t * s**)

credentials.c

credentials.c

LOW: Add a function for detecting potentially illegal username/address sequences. Valid usernames must start with an alpha character, end with an alphanumeric character and not user consecutive underscores. If present, the domain portion of the username must follow the applicable standard for the TLD being used. Process a user supplied credential address to ensures it only contains valid characters.

Parameters:

s a managed string containing the user's credential address to be processed, with an optional domain suffix.

Note:

This function duplicates the input address string, with all characters converted to lowercase, and whitespace removed. '.' and '-' are also converted to '_' in the username, and if there is a '+' in the username portion of the credential address, all subsequent characters in that username will be ignored.

Returns:

NULL on failure, or a managed string containing the validated credential address on success.

Definition at line 30 of file credentials.c.

References chr_whitespace(), CONTIGUOUS, lower_chr(), MANAGED_T, PLACER, SECURE, st_alloc_opts(), st_char_get(), st_empty_out(), st_length_get(), and st_length_set().

Referenced by credential_alloc_auth(), credential_alloc_mail(), credential_username(), and pop_user().

credential_t* credential_alloc_auth (stringer_t * username, stringer_t * password)

Construct a user credential object from supplied username and password.

Note:

The credential's auth.key field becomes a single pass hash of the password, while auth.password is created from a three-time hash.

Parameters:

username the input username.

password the plaintext password of the user.

Returns:

NULL on failure, or a pointer to the requested user's auth credentials on success.

Definition at line 207 of file credentials.c.

References credential_t::auth, BLOCK_T, CONTIGUOUS, credential_address(), CREDENTIAL_AUTH, credential_free(), credential_username(), digest_sha512(), magma_t::domain, hex_encode_st(), JOINTED, magma, MANAGED_T, mm_alloc(), PLACER, magma_t::salt, magma_t::secure, SECURE, st_alloc_opts(), st_cleanup(), st_copy_in(), st_data_get(), st_empty(), st_free(), st_length_get(), st_merge_opts(), st_search_cs(), magma_t::system, and credential_t::type.

Referenced by imap_login(), pop_pass(), portal_endpoint_auth(), register_data_insert_user(), smtp_auth_login(), smtp_auth_plain(), and teacher_process().

credential_t* credential_alloc_mail (stringer_t * address)

Get a user mail credential for anonymous use within the mail subsystem.

Note:

This function is used by the smtp service to fetch the smtp inbound preferences of a recipient user.

Parameters:

address a managed string containing the email address of the user account.

Returns:

NULL on failure, or a pointer to the requested user's mail credentials on success.

Definition at line 168 of file credentials.c.

References credential_t::auth, credential_address(), credential_free(), CREDENTIAL_MAIL, FOREIGNDATA, JOINTED, credential_t::mail, mm_alloc(), PLACER, PLACER_T, SECURE,

st_alloc_opts(), st_data_get(), st_data_set(), st_length_get(), st_length_set(), st_search_cs(), and credential_t::type.

Referenced by smtp_rcpt_to().

void credential_free (credential_t * cred)

Free a user credential.

Parameters:

cred a pointer to the user credential object to be freed.

Returns:

This function returns no value.

Definition at line 140 of file credentials.c.

References credential_t::auth, CREDENTIAL_AUTH, CREDENTIAL_MAIL, credential_t::mail, mm_free(), st_cleanup(), and credential_t::type.

Referenced by credential_alloc_auth(), credential_alloc_mail(), imap_login(), pop_pass(), register_data_insert_user(), sess_destroy(), smtp_auth_login(), smtp_auth_plain(), smtp_rcpt_to(), and teacher_process().

stringer_t* credential_username (stringer_t * s)

Get the valid credential username portion of a fully qualified user address.

Parameters:

s a managed string containing the user's credential address to be processed, with an optional domain suffix.

Returns:

NULL on failure, or a managed string containing the credential username portion of the supplied address.

Definition at line 115 of file credentials.c.

References credential_address(), magma_t::domain, magma, PLACER, st_char_get(), st_cmp_cs_eq(), st_length_get(), st_length_set(), st_search_cs(), and magma_t::system.

Referenced by credential_alloc_auth(), and pop_pass().

magma/objects/objects.h File Reference

Functions used to interface with objects.

```
#include "warehouse/warehouse.h"
#include "folders/folders.h"
#include "contacts/contacts.h"
#include "messages/messages.h"
#include "config/config.h"
#include "neue/neue.h"
#include "users/users.h"
#include "mail/mail.h"
#include "sessions/sessions.h"
```

Data Structures

- struct **object_cache_t**

Enumerations

- enum { **OBJECT_USER**, **OBJECT_CONFIG**, **OBJECT_FOLDERS**, **OBJECT_MESSAGES**, **OBJECT_CONTACTS** }

Functions

- **int_t lock_get** (**stringer_t** *key)
- *locks.c* void **lock_release** (**stringer_t** *key)
- *Release a named lock, with synchronization provided via memcached.* **int_t user_lock** (uint64_t usernum)
- *Acquire a lock in the magma.user keypace.* void **user_unlock** (uint64_t usernum)
- *Unlock a lock in the magma.user keypace.* **bool_t obj_cache_start** (void)
- *objects.c* void **obj_cache_prune** (void)
- void **obj_cache_stop** (void)
- *Stop the object cache and free all active user and web session objects.* uint64_t **serial_get** (uint64_t type, uint64_t num)
- *serials.c* uint64_t **serial_increment** (uint64_t type, uint64_t num)
- *Increment the serial number for an object in memcached.* uint64_t **serial_reset** (uint64_t type, uint64_t num)

Reset the serial number to 1 for an object in memcached. Variables

- **object_cache_t** objects

Detailed Description

Functions used to interface with objects.

Definition in file **objects.h**.

Enumeration Type Documentation

anonymous enum

Enumerator:

OBJECT_USER
OBJECT_CONFIG
OBJECT_FOLDERS
OBJECT_MESSAGES
OBJECT_CONTACTS

Definition at line 26 of file objects.h.

Function Documentation

int_t lock_get (stringer_t * key)

locks.c

locks.c

See also:

cache_silent_add()

Note:

The lock will be held for 10 seconds, and locking attempts will occur periodically for 60 seconds prior to failure.

Parameters:

key a managed string containing the name of the lock to be acquired.

Returns:

-1 on general failure, 0 on memcached failure, or 1 on success.

Definition at line 25 of file locks.c.

References `cache_silent_add()`, `lock`, `log_pedantic`, `MAGMA_LOCK_EXPIRATION`, `MAGMA_LOCK_TIMEOUT`, `MANAGEDBUF`, `PLACER`, `st_char_get()`, `st_empty()`, `st_length_int()`, and `st_sprint()`.

Referenced by `smtp_reply()`, and `user_lock()`.

void lock_release (stringer_t * key)

Release a named lock, with synchronization provided via memcached.

See also:

cache_delete()

Note:

The lock will be held for 10 seconds, and locking attempts will occur periodically for 60 seconds prior to failure.

Parameters:

key a managed string containing the name of the lock to be released.

Returns:

-1 on general failure, 0 on memcached failure, or 1 on success.

LOW: At some point we should add logic to check whether this cluster node even owns the lock before taking the drastic step of deleting it.

Definition at line 64 of file locks.c.

References cache_delete(), lock, log_pedantic, MANAGEDBUF, st_char_get(), st_empty(), st_length_int(), and st_sprint().

Referenced by smtp_reply(), and user_unlock().

void obj_cache_prune (void)

Definition at line 59 of file objects.c.

References count, inx_count(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_key_active(), inx_cursor_reset(), inx_cursor_value_next(), inx_delete(), inx_lock_read(), inx_lock_write(), inx_unlock(), meta_user_ref_stamp(), meta_user_ref_total(), sess_ref_stamp(), sess_ref_total(), object_cache_t::sessions, stats_adjust_by_name(), stats_set_by_name(), and object_cache_t::users.

Referenced by process_maint().

bool_t obj_cache_start (void)

objects.c

objects.c

Returns:

true on success or false on failure.

Definition at line 24 of file objects.c.

References inx_alloc(), log_critical, M_INX_HASHED, M_INX_LOCK_MANUAL, meta_user_destroy(), sess_destroy(), object_cache_t::sessions, and object_cache_t::users.

Referenced by process_start().

void obj_cache_stop (void)

Stop the object cache and free all active user and web session objects.

Returns:

This function returns no value.

Definition at line 43 of file objects.c.

References inx_free(), object_cache_t::sessions, and object_cache_t::users.

Referenced by process_stop().

uint64_t serial_get (uint64_t type, uint64_t num)

serials.c

serials.c

Parameters:

type the serial type to be queried (OBJECT_USER, OBJECT_CONFIG, OBJECT_FOLDERS, OBJECT_MESSAGES, or OBJECT_CONTACTS).
num the specific object identifier.

Returns:

0 on failure or the serial number of the requested object.

Definition at line 63 of file serials.c.

References `cache_increment()`, `log_pedantic`, `serial_prefix()`, `st_aprint()`, `st_char_get()`, `st_free()`, and `st_length_int()`.

Referenced by `imap_append_message()`, `imap_close()`, `imap_create()`, `imap_delete()`, `imap_expunge()`, `imap_fetch()`, `imap_message_copier()`, `imap_rename()`, `imap_session_update()`, `imap_store()`, `meta_contacts_update()`, `meta_folders_update()`, `meta_message_folders_update()`, `meta_messages_login_update()`, `meta_messages_update()`, `meta_user_build()`, `meta_user_serial_check()`, `meta_user_update()`, `portal_endpoint_folders_rename()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `user_config_create()`, and `user_config_update()`.

uint64_t serial_increment (uint64_t type, uint64_t num)

Increment the serial number for an object in memcached.

Parameters:

type the serial type to be queried (OBJECT_USER, OBJECT_CONFIG, OBJECT_FOLDERS, OBJECT_MESSAGES, or OBJECT_CONTACTS).
num the specific object identifier.

Returns:

0 on failure or the new serial number of the requested object.

Definition at line 88 of file serials.c.

References `cache_increment()`, `log_pedantic`, `serial_prefix()`, `st_aprint()`, `st_char_get()`, `st_free()`, and `st_length_int()`.

Referenced by `contact_move()`, `imap_append_message()`, `imap_close()`, `imap_create()`, `imap_delete()`, `imap_expunge()`, `imap_fetch()`, `imap_message_copier()`, `imap_rename()`, `imap_store()`, `mail_load_header()`, `mail_load_message()`, `meta_contacts_update()`, `meta_folders_update()`, `meta_message_folders_update()`, `meta_messages_login_update()`, `meta_messages_update()`, `meta_user_build()`, `meta_user_serial_check()`, `meta_user_update()`, `pop_session_destroy()`, `portal_endpoint_folders_rename()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `smtp_rollout()`, `smtp_store_message()`, and `user_config_edit()`.

uint64_t serial_reset (uint64_t type, uint64_t num)

Reset the serial number to 1 for an object in memcached.

Parameters:

type the serial type to be queried (OBJECT_USER, OBJECT_CONFIG, OBJECT_FOLDERS, OBJECT_MESSAGES, or OBJECT_CONTACTS).
num the specific object identifier.

Returns:

0 on failure or the new value of the cached serial number (1) on success.

Definition at line 112 of file serials.c.

References `cache_set()`, `CONSTANT`, `log_pedantic`, `serial_prefix()`, `st_aprint()`, `st_char_get()`, `st_free()`, and `st_length_int()`.

int_t user_lock (uint64_t *usernum*)

Acquire a lock in the magma.user keyspace.

Parameters:

usernum the numerical id of the user for whom the lock will be acquired.

Returns:

-1 on general failure, 0 on memcached failure, or 1 on success.

Definition at line 84 of file locks.c.

References `lock_get()`, `MANAGEDDBUF`, and `st_sprint()`.

Referenced by `imap_close()`, `imap_copy()`, `imap_expunge()`, `smtp_rollout()`, and `smtp_store_message()`.

void user_unlock (uint64_t *usernum*)

Unlock a lock in the magma.user keyspace.

Parameters:

usernum the numerical id of the user for whom the lock will be unlocked.

Returns:

This function returns no value.

Definition at line 100 of file locks.c.

References `lock_release()`, `MANAGEDDBUF`, and `st_sprint()`.

Referenced by `imap_close()`, `imap_copy()`, `imap_expunge()`, `smtp_rollout()`, and `smtp_store_message()`.

Variable Documentation

object_cache_t objects

Definition at line 15 of file objects.c.

Referenced by `meta_get()`, `meta_remove()`, `meta_user_prune()`, `sess_create()`, and `sess_get()`.

magma/objects/serials.c File Reference

Functions used for track and update the object checkpoints.

```
#include "magma.h"
```

Functions

- **stringer_t * serial_prefix** (uint64_t type)
- *Return a textual prefix describing an object type.* uint64_t **serial_get** (uint64_t type, uint64_t num)
- *Get the serial number (checkpoint value) for an object from memcached.* uint64_t **serial_increment** (uint64_t type, uint64_t num)
- *Increment the serial number for an object in memcached.* uint64_t **serial_reset** (uint64_t type, uint64_t num)

Reset the serial number to 1 for an object in memcached. Variables

- **stringer_t * serial_prefix_strings** []

Detailed Description

Functions used for track and update the object checkpoints.

Definition in file **serials.c**.

Function Documentation

uint64_t serial_get (uint64_t type, uint64_t num)

Get the serial number (checkpoint value) for an object from memcached.

serials.c

Parameters:

type the serial type to be queried (OBJECT_USER, OBJECT_CONFIG, OBJECT_FOLDERS, OBJECT_MESSAGES, or OBJECT_CONTACTS).

num the specific object identifier.

Returns:

0 on failure or the serial number of the requested object.

Definition at line 63 of file serials.c.

References `cache_increment()`, `log_pedantic`, `serial_prefix()`, `st_aprint()`, `st_char_get()`, `st_free()`, and `st_length_int()`.

Referenced by `imap_append_message()`, `imap_close()`, `imap_create()`, `imap_delete()`, `imap_expunge()`, `imap_fetch()`, `imap_message_copier()`, `imap_rename()`, `imap_session_update()`, `imap_store()`, `meta_contacts_update()`, `meta_folders_update()`, `meta_message_folders_update()`, `meta_messages_login_update()`, `meta_messages_update()`, `meta_user_build()`, `meta_user_serial_check()`, `meta_user_update()`, `portal_endpoint_folders_rename()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `user_config_create()`, and `user_config_update()`.

uint64_t serial_increment (uint64_t type, uint64_t num)

Increment the serial number for an object in memcached.

Parameters:

type the serial type to be queried (OBJECT_USER, OBJECT_CONFIG, OBJECT_FOLDERS, OBJECT_MESSAGES, or OBJECT_CONTACTS).

num the specific object identifier.

Returns:

0 on failure or the new serial number of the requested object.

Definition at line 88 of file serials.c.

References `cache_increment()`, `log_pedantic`, `serial_prefix()`, `st_aprint()`, `st_char_get()`, `st_free()`, and `st_length_int()`.

Referenced by `contact_move()`, `imap_append_message()`, `imap_close()`, `imap_create()`, `imap_delete()`, `imap_expunge()`, `imap_fetch()`, `imap_message_copier()`, `imap_rename()`, `imap_store()`, `mail_load_header()`, `mail_load_message()`, `meta_contacts_update()`, `meta_folders_update()`, `meta_message_folders_update()`, `meta_messages_login_update()`, `meta_messages_update()`, `meta_user_build()`, `meta_user_serial_check()`, `meta_user_update()`, `pop_session_destroy()`, `portal_endpoint_folders_rename()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `smtp_rollout()`, `smtp_store_message()`, and `user_config_edit()`.

stringer_t* serial_prefix (uint64_t type)

Return a textual prefix describing an object type.

Parameters:

type the serial object type: OBJECT_USER, OBJECT_CONFIG, OBJECT_FOLDERS, OBJECT_MESSAGES, or OBJECT_CONTACTS.

Returns:

a managed string containing a description of the object type, or NULL on failure.

Definition at line 28 of file serials.c.

References `log_pedantic`, `OBJECT_CONFIG`, `OBJECT_CONTACTS`, `OBJECT_FOLDERS`, `OBJECT_MESSAGES`, `OBJECT_USER`, and `serial_prefix_strings`.

Referenced by `serial_get()`, `serial_increment()`, and `serial_reset()`.

uint64_t serial_reset (uint64_t type, uint64_t num)

Reset the serial number to 1 for an object in memcached.

Parameters:

type the serial type to be queried (OBJECT_USER, OBJECT_CONFIG, OBJECT_FOLDERS, OBJECT_MESSAGES, or OBJECT_CONTACTS).

num the specific object identifier.

Returns:

0 on failure or the new value of the cached serial number (1) on success.

Definition at line 112 of file serials.c.

References `cache_set()`, `CONSTANT`, `log_pedantic`, `serial_prefix()`, `st_aprint()`, `st_char_get()`, `st_free()`, and `st_length_int()`.

Variable Documentation

stringer_t* serial_prefix_strings[]

```
Initial value: {  
    CONSTANT("user"),  
    CONSTANT("config"),  
    CONSTANT("folders"),  
    CONSTANT("messages"),  
}
```

Definition at line 15 of file `serials.c`.

Referenced by `serial_prefix()`.

magma/objects/sessions/sessions.c File Reference

Routines to handle web sessions.

```
#include "magma.h"
```

Functions

- `uint64_t sess_key` (void)
- *Generate a random 12 digit 64-bit web session key.* `uint64_t sess_number` (void)
- *Reserve a unique web session identifier.* `void sess_destroy (session_t *sess)`
- *Destroy a web session and its associated data.* `void sess_ref_add (session_t *sess)`
- *Increment the web session's reference counter and update its timestamp.* `void sess_ref_dec (session_t *sess)`
- *Decrement the web session's reference counter and update its timestamp.* `uint64_t sess_ref_total (session_t *sess)`
- *Get the reference count of a web session.* `time_t sess_ref_stamp (session_t *sess)`
- *Get the timestamp for a web session's reference counter.* `void sess_refresh_flush (session_t *sess)`
- *Update a web session's refresh stamp and prevent redundant refreshing of a web session.* `time_t sess_refresh_stamp (session_t *sess)`
- *Get the timestamp for the last time the session was refreshed.* `bool_t sess_refresh_check (session_t *sess)`
- *Check to see if a web session is ready to be refreshed, and if so, disarm its trigger and update its refresh stamp.* `stringer_t * sess_token (session_t *sess)`
- *Securely generate a unique zbase32-encoded token for a session.* `void sess_update (session_t *sess)`
- *Update a session's underlying data and its refresh timestamp.* `void sess_trigger (session_t *sess)`
- *Set a web session's trigger so it will be refreshed as soon as possible.* `void sess_release (session_t *sess)`
- *Release a web session.* `void sess_release_attachment (attachment_t *attachment)`
- *Free an attachment object.* `void sess_release_composition (composition_t *comp)`
- *Free a composition object.* `void sess_serial_check (session_t *sess, uint64_t object)`
- *Queue a web session refresh if there is stale data.* `session_t * sess_create (connection_t *con, stringer_t *path, stringer_t *application)`
- *Create a new session for a given web connection.* `int_t sess_get (connection_t *con, stringer_t *application, stringer_t *path, stringer_t *token)`

Try to retrieve the session associated with a client connection's supplied cookie.

Variables

- `struct {`
- `uint64_t number`
- `pthread_mutex_t lock`
- `} sessions`

Detailed Description

Routines to handle web sessions.

Definition in file `sessions.c`.

Function Documentation

session_t* sess_create (connection_t * con, stringer_t * path, stringer_t * application)

Create a new session for a given web connection.

sessions.c

Note:

The session stores the following data points: remote IP address, request path, application name, the specified http hostname, the remote client's user agent string, the server's host number, a unique session id, the server's current timestamp, a randomly-generated session key for authentication, and an encrypted token for the session returned to the user as a cookie.

Parameters:

con a pointer to the connection underlying the web session.

path a pointer to a managed string containing the pathname of the generating request (should be "/portal/camel").

application a pointer to a managed string containing the name of the parent application of the session (should be "portal").

Returns:

NULL on failure or a pointer to a newly allocated session object for the specified connection.

Definition at line 384 of file sessions.c.

References session_t::agent, session_t::application, session_t::compositions, con_addr(), con_secure(), CONTIGUOUS, HEAP, magma_t::host, session_t::host, connection_t::http, session_t::httponly, inx_alloc(), inx_insert(), session_t::ip, ip_copy(), session_t::key, session_t::lock, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, magma, MANAGED_T, MEMORYBUF, mm_alloc(), mm_free(), session_t::number, magma_t::number, objects, session_t::path, session_t::request, session_t::secure, sess_destroy(), sess_key(), sess_number(), sess_ref_add(), sess_ref_dec(), sess_release_composition(), sess_token(), object_cache_t::sessions, st_dupe_opts(), session_t::stamp, session_t::token, multi_t::u64, multi_t::val, and session_t::warden.

Referenced by portal_endpoint().

void sess_destroy (session_t * sess)

Destroy a web session and its associated data.

Parameters:

sess a pointer to the web session to be destroyed.

Returns:

This function returns no value.

Definition at line 64 of file sessions.c.

References session_t::agent, session_t::application, session_t::compositions, session_t::cred, credential_free(), session_t::host, inx_cleanup(), session_t::lock, META_PROT_WEB, meta_remove(), mm_free(), mutex_destroy(), session_t::path, session_t::request, st_cleanup(), session_t::token, session_t::user, meta_user_t::username, and session_t::warden.

Referenced by obj_cache_start(), and sess_create().

int_t sess_get (connection_t * con, stringer_t * application, stringer_t * path, stringer_t * token)

Try to retrieve the session associated with a client connection's supplied cookie.

Parameters:

con a pointer to the connection object sending the cookie.

application a managed string containing the application associated with the session.

path a managed string containing the path associated with the session.

token the encrypted user token retrieved from the supplied http cookie.

Returns:

1 if the cookie was found and valid, or one of the following values on failure: 0 = Session not found. -1 = Server error. -2 = Invalid token. -3 = Security violation / incorrect user-agent. -4 = Security violation / incorrect session key. -5 = Security violation / incorrect source address. -6 = Session terminated by logout. -7 = Session timed out.

Most session attributes need simple equality comparison, except for timeout checking. Make sure not to validate against a stale session that should have already timed out (which will have to be determined dynamically).

Definition at line 446 of file sessions.c.

References `con_addr()`, `con_secure()`, `magma_t::http`, `connection_t::http`, `inx_delete()`, `inx_find()`, `inx_lock_read()`, `inx_unlock()`, `ip_address_equal()`, `log_error`, `log_pedantic`, `M_TYPE_UINT64`, `magma`, `MEMORYBUF`, `objects`, `scramble_decrypt()`, `scramble_import()`, `magma_t::secure`, `sess_ref_add()`, `sess_ref_dec()`, `sess_refresh_stamp()`, `sess_update()`, `object_cache_t::sessions`, `magma_t::sessions`, `st_char_get()`, `st_cleanup()`, `st_cmp_cs_eq()`, `st_data_get()`, `st_free()`, `multi_t::u64`, `multi_t::val`, and `zbase32_decode()`.

Referenced by `http_parse_context()`.

uint64_t sess_key (void)

Generate a random 12 digit 64-bit web session key.

Returns:

the randomly generated web session key as an unsigned 64 bit integer.

Definition at line 27 of file sessions.c.

References `log_pedantic`, `rand_get_uint64()`, and `uint64_digits()`.

Referenced by `sess_create()`.

uint64_t sess_number (void)

Reserve a unique web session identifier.

Returns:

a number containing a unique web session identifier.

Definition at line 48 of file sessions.c.

References mutex_lock(), mutex_unlock(), and sessions.

Referenced by sess_create().

void sess_ref_add (session_t * sess)

Increment the web session's reference counter and update its timestamp.

Parameters:

sess a pointer to the web session to be updated.

Returns:

This function returns no value.

Definition at line 98 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refs, session_t::stamp, and session_t::web.

Referenced by sess_create(), and sess_get().

void sess_ref_dec (session_t * sess)

Decrement the web session's reference counter and update its timestamp.

Parameters:

sess a pointer to the web session to be updated.

Returns:

This function returns no value.

Definition at line 123 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refs, session_t::stamp, and session_t::web.

Referenced by sess_create(), sess_get(), and sess_release().

time_t sess_ref_stamp (session_t * sess)

Get the timestamp for a web session's reference counter.

Parameters:

sess a pointer to the web session to be queried.

Returns:

the last-modified UTC timestamp value of the specified web session.

Definition at line 173 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refs, and session_t::stamp.

Referenced by obj_cache_prune().

uint64_t sess_ref_total (session_t * sess)

Get the reference count of a web session.

Parameters:

sess a pointer to the web session to be queried.

Returns:

the total number of references to the specified web session.

Definition at line 148 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refs, and session_t::web.

Referenced by obj_cache_prune().

bool_t sess_refresh_check (session_t * sess)

Check to see if a web session is ready to be refreshed, and if so, disarm its trigger and update its refresh stamp.

Note:

The caller needs to make immediate use of this function's return value because the refresh trigger will be cleared if it was previously set. Any session longer than 2 minutes will be marked as ready for refresh.

Parameters:

a pointer to the web session to be queried.

Returns:

true if the web session is ready to be refreshed, or false if it is not.

Definition at line 228 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refresh, session_t::stamp, and session_t::trigger.

Referenced by sess_release().

void sess_refresh_flush (session_t * sess)

Update a web session's refresh stamp and prevent redundant refreshing of a web session.

Parameters:

sess a pointer to the web session to be touched.

Returns:

This function returns no value.

Definition at line 191 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refresh, session_t::stamp, and session_t::trigger.

Referenced by sess_update().

time_t sess_refresh_stamp (session_t * sess)

Get the timestamp for the last time the session was refreshed.

Parameters:

sess a pointer to the web session to be queried.

Returns:

the last-refreshed UTC timestamp value of the specified web session.

Definition at line 208 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refresh, and session_t::stamp.

Referenced by sess_get().

void sess_release (session_t * sess)

Release a web session.

Note:

If the web session should have been refreshed, it will be refreshed before the reference counter is decremented.

Parameters:

sess a pointer to the web session to be released.

Returns:

This function returns no value.

Definition at line 315 of file sessions.c.

References requeue(), sess_ref_dec(), sess_refresh_check(), and sess_update().

Referenced by http_session_reset().

void sess_release_attachment (attachment_t * attachment)

Free an attachment object.

Note:

This is an inx helper function.

Parameters:

attachment a pointer to the attachment object to be destroyed.

Returns:

This function returns no value.

Definition at line 334 of file sessions.c.

References attachment_t::filedata, attachment_t::filename, mm_free(), and st_cleanup().

Referenced by portal_endpoint_attachments_add(), and portal_endpoint_messages_compose().

void sess_release_composition (composition_t * comp)

Free a composition object.

Note:

This is an inx helper function.

Parameters:

comp a pointer to the composition object to be destroyed.

Returns:

This function returns no value.

Definition at line 350 of file sessions.c.

References composition_t::attachments, inx_cleanup(), and mm_free().

Referenced by portal_endpoint_messages_compose(), and sess_create().

void sess_serial_check (session_t * sess, uint64_t object)

Queue a web session refresh if there is stale data.

Parameters:

sess a pointer to the web session to be queried.

object the value of the object in the cache to be checked (can include OBJECT_CONTACTS and OBJECT_FOLDERS).

Definition at line 365 of file sessions.c.

References meta_user_serial_check(), sess_trigger(), and session_t::user.

Referenced by portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(), portal_endpoint_contacts_move(), portal_endpoint_contacts_remove(), portal_folder_contacts_add(), portal_folder_contacts_remove(), portal_folder_mail_add(), and portal_folder_mail_remove().

stringer_t* sess_token (session_t * sess)

Securely generate a unique zbase32-encoded token for a session.

Parameters:

sess a pointer to the input web session.

Returns:

a managed string containing the generated web session token.

Definition at line 251 of file sessions.c.

References session_t::host, session_t::key, log_pedantic, magma, session_t::number, PLACER, scramble_encrypt(), scramble_free(), scramble_total_length(), magma_t::secure, magma_t::sessions, st_cleanup(), st_merge, session_t::stamp, session_t::warden, and zbase32_encode().

Referenced by sess_create().

void sess_trigger (session_t * sess)

Set a web session's trigger so it will be refreshed as soon as possible.

Parameters:

sess a pointer to the web session to be refreshed.

Returns:

This function returns no value.

Definition at line 298 of file sessions.c.

References session_t::lock, mutex_lock(), mutex_unlock(), session_t::refresh, and session_t::trigger.

Referenced by portal_endpoint_folders_rename(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(), and sess_serial_check().

void sess_update (session_t * sess)

Update a session's underlying data and its refresh timestamp.

Note:

This function ensures the session's associated meta user data, mail folders and messages, and contact folders and contact entries are all current.

Parameters:

sess a pointer to the session object to be updated.

Returns:

This function returns no value.

Definition at line 278 of file sessions.c.

References meta_contacts_update(), meta_folders_update(), meta_messages_update(), META_NEED_LOCK, meta_user_update(), sess_refresh_flush(), and session_t::user.

Referenced by sess_get(), and sess_release().

Variable Documentation**pthread_mutex_t lock**

Definition at line 17 of file sessions.c.

uint64_t number

Definition at line 16 of file sessions.c.

Referenced by ecies_key_private(), hex_encode_chr(), imap_build_array(), imap_fetch_body(), imap_fetch_body_header(), imap_fetch_body_part(), imap_fetch_body_tag(), imap_fetch_parse_partial(), imap_flag_parse(), imap_narrow_messages(), imap_parse_dataitems(), imap_parse_literal(), imap_search_messages_inner(), imap_search_messages_range(), imap_status(), pop_dele(), pop_last(), pop_list(), pop_retr(), pop_top(), pop_uidl(), smtp_bounce(), and smtp_check_receive_quota().

struct { ... } sessions

Referenced by sess_number().

magma/servers/http/sessions.c File Reference

HTTP session handlers.

```
#include "magma.h"
```

Functions

- void **http_session_reset** (**connection_t** *con)
- *Reset a client http connection to its original, uninitialized state.* void **http_session_destroy** (**connection_t** *con)

Destroy an http client connection.

Detailed Description

HTTP session handlers.

Definition in file **sessions.c**.

Function Documentation

void http_session_destroy (connection_t * con)

Destroy an http client connection.

sessions.c

Parameters:

con the connection object to have its http session destroyed.

Returns:

This function returns no value.

Definition at line 82 of file sessions.c.

References [http_session_reset\(\)](#).

Referenced by [con_destroy\(\)](#).

void http_session_reset (connection_t * con)

Reset a client http connection to its original, uninitialized state.

Parameters:

con the connection object to have its http session state reset.

Returns:

This function returns no value.

Definition at line 20 of file sessions.c.

References [connection_t::http](#), [HTTP_CLOSE](#), [HTTP_CONNECTION_CLOSE](#), [HTTP_CONNECTION_NEUTRAL](#), [HTTP_MERGED](#), [HTTP_METHOD_NONE](#), [HTTP_PORTAL](#), [HTTP_READY](#), [inx_cleanup\(\)](#), [json_decref_d](#), [sess_release\(\)](#), and [st_cleanup\(\)](#).

Referenced by `http_requeue()`, and `http_session_destroy()`.

magma/servers/imap/sessions.c File Reference

IMAP session handlers.

```
#include "magma.h"
```

Functions

- **int_t imap_session_update** (connection_t *con)
- **void imap_session_destroy** (connection_t *con)

sessions.c

Detailed Description

IMAP session handlers.

Definition in file **sessions.c**.

Function Documentation

void imap_session_destroy (connection_t * con)

sessions.c

Definition at line 98 of file sessions.c.

References `ar_free()`, `meta_message_t::foldernum`, `connection_t::imap`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `mail_cache_reset()`, `MAIL_STATUS_RECENT`, `META_PROT_IMAP`, `meta_remove()`, `meta_user_unlock()`, `meta_user_wlock()`, `st_cleanup()`, and `meta_message_t::status`.

Referenced by `con_destroy()`.

int_t imap_session_update (connection_t * con)

Returns 0 if the selected folder wasn't modified, or 1 if things changed and the updated status should be sent to the client. A -1 is used to indicate the update check encountered a problem and should be retried later.

Definition at line 19 of file sessions.c.

References `meta_message_t::foldernum`, `connection_t::imap`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_RECENT`, `meta_folders_update()`, `META_LOCKED`, `meta_messages_update()`, `meta_user_unlock()`, `meta_user_update()`, `meta_user_wlock()`, `OBJECT_FOLDERS`, `OBJECT_MESSAGES`, `OBJECT_USER`, `serial_get()`, and `meta_message_t::status`.

Referenced by `imap_check()`, `imap_expunge()`, `imap_noop()`, and `imap_store()`.

magma/servers/molten/sessions.c File Reference

Molten session handlers.

```
#include "magma.h"
```

Functions

- void **molten_session_destroy** (connection_t *con)
sessions.c
-

Detailed Description

Molten session handlers.

Definition in file **sessions.c**.

Function Documentation

void molten_session_destroy (connection_t * con)

sessions.c

Definition at line 15 of file sessions.c.

Referenced by con_destroy().

magma/servers/pop/sessions.c File Reference

Functions for managing POP3 sessions.

```
#include "magma.h"
```

Functions

- **int_t pop_session_reset (connection_t *con)**
- *Reset a POP3 session by guaranteeing that no messages are flagged to be deleted.* void **pop_session_destroy (connection_t *con)**

Destroy a POP3 session and delete any flagged messages belonging to the user.

Detailed Description

Functions for managing POP3 sessions.

Definition in file `sessions.c`.

Function Documentation

void pop_session_destroy (connection_t * con)

Destroy a POP3 session and delete any flagged messages belonging to the user.

`sessions.c`

Parameters:

con the POP3 client connection issuing the command. This function returns no value.

Definition at line 55 of file `sessions.c`.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_delete()`, `M_TYPE_UINT64`, `mail_cache_reset()`, `mail_remove_message()`, `MAIL_STATUS_HIDDEN`, `meta_message_t::messagenum`, `meta_messages_update_sequences()`, `META_PROT_POP`, `meta_remove()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_MESSAGES`, `connection_t::pop`, `serial_increment()`, `meta_message_t::server`, `meta_message_t::size`, `st_cleanup()`, `meta_message_t::status`, `multi_t::u64`, and `multi_t::val`.

Referenced by `con_destroy()`.

int_t pop_session_reset (connection_t * con)

Reset a POP3 session by guaranteeing that no messages are flagged to be deleted.

Parameters:

con the POP3 client connection issuing the command.

Returns:

-1 on general error, 0 if the connection hasn't been authenticated, or 1 if all messages have had their statuses successfully reset.

Definition at line 20 of file `sessions.c`.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_HIDDEN`, `meta_user_unlock()`, `meta_user_wlock()`, `connection_t::pop`, and `meta_message_t::status`.

Referenced by `pop_rset()`, and `pop_starttls()`.

magma/web/register/sessions.c File Reference

Functions for handling the internal session structure used by new user registration attempts.

```
#include "magma.h"
```

Functions

- void **register_session_free** (**register_session_t** *session)
- *Destroy a registration session and all its associated data.* **register_session_t** * **register_session_generate** (void)
- *Generate a new registration session.* **register_session_t** * **register_session_get** (**connection_t** *con, **stringer_t** *name)
- *Retrieve a registration session by a data key supplied in the user's http POST request.* **bool_t** **register_session_cache** (**connection_t** *con, **register_session_t** *session)

Save a registration session's state into the cache.

Detailed Description

Functions for handling the internal session structure used by new user registration attempts.

Definition in file **sessions.c**.

Function Documentation

bool_t register_session_cache (connection_t * con, register_session_t * session)

Save a registration session's state into the cache.

sessions.c

Note:

The session is stored under the parent key "lavad.register.session."

Parameters:

con a pointer to the client connection underlying the user session.

session a pointer to the registration session to be persisted.

Returns:

false on failure or true if the session was cached successfully.

Definition at line 111 of file sessions.c.

References `cache_set()`, `con_addr_presentation()`, `data`, `register_session_t::hvf_input`, `register_session_t::hvf_value`, `log_pedantic`, `MANAGEDBUF`, `register_session_t::name`, `NULLER`, `register_session_t::password`, `register_session_t::plan`, `serialize_st()`, `serialize_uint16()`, `serialize_uint64()`, `st_char_get()`, `st_cleanup()`, `st_free()`, `st_length_int()`, `register_session_t::username`, and `register_session_t::usernum`.

Referenced by `register_process()`.

void register_session_free (register_session_t * session)

Destroy a registration session and all its associated data.

Returns:

This function returns no value.

Definition at line 19 of file sessions.c.

References `register_session_t::hvf_input`, `register_session_t::hvf_value`, `mm_free()`, `register_session_t::name`, `register_session_t::password`, `st_cleanup()`, and `register_session_t::username`.

Referenced by `register_process()`, `register_session_generate()`, and `register_session_get()`.

register_session_t* register_session_generate (void)

Generate a new registration session.

Returns:

NULL on failure, or a pointer to a new randomly named session on success.

Definition at line 40 of file sessions.c.

References `log_pedantic`, `mm_alloc()`, `register_session_t::name`, `rand_choices()`, and `register_session_free()`.

Referenced by `register_process()`.

register_session_t* register_session_get (connection_t * con, stringer_t * name)

Retrieve a registration session by a data key supplied in the user's http POST request.

Note:

The session is stored under the parent key "lavad.register.session."

Parameters:

con the client connection underlying the user request.

name a managed string containing the registration session identifier.

Returns:

NULL on failure, or a pointer to the user's registration session on success.

Definition at line 64 of file sessions.c.

References `cache_get()`, `con_addr_presentation()`, `serialization_t::data`, `data`, `deserialize_st()`, `deserialize_uint16()`, `deserialize_uint64()`, `register_session_t::hvf_input`, `register_session_t::hvf_value`, `log_pedantic`, `MANAGEDBUF`, `mm_alloc()`, `mm_wipe()`, `register_session_t::name`, `NULLER`, `register_session_t::password`, `register_session_t::plan`, `register_session_free()`, `st_char_get()`, `st_dupe()`, `st_free()`, `st_length_int()`, `register_session_t::username`, and `register_session_t::usernum`.

Referenced by `register_process()`.

magma/objects/users/alerts.c File Reference

Functions for handling user alerts and warnings.

```
#include "magma.h"
```

Functions

- `meta_alert_t * alert_alloc (uint64_t alertnum, stringer_t *type, stringer_t *message, uint64_t created)`
Allocate and initialize a new user alert message object.
-

Detailed Description

Functions for handling user alerts and warnings.

Definition in file `alerts.c`.

Function Documentation

`meta_alert_t * alert_alloc (uint64_t alertnum, stringer_t * type, stringer_t * message, uint64_t created)`

Allocate and initialize a new user alert message object.

`alerts.c`

Parameters:

alertnum the numerical id of the user alert.

type a pointer to a managed string containing the type of the alert (e.g. "warning" or "alert").

message a pointer to a managed string containing the alert message.

created the UTC timecode for when the alert message was created.

Returns:

NULL on error or a pointer to the newly initialized user alert message object on success.

Definition at line 23 of file `alerts.c`.

References `align()`, `FOREIGNDATA`, `JOINTED`, `log_pedantic`, `mm_alloc()`, `mm_copy()`, `PLACER_T`, `placer_t`, `st_data_get()`, `st_length_get()`, and `STACK`.

Referenced by `meta_data_fetch_alerts()`.

magma/objects/users/aliases.c File Reference

Functions for handling user aliases.

```
#include "magma.h"
```

Functions

- `meta_alias_t * alias_alloc (uint64_t aliasnum, stringer_t *address, stringer_t *display, int_t selected, uint64_t created)`

Allocate and initialize a new user alias object.

Detailed Description

Functions for handling user aliases.

Definition in file `aliases.c`.

Function Documentation

`meta_alias_t* alias_alloc (uint64_t aliasnum, stringer_t * address, stringer_t * display, int_t selected, uint64_t created)`

Allocate and initialize a new user alias object.

aliases.c

Parameters:

aliasnum the numerical identifier of the user alias.

address a managed string containing the email address associated with the alias.

display a managed string containing the display name associated with the alias.

selected a flag specifying whether this alias is the default selection for the user.

created a UNIX timestamp for the creation of this alias.

Returns:

NULL on failure, or a pointer to the newly initialized user alias object on success.

Definition at line 24 of file `aliases.c`.

References `align()`, `FOREIGNDATA`, `JOINTED`, `log_pedantic`, `mm_alloc()`, `mm_copy()`, `PLACER_T`, `placer_t`, `st_data_get()`, `st_length_get()`, and `STACK`.

Referenced by `meta_data_fetch_mailbox_aliases()`.

magma/objects/users/users.c File Reference

Functions for handling the user context.

```
#include "magma.h"
```

Functions

- void **meta_user_ref_add** (**meta_user_t** *user, **META_PROT** protocol)
- *Increment a meta user's reference counter for a specified protocol and update the activity timestamp.* void **meta_user_ref_dec** (**meta_user_t** *user, **META_PROT** protocol)
- *Decrement a user's reference counter for a specified protocol and update the activity timestamp.* uint64_t **meta_user_ref_total** (**meta_user_t** *user)
- *Get the total session reference count for a user for all protocols.* time_t **meta_user_ref_stamp** (**meta_user_t** *user)
- *Get the activity timestamp for a meta user object.* void **meta_user_rlock** (**meta_user_t** *user)
- *Acquire a read lock for a meta user object.* void **meta_user_wlock** (**meta_user_t** *user)
- *Acquire a write lock for a meta user object.* void **meta_user_unlock** (**meta_user_t** *user)
- *Release the lock for a meta user object.* void **meta_user_destroy** (**meta_user_t** *user)
- *Destroy a meta user object and free all of its underlying data.* void **meta_remove** (**stringer_t** *username, **META_PROT** flags)
- *Lock a user's object in the cache and decrement their reference counter.* int_t **meta_user_prune** (**stringer_t** *username)
- *Locate and remove a user's object from the cache if the reference counter is zero.* **meta_user_t** * **meta_user_create** (void)
- *Allocate and initialize a meta user object.* int_t **meta_user_update** (**meta_user_t** *user, **META_LOCK_STATUS** locked)
- *Update a meta user object from the cache.* int_t **meta_user_build** (**meta_user_t** *user, **stringer_t** *username, **stringer_t** *passhash, **stringer_t** *passkey, **META_LOCK_STATUS** locked)
- *Build a user's meta information from specified data parameters.* void **meta_user_serial_set** (**meta_user_t** *user, uint64_t object, uint64_t serial)
- *Set an object serial number for a given meta user structure.* uint64_t **meta_user_serial_get** (**meta_user_t** *user, uint64_t object)
- *Get an object serial number for a given meta user structure.* bool_t **meta_user_serial_check** (**meta_user_t** *user, uint64_t object)
- *Check an object's serial number to see if it is up-to-date.* int_t **meta_get** (**stringer_t** *username, **stringer_t** *mbxdomain, **stringer_t** *passhash, **stringer_t** *passkey, **META_PROT** flags, **META_GET** get, **meta_user_t** **output)

Lookup user by information and and return a meta user object.

Detailed Description

Functions for handling the user context.

\$Author:\$ \$Author\$ \$Revision\$

Definition in file **users.c**.

Function Documentation

```
int_t meta_get (stringer_t * username,  stringer_t * mbxdomain,  stringer_t * passhash,
stringer_t * passkey,  META_PROT flags,  META_GET get,  meta_user_t ** output)
```


Lookup user by information and return a meta user object.

users.c

Note:

If the user is not found, one will be created.

Parameters:

username the username to be looked up.

mboxdomain if not NULL, an optional managed string containing the domain suffix of a mailbox address to be verified first.

passhash a managed string with a multi-round hash of the user's password.

passkey a managed string with a single-round hash of the user's password.

flags a set of flags specifying the protocol used by the calling function. Values can be META_PROT_NONE, META_PROT_SMTP, META_PROT_POP, META_PROT_IMAP, META_PROT_WEB, or META_PROT_GENERIC.

get a set of flags specifying the data to be retrieved (META_GET_NONE, META_GET_MESSAGES, META_GET_FOLDERS, or META_GET_CONTACTS)

output the address of a meta user object that will store a pointer to the result of the lookup.

Returns:

-1 on error, 0 if the username information exists but there was an error, and 1 on success.

Definition at line 533 of file users.c.

References `inx_find()`, `inx_insert()`, `inx_lock_write()`, `inx_unlock()`, `log_error`, `log_pedantic`, `M_TYPE_STRINGER`, `meta_contacts_update()`, `meta_data_check_mailbox()`, `meta_folders_update()`, `META_GET_CONTACTS`, `META_GET_FOLDERS`, `META_GET_MESSAGES`, `META_LOCKED`, `meta_message_folders_update()`, `meta_messages_update()`, `meta_remove()`, `meta_user_build()`, `meta_user_create()`, `meta_user_destroy()`, `meta_user_ref_add()`, `meta_user_unlock()`, `meta_user_wlock()`, `objects`, `meta_user_t::passhash`, `st_cleanup()`, `st_cmp_cs_eq()`, `st_empty()`, `st_free()`, `st_merge`, and `object_cache_t::users`.

Referenced by `imap_login()`, `pop_pass()`, and `portal_endpoint_auth()`.

void meta_remove (stringer_t * username, META_PROT flags)

Lock a user's object in the cache and decrement their reference counter.

See also:

`meta_user_ref_dec()`

Parameters:

username a managed string containing the name of the user to be adjusted.

flags specifies the protocol bound to the reference counter to be decremented (META_PROT_WEB, META_PROT_IMAP, etc.)

Returns:

This function returns no value.

Definition at line 220 of file users.c.

References `inx_find()`, `inx_lock_read()`, `inx_unlock()`, `M_TYPE_STRINGER`, `meta_user_ref_dec()`, `objects`, `st_empty()`, and `object_cache_t::users`.

Referenced by `imap_login()`, `imap_session_destroy()`, `meta_get()`, `pop_pass()`, `pop_session_destroy()`, `portal_endpoint_auth()`, and `sess_destroy()`.

int_t meta_user_build (meta_user_t * *user*, stringer_t * *username*, stringer_t * *passhash*, stringer_t * *passkey*, META_LOCK_STATUS *locked*)

Build a user's meta information from specified data parameters.

Note:

The user object will be pulled from the cache, if possible, or falls back to a database lookup by username+password.

Parameters:

user a pointer to the meta user object of the user that is to be filled.

passhash a managed string with a multi-round hash of the user's password.

passkey a managed string with a single-round hash of the user's password.

locked the meta lock status of the operation (if META_NEED_LOCK is supplied, the meta user object will be locked for the duration of the function).

Returns:

true on success or false on failure.

Definition at line 386 of file users.c.

References meta_data_fetch_mailbox_aliases(), meta_data_fetch_user(), meta_data_user_build(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), OBJECT_USER, meta_user_t::passhash, serial_get(), serial_increment(), meta_user_t::serials, st_cleanup(), st_dupe(), st_empty(), meta_user_t::user, meta_user_t::username, and meta_user_t::usernum.

Referenced by meta_get().

meta_user_t* meta_user_create (void)

Allocate and initialize a meta user object.

Returns:

NULL on failure, or a pointer to the newly allocated meta user object on success.

Definition at line 290 of file users.c.

References meta_user_t::lock, log_pedantic, mm_alloc(), mm_free(), mutex_destroy(), mutex_init(), and meta_user_t::refs.

Referenced by meta_get().

void meta_user_destroy (meta_user_t * *user*)

Destroy a meta user object and free all of its underlying data.

Parameters:

user a pointer to the meta user object to be destroyed.

Returns:

This function returns no value.

Definition at line 185 of file users.c.

References meta_user_t::ads, meta_user_t::aliases, meta_user_t::contacts, meta_user_t::folders, inx_cleanup(), meta_user_t::lock, meta_user_t::message_folders, meta_user_t::messages, mm_free(), mutex_destroy(),

meta_user_t::passhash, meta_user_t::refs, st_cleanup(), meta_user_t::storage_privkey,
meta_user_t::storage_pubkey, and meta_user_t::username.

Referenced by meta_get(), and obj_cache_start().

int_t meta_user_prune (stringer_t * username)

Locate and remove a user's object from the cache if the reference counter is zero.

Parameters:

username a managed string containing the name of the user whose data should be removed from the object cache.

Returns:

returns 1 if the user object was successfully pruned, 0 if the user isn't found, a non-zero reference count causes a return of -1, and a value of -2 is returned if an error occurs while trying to delete the user from the object cache index.

Definition at line 248 of file users.c.

References count, inx_count(), inx_delete(), inx_find(), inx_lock_write(), inx_unlock(), log_error, M_TYPE_STRINGER, meta_user_ref_total(), objects, st_empty(), stats_adjust_by_name(), stats_set_by_name(), and object_cache_t::users.

void meta_user_ref_add (meta_user_t * user, META_PROT protocol)

Increment a meta user's reference counter for a specified protocol and update the activity timestamp.

Note:

META_PROT_GENERIC can be specified for protocol non-specific accounting purposes.

Parameters:

user a pointer to the meta user object to be adjusted.

protocol the protocol identifier for the session.

Returns:

This function returns no value.

Definition at line 22 of file users.c.

References meta_user_t::generic, meta_user_t::imap, meta_user_t::lock, META_PROT_GENERIC, META_PROT_IMAP, META_PROT_POP, META_PROT_SMTP, META_PROT_WEB, mutex_lock(), mutex_unlock(), meta_user_t::pop, meta_user_t::refs, meta_user_t::smtp, meta_user_t::stamp, and meta_user_t::web.

Referenced by meta_check_message_encryption(), meta_get(), and portal_endpoint_auth().

void meta_user_ref_dec (meta_user_t * user, META_PROT protocol)

Decrement a user's reference counter for a specified protocol and update the activity timestamp.

Note:

META_PROT_GENERIC can be specified for protocol non-specific accounting purposes.

Parameters:

user a pointer to the meta user object to be adjusted.
protocol the protocol identifier for the session (META_PROT_WEB, META_PROT_IMAP, META_PROT_POP, META_PROT_SMTP, META_PROT_GENERIC).

Returns:

This function returns no value.

Definition at line 54 of file users.c.

References meta_user_t::generic, meta_user_t::imap, meta_user_t::lock, META_PROT_GENERIC, META_PROT_IMAP, META_PROT_POP, META_PROT_SMTP, META_PROT_WEB, mutex_lock(), mutex_unlock(), meta_user_t::pop, meta_user_t::refs, meta_user_t::smtp, meta_user_t::stamp, and meta_user_t::web.

Referenced by decrypt_user_messages(), encrypt_user_messages(), meta_remove(), and portal_endpoint_logout().

time_t meta_user_ref_stamp (meta_user_t * user)

Get the activity timestamp for a meta user object.

Parameters:

user a pointer to the meta user object to be examined.

Returns:

a timestamp containing the last time the meta user object's reference count changed.

Definition at line 109 of file users.c.

References meta_user_t::lock, mutex_lock(), mutex_unlock(), meta_user_t::refs, and meta_user_t::stamp.

Referenced by obj_cache_prune().

uint64_t meta_user_ref_total (meta_user_t * user)

Get the total session reference count for a user for all protocols.

Parameters:

user a pointer to the meta user object to be examined.

Returns:

the total number of references held by the user.

Definition at line 84 of file users.c.

References meta_user_t::generic, meta_user_t::imap, meta_user_t::lock, mutex_lock(), mutex_unlock(), meta_user_t::pop, meta_user_t::refs, meta_user_t::smtp, and meta_user_t::web.

Referenced by meta_user_prune(), and obj_cache_prune().

void meta_user_rlock (meta_user_t * user)

Acquire a read lock for a meta user object.

Parameters:

user a pointer to the meta user object to be locked.

Returns:

This function returns no value.

Definition at line 134 of file users.c.

References meta_user_t::lock, and mutex_lock().

Referenced by imap_close(), imap_examine(), imap_expunge(), imap_fetch(), imap_list(), imap_login(), imap_lsub(), imap_search_messages(), imap_status(), pop_last(), pop_list(), pop_retr(), pop_stat(), pop_top(), pop_uidl(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(), portal_endpoint_folders_tags(), portal_endpoint_messages_flag(), and portal_endpoint_messages_tag().

bool_t meta_user_serial_check (meta_user_t * *user*, uint64_t *object*)

Check an object's serial number to see if it is up-to-date.

Note:

The object's serial number will be incremented regardless of whether it is consistent with the cache.

Parameters:

user the meta user object to whom the object belongs.

object the serial object to be checked for changes.

Returns:

0 if the object did not to be refreshed, or 1 if it doesn't match the internal checkpoint and should be updated.

Definition at line 502 of file users.c.

References meta_user_serial_get(), meta_user_serial_set(), serial_get(), serial_increment(), and meta_user_t::usernum.

Referenced by portal_endpoint_messages_tag(), and sess_serial_check().

uint64_t meta_user_serial_get (meta_user_t * *user*, uint64_t *object*)

Get an object serial number for a given meta user structure.

Parameters:

user the meta user structure to be examined.

object the object to query for the serial number.

Returns:

the serial number value of the specified object, or 0 on failure.

Definition at line 472 of file users.c.

References meta_user_t::contacts, meta_user_t::folders, meta_user_t::messages, OBJECT_CONTACTS, OBJECT_FOLDERS, OBJECT_MESSAGES, OBJECT_USER, meta_user_t::serials, and meta_user_t::user.

Referenced by meta_user_serial_check().

void meta_user_serial_set (meta_user_t * *user*, uint64_t *object*, uint64_t *serial*)

Set an object serial number for a given meta user structure.

Parameters:

user the meta user object to be adjusted.
object the object to receive the new serial number.
serial the new serial number to be set.

Returns:

This function returns no value.

Definition at line 445 of file users.c.

References meta_user_t::contacts, meta_user_t::folders, meta_user_t::messages, OBJECT_CONTACTS, OBJECT_FOLDERS, OBJECT_MESSAGES, OBJECT_USER, meta_user_t::serials, and meta_user_t::user.

Referenced by meta_user_serial_check().

void meta_user_unlock (meta_user_t * user)

Release the lock for a meta user object.

Parameters:

user a pointer to the meta user object to be unlocked.

Returns:

This function returns no value.

Definition at line 168 of file users.c.

References meta_user_t::lock, and mutex_unlock().

Referenced by imap_append(), imap_close(), imap_copy(), imap_create(), imap_delete(), imap_examine(), imap_expunge(), imap_fetch(), imap_list(), imap_login(), imap_lsub(), imap_rename(), imap_search_messages(), imap_select(), imap_session_destroy(), imap_session_update(), imap_status(), imap_store(), imap_subscribe(), meta_contacts_update(), meta_folders_update(), meta_get(), meta_message_folders_update(), meta_messages_copier(), meta_messages_login_update(), meta_messages_mover(), meta_messages_update(), meta_user_build(), meta_user_update(), pop_dele(), pop_last(), pop_list(), pop_pass(), pop_retr(), pop_session_destroy(), pop_session_reset(), pop_stat(), pop_top(), pop_uidl(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(), portal_endpoint_contacts_move(), portal_endpoint_contacts_remove(), portal_endpoint_folders_rename(), portal_endpoint_folders_tags(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(), portal_endpoint_messages_tag(), portal_folder_contacts_add(), portal_folder_contacts_remove(), portal_folder_mail_add(), and portal_folder_mail_remove().

int_t meta_user_update (meta_user_t * user, META_LOCK_STATUS locked)

Update a meta user object from the cache.

Parameters:

user a pointer to the meta user object that should be updated.
locked if META_NEED_LOCK is specified, a writer's lock will be acquired for the meta user object.

Returns:

-1 on general failure, 0 if the user or its mailbox aliases could not be fetched, or 1 on success.

Definition at line 339 of file users.c.

References meta_data_fetch_mailbox_aliases(), meta_data_fetch_user(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), OBJECT_USER, serial_get(), serial_increment(), meta_user_t::serials, meta_user_t::user, and meta_user_t::usernum.

Referenced by imap_session_update(), and sess_update().

void meta_user_wlock (meta_user_t * user)

Acquire a write lock for a meta user object.

Parameters:

user a pointer to the meta user object to be locked.

Returns:

This function returns no value.

Definition at line 151 of file users.c.

References meta_user_t::lock, and mutex_lock().

Referenced by imap_append(), imap_close(), imap_copy(), imap_create(), imap_delete(), imap_examine(), imap_expunge(), imap_fetch(), imap_rename(), imap_select(), imap_session_destroy(), imap_session_update(), imap_store(), imap_subscribe(), meta_contacts_update(), meta_folders_update(), meta_get(), meta_message_folders_update(), meta_messages_copier(), meta_messages_login_update(), meta_messages_mover(), meta_messages_update(), meta_user_build(), meta_user_update(), pop_delete(), pop_pass(), pop_session_destroy(), pop_session_reset(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(), portal_endpoint_contacts_move(), portal_endpoint_contacts_remove(), portal_endpoint_folders_rename(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(), portal_endpoint_messages_tag(), portal_folder_contacts_add(), portal_folder_contacts_remove(), portal_folder_mail_add(), and portal_folder_mail_remove().

magma/objects/users/users.h File Reference

Functions to interface with and manage user data.

Data Structures

- struct **meta_stats_tag_t**

Defines

- #define **MAIL_STATUS_USER_FLAGS** (MAIL_STATUS_SEEN | MAIL_STATUS_ANSWERED | MAIL_STATUS_FLAGGED | MAIL_STATUS_DELETED | MAIL_STATUS_DRAFT)
- #define **MAIL_STATUS_SYSTEM_FLAGS**
- #define **MAIL_STATUS_ALL_FLAGS** (MAIL_STATUS_USER_FLAGS | MAIL_STATUS_SYSTEM_FLAGS)
- #define **MESSAGE_TYPE_UNKNOWN** 0
- #define **MESSAGE_TYPE_PLAIN** 1
- #define **MESSAGE_TYPE_HTML** 2
- #define **MESSAGE_TYPE_MULTI_ALTERNATIVE** 3
- #define **MESSAGE_TYPE_MULTI_MIXED** 4
- #define **MESSAGE_TYPE_MULTI_RELATED** 5
- #define **MESSAGE_TYPE_MULTI_RFC822** 5
- #define **MESSAGE_TYPE_MULTI_UNKOWN** 6
- #define **MESSAGE_ENCODING_UNKNOWN** 0
- #define **MESSAGE_ENCODING_QUOTED_PRINTABLE** 1
- #define **MESSAGE_ENCODING_BASE64** 2
- #define **MESSAGE_ENCODING_7BIT** 3
- #define **MESSAGE_ENCODING_8BIT** 4

Enumerations

- enum { **MAIL_STATUS_NONE** = 0, **MAIL_STATUS_EMPTY** = 1, **MAIL_STATUS_RECENT** = 2, **MAIL_STATUS_SEEN** = 4, **MAIL_STATUS_ANSWERED** = 8, **MAIL_STATUS_FLAGGED** = 16, **MAIL_STATUS_DELETED** = 32, **MAIL_STATUS_DRAFT** = 64, **MAIL_STATUS_SECURE** = 128, **MAIL_STATUS_APPENDED** = 256, **MAIL_STATUS_HIDDEN** = 512, **MAIL_MARK_JUNK** = 1024, **MAIL_MARK_INFECTED** = 2048, **MAIL_MARK_SPOOFED** = 4096, **MAIL_MARK_BLACKHOLED** = 8192, **MAIL_MARK_PHISHING** = 16384, **MAIL_STATUS_TAGGED** = 32768, **MAIL_STATUS_ENCRYPTED** = 65536 }

Functions

- **int_t meta_message_folders_update** (meta_user_t *user, META_LOCK_STATUS locked)
- *messages.c* **meta_stats_tag_t * meta_folder_stats_tag_alloc** (stringer_t *tag)
- *folders.c* **meta_folder_t * meta_folders_by_name** (inx_t *folders, stringer_t *name)
- *Get a folder by its fully qualified name.* **meta_folder_t * meta_folders_by_number** (inx_t *folders, uint64_t number)
- *Get a folder by number.* **int_t meta_folders_children** (inx_t *folders, uint64_t number)
- *Get the number of direct child folders of a specified parent folder.* **stringer_t * meta_folders_name** (inx_t *list, meta_folder_t *folder)
- *Get a folder's fully qualified name.* **inx_t * meta_folders_stats_tags** (inx_t *messages, uint64_t folder)
- *Create a collection of meta tag stats for a set of messages.* **int_t meta_folders_update** (meta_user_t *user, META_LOCK_STATUS locked)
- *Update a user's message folders if necessary.* **bool_t meta_data_acknowledge_alert** (uint64_t alertnum, uint64_t usernum, uint32_t transaction)

- *datatier.c* uint64_t **meta_data_delete_folder** (uint64_t usernum, uint64_t foldernum)
- *Delete a message folder from the database.* int_t **meta_data_delete_tag** (meta_message_t *message, stringer_t *tag)
- *Remove a tag from a message in the database..* inx_t * **meta_data_fetch_alerts** (uint64_t usernum)
- *Get all unacknowledged alert messages for a user.* bool_t **meta_data_fetch_folders** (meta_user_t *user)
- *Retrieve all of a user's message folders from the database.* bool_t **meta_data_fetch_mailbox_aliases** (meta_user_t *user)
- *Get all the mailbox aliases for a specified user.* bool_t **meta_data_check_mailbox** (stringer_t *address)
- *Check to see if a specified mailbox exists.* bool_t **meta_data_fetch_messages** (meta_user_t *user)
- *Fetch all of a user's stored messages from the database and attach them to the meta user object.* int_t **meta_check_message_encryption** (meta_user_t *user)
- *Make sure that a user's messages' on-disk encryption statuses match the user's security settings.* inx_t * **meta_data_fetch_all_tags** (uint64_t usernum)
- *Fetch all the tags attached to messages of a specified user from the database.* void **meta_data_fetch_message_tags** (meta_message_t *message)
- *Fetch the tags for a specified message from the database.* bool_t **meta_data_fetch_user** (meta_user_t *user)
- *Populate a meta user object with user information stored in the database.* bool_t **meta_data_flags_add** (inx_t *messages, uint64_t usernum, uint64_t foldernum, uint32_t flags)
- *Add the specified flags mask to a collection of mail messages.* bool_t **meta_data_flags_remove** (inx_t *messages, uint64_t usernum, uint64_t foldernum, uint32_t flags)
- *Remove the specified flags mask from a collection of mail messages.* bool_t **meta_data_flags_replace** (inx_t *messages, uint64_t usernum, uint64_t foldernum, uint32_t flags)
- *Remove all user (non-system) flags from a collection of mail messages, and set the specified flags mask for them.* uint64_t **meta_data_insert_folder** (uint64_t usernum, stringer_t *name, uint64_t parent, uint32_t order)
- *Insert a new mail folder into the database.* int_t **meta_data_insert_tag** (meta_message_t *message, stringer_t *tag)
- *Insert a tag for a message into the database.* int_t **meta_data_truncate_tags** (meta_message_t *message)
- *Remove all tags associated with a message in the database.* uint64_t **meta_data_update_folder_name** (uint64_t usernum, uint64_t foldernum, stringer_t *name, uint64_t parent, uint32_t order)
- *Update the record for a message folder in the database.* void **meta_data_update_lock** (uint64_t usernum, uint8_t lock)
- *Update a user's lock in the database.* void **meta_data_update_log** (meta_user_t *user, META_PROT prot)
- *Update the per-user entry in the Log table for the specified protocol.* int_t **meta_data_user_build** (meta_user_t *user, stringer_t *passhash, stringer_t *passkey)
- *Build a meta user object by username, hashed password, and hashed key storage password.* int_t **meta_data_user_build_storage_keys** (uint64_t usernum, stringer_t *passkey, stringer_t **priv_out, stringer_t **pub_out, bool_t dont_create, bool_t do_trans, uint32_t tid)
- *Retrieve the on-disk mail storage key pair associated with the user, or create it if it doesn't exist.* int_t **meta_data_user_save_storage_keys** (uint64_t usernum, stringer_t *passkey, stringer_t *pubkey, stringer_t *privkey, bool_t do_trans, uint32_t tid)
- *Persist the user's storage keys into the mysql database.* meta_alert_t * **alert_alloc** (uint64_t alertnum, stringer_t *type, stringer_t *message, uint64_t created)
- *alerts.c* meta_alias_t * **alias_alloc** (uint64_t aliasnum, stringer_t *address, stringer_t *display, int_t selected, uint64_t created)
- *aliases.c* int_t **meta_contacts_update** (meta_user_t *user, META_LOCK_STATUS locked)
- *contacts.c* int_t **meta_get** (stringer_t *username, stringer_t *mbboxdomain, stringer_t *passhash, stringer_t *passkey, META_PROT flags, META_GET get, meta_user_t **output)
- *users.c* void **meta_remove** (stringer_t *username, META_PROT flags)
- *Lock a user's object in the cache and decrement their reference counter.* int_t **meta_user_build** (meta_user_t *user, stringer_t *username, stringer_t *passhash, stringer_t *passkey, META_LOCK_STATUS locked)
- *Build a user's meta information from specified data parameters.* meta_user_t * **meta_user_create** (void)
- *Allocate and initialize a meta user object.* void **meta_user_destroy** (meta_user_t *user)
- *Destroy a meta user object and free all of its underlying data.* int_t **meta_user_prune** (stringer_t *username)

- *Locate and remove a user's object from the cache if the reference counter is zero.* void **meta_user_ref_add** (**meta_user_t** *user, **META_PROT** protocol)
 - *Increment a meta user's reference counter for a specified protocol and update the activity timestamp.* void **meta_user_ref_dec** (**meta_user_t** *user, **META_PROT** protocol)
 - *Decrement a user's reference counter for a specified protocol and update the activity timestamp.* time_t **meta_user_ref_stamp** (**meta_user_t** *user)
 - *Get the activity timestamp for a meta user object.* uint64_t **meta_user_ref_total** (**meta_user_t** *user)
 - *Get the total session reference count for a user for all protocols.* void **meta_user_rlock** (**meta_user_t** *user)
 - *Acquire a read lock for a meta user object.* **bool_t** **meta_user_serial_check** (**meta_user_t** *user, uint64_t object)
 - *Check an object's serial number to see if it is up-to-date.* uint64_t **meta_user_serial_get** (**meta_user_t** *user, uint64_t object)
 - *Get an object serial number for a given meta user structure.* void **meta_user_serial_set** (**meta_user_t** *user, uint64_t object, uint64_t serial)
 - *Set an object serial number for a given meta user structure.* void **meta_user_unlock** (**meta_user_t** *user)
 - *Release the lock for a meta user object.* **int_t** **meta_user_update** (**meta_user_t** *user, **META_LOCK_STATUS** locked)
 - *Update a meta user object from the cache.* void **meta_user_wlock** (**meta_user_t** *user)
- Acquire a write lock for a meta user object.*
-

Detailed Description

Functions to interface with and manage user data.

Definition in file **users.h**.

Define Documentation

#define MAIL_STATUS_ALL_FLAGS (MAIL_STATUS_USER_FLAGS | MAIL_STATUS_SYSTEM_FLAGS)

Definition at line 51 of file **users.h**.

#define MAIL_STATUS_SYSTEM_FLAGS

```
Value: (MAIL_STATUS_EMPTY | MAIL_STATUS_RECENT | MAIL_STATUS_SECURE | MAIL_STATUS_APPENDED |
MAIL_STATUS_HIDDEN | \
        MAIL_MARK_JUNK | MAIL_MARK_INFECTED | MAIL_MARK_SPOOFED | MAIL_MARK_BLACKHOLED |
MAIL_MARK_PHISHING | MAIL_STATUS_TAGGED | MAIL_STATUS_ENCRYPTED)
```

Definition at line 47 of file **users.h**.

Referenced by **portal_endpoint_messages_flag()**.

#define MAIL_STATUS_USER_FLAGS (MAIL_STATUS_SEEN | MAIL_STATUS_ANSWERED | MAIL_STATUS_FLAGGED | MAIL_STATUS_DELETED | MAIL_STATUS_DRAFT)

Definition at line 44 of file **users.h**.

Referenced by **meta_data_flags_replace()**, and **portal_endpoint_messages_flag()**.

#define MESSAGE_ENCODING_7BIT 3

Definition at line 67 of file users.h.

Referenced by mail_discover_encoding(), and mail_mime_encoding().

#define MESSAGE_ENCODING_8BIT 4

Definition at line 68 of file users.h.

Referenced by mail_discover_encoding(), and mail_mime_encoding().

#define MESSAGE_ENCODING_BASE64 2

Definition at line 66 of file users.h.

Referenced by mail_discover_encoding(), mail_mime_encoding(), and mail_modify_part().

#define MESSAGE_ENCODING_QUOTED_PRINTABLE 1

Definition at line 65 of file users.h.

Referenced by mail_build_signature(), mail_discover_encoding(), and mail_mime_encoding().

#define MESSAGE_ENCODING_UNKNOWN 0

Definition at line 64 of file users.h.

Referenced by mail_discover_encoding(), and mail_mime_encoding().

#define MESSAGE_TYPE_HTML 2

Definition at line 56 of file users.h.

Referenced by mail_build_signature(), mail_discover_insertion_point(), mail_discover_type(), mail_mime_type(), mail_modify_part(), portal_message_attachments(), and portal_message_body().

#define MESSAGE_TYPE_MULTI_ALTERNATIVE 3

Definition at line 57 of file users.h.

Referenced by mail_discover_type(), mail_mime_part(), mail_mime_type(), mail_modify_part(), and portal_message_body().

#define MESSAGE_TYPE_MULTI_MIXED 4

Definition at line 58 of file users.h.

Referenced by mail_discover_type(), mail_mime_part(), mail_mime_type(), mail_modify_part(), portal_message_attachments(), and portal_message_body().

#define MESSAGE_TYPE_MULTI_RELATED 5

Definition at line 59 of file users.h.

Referenced by mail_discover_type(), mail_mime_part(), mail_mime_type(), mail_modify_part(), and portal_message_body().

#define MESSAGE_TYPE_MULTI_RFC822 5

Definition at line 60 of file users.h.

Referenced by mail_mime_part(), and mail_mime_type().

#define MESSAGE_TYPE_MULTI_UNKOWN 6

Definition at line 61 of file users.h.

Referenced by mail_discover_type(), mail_mime_part(), mail_mime_type(), mail_modify_part(), and portal_message_body().

#define MESSAGE_TYPE_PLAIN 1

Definition at line 55 of file users.h.

Referenced by mail_discover_type(), mail_mime_type(), mail_modify_part(), portal_message_attachments(), and portal_message_body().

#define MESSAGE_TYPE_UNKNOWN 0

Definition at line 54 of file users.h.

Referenced by mail_mime_type(), and mail_modify_part().

Enumeration Type Documentation

anonymous enum

Enumerator:

MAIL_STATUS_NONE
MAIL_STATUS_EMPTY
MAIL_STATUS_RECENT
MAIL_STATUS_SEEN
MAIL_STATUS_ANSWERED
MAIL_STATUS_FLAGGED
MAIL_STATUS_DELETED
MAIL_STATUS_DRAFT
MAIL_STATUS_SECURE
MAIL_STATUS_APPENDED
MAIL_STATUS_HIDDEN
MAIL_MARK_JUNK
MAIL_MARK_INFECTED
MAIL_MARK_SPOOFED
MAIL_MARK_BLACKHOLED
MAIL_MARK_PHISHING
MAIL_STATUS_TAGGED

MAIL_STATUS_ENCRYPTED

Definition at line 17 of file users.h.

Function Documentation

meta_alert_t* alert_alloc (uint64_t *alertnum*, stringer_t * *type*, stringer_t * *message*, uint64_t *created*)

alerts.c

alerts.c

Parameters:

alertnum the numerical id of the user alert.

type a pointer to a managed string containing the type of the alert (e.g. "warning" or "alert").

message a pointer to a managed string containing the alert message.

created the UTC timecode for when the alert message was created.

Returns:

NULL on error or a pointer to the newly initialized user alert message object on success.

Definition at line 23 of file alerts.c.

References align(), FOREIGNDATA, JOINED, log_pedantic, mm_alloc(), mm_copy(), PLACER_T, placer_t, st_data_get(), st_length_get(), and STACK.

Referenced by meta_data_fetch_alerts().

meta_alias_t* alias_alloc (uint64_t *aliasnum*, stringer_t * *address*, stringer_t * *display*, int_t *selected*, uint64_t *created*)

aliases.c

aliases.c

Parameters:

aliasnum the numerical identifier of the user alias.

address a managed string containing the email address associated with the alias.

display a managed string containing the display name associated with the alias.

selected a flag specifying whether this alias is the default selection for the user.

created a UNIX timestamp for the creation of this alias.

Returns:

NULL on failure, or a pointer to the newly initialized user alias object on success.

Definition at line 24 of file aliases.c.

References align(), FOREIGNDATA, JOINED, log_pedantic, mm_alloc(), mm_copy(), PLACER_T, placer_t, st_data_get(), st_length_get(), and STACK.

Referenced by meta_data_fetch_mailbox_aliases().

int_t meta_check_message_encryption (meta_user_t * *user*)

Make sure that a user's messages' on-disk encryption statuses match the user's security settings.

Note:

If the user's secure flag is on, then all messages should be encrypted. Any unencrypted messages need to be encrypted in place. If the secure flag has been disabled and there are still encrypted messages on disk, the messages need to be reverted to plaintext form.

Parameters:

user the meta user object that owns the specified messages.

Returns:

-1 on failure, 0 if there were no messages to be sync'ed, or 1 if additional encryption processing is required.

Definition at line 1044 of file `datatier.c`.

References `decrypt_user_messages()`, `encrypt_user_messages()`, `enqueue()`, `meta_user_t::flags`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_ENCRYPTED`, `meta_user_t::messages`, `META_PROT_GENERIC`, `META_USER_ENCRYPT_DATA`, `meta_user_ref_add()`, `meta_message_t::status`, `meta_user_t::storage_privkey`, and `meta_user_t::storage_pubkey`.

Referenced by `meta_data_fetch_messages()`.

`int_t meta_contacts_update (meta_user_t * user, META_LOCK_STATUS locked)`

`contacts.c`

`contacts.c`

Note:

This function will try to retrieve the folders from the cache, if possible, or fall back to the database.

Parameters:

user a pointer to the meta user object that will have its message folders updated.

locked if `META_NEED_LOCK` is specified, the meta user object will be locked for operation.

Returns:

-1 on failure or 1 on success.

Definition at line 22 of file `contacts.c`.

References `meta_user_t::contacts`, `contacts_update()`, `inx_free()`, `M_FOLDER_CONTACTS`, `META_NEED_LOCK`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_CONTACTS`, `serial_get()`, `serial_increment()`, `meta_user_t::serials`, and `meta_user_t::usernum`.

Referenced by `meta_get()`, and `sess_update()`.

`bool_t meta_data_acknowledge_alert (uint64_t alertnum, uint64_t usernum, uint32_t transaction)`

`datatier.c`

`datatier.c`

Note:

If the table is not updated immediately, another check is made to see if the alert is still pending. If so, false is returned.

Parameters:

alertnum the numerical id of the alert message to be acknowledged.

usernum the numerical id of the user to whom the alert message belongs.

transaction the mysql transaction id of the acknowledgment operation, in cases batch changes need to be rolled back.

Returns:

true if the alert was acknowledged successfully, or false on failure.

Definition at line 1655 of file datatier.c.

References log_pedantic, mm_wipe(), res_field_uint64(), res_row_next(), res_table_free(), stmt_exec_affected_conn(), and stmt_get_result_conn().

Referenced by portal_endpoint_alert_acknowledge().

bool_t meta_data_check_mailbox (stringer_t * address)

Check to see if a specified mailbox exists.

Parameters:

address a managed string containing the email address to be matched against any of the rows in the Mailboxes table.

Returns:

true if the specified email address exists as a configured mailbox in the database or false otherwise.

Definition at line 1837 of file datatier.c.

References log_pedantic, mm_wipe(), res_row_next(), res_table_free(), st_char_get(), st_empty(), st_length_get(), and stmt_get_result().

Referenced by meta_get().

uint64_t meta_data_delete_folder (uint64_t usernum, uint64_t foldernum)

Delete a message folder from the database.

Parameters:

usernum the numerical id of the user that owns the folder to be deleted.

foldernum the folder id of the message folder to be deleted.

Returns:

1 if the message folder was successfully, or <= 0 if there was an error.

Definition at line 237 of file datatier.c.

References M_FOLDER_MESSAGES, mm_wipe(), stmt_exec_affected(), and type().

Referenced by imap_folder_remove().

int_t meta_data_delete_tag (meta_message_t * message, stringer_t * tag)

Remove a tag from a message in the database..

Parameters:

message a pointer to the meta message object of the message to have the tag stripped.

tag a managed string containing the name of the tag to be deleted.

Returns:

0 on success or -1 on failure.

Definition at line 503 of file datatier.c.

References meta_message_t::messagenum, mm_wipe(), st_char_get(), st_length_get(), and stmt_exec_affected().

Referenced by portal_endpoint_messages_tag().

inx_t* meta_data_fetch_alerts (uint64_t usernum)

Get all unacknowledged alert messages for a user.

Parameters:

usernum the numerical id of the user for whom the alert messages will be fetched.

Returns:

NULL on failure, or a pointer to an inx holder containing all of the user's alert messages on success.

Definition at line 1718 of file datatier.c.

References alert_alloc(), inx_alloc(), inx_free(), inx_insert(), log_error, log_info, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, mm_free(), mm_wipe(), PLACER, res_field_block(), res_field_length(), res_field_uint64(), res_row_next(), res_table_free(), stmt_get_result(), multi_t::u64, and multi_t::val.

Referenced by portal_endpoint_alert_list().

inx_t* meta_data_fetch_all_tags (uint64_t usernum)

Fetch all the tags attached to messages of a specified user from the database.

Parameters:

usernum the numerical id of the user for whom the message tags will be fetched.

Returns:

NULL on failure, or an inx holder containing a list of all the user's messages' tags as managed strings on success.

Definition at line 534 of file datatier.c.

References inx_alloc(), inx_free(), inx_insert(), log_error, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, res_field_string(), res_row_next(), res_table_free(), st_free(), stmt_get_result(), multi_t::type, multi_t::u64, and multi_t::val.

Referenced by portal_endpoint_messages_tags().

bool_t meta_data_fetch_folders (meta_user_t * user)

Retrieve all of a user's message folders from the database.

Note:

If the user already had a working set of message folders they will be deleted first.

Parameters:

user a pointer to the meta user object of the user making the request, which will be updated on success.

Returns:

-1 on failure or 1 on success.

Definition at line 1093 of file datatier.c.

References meta_folder_t::foldernum, meta_user_t::folders, inx_alloc(), inx_cleanup(), inx_insert(), log_error, log_pedantic, M_FOLDER_MESSAGES, M_INX_LINKED, M_TYPE_UINT64, mm_alloc(), mm_copy(), mm_free(), mm_wipe(), meta_folder_t::name, meta_folder_t::order, meta_folder_t::parent, res_field_block(), res_field_length(), res_field_uint32(), res_field_uint64(), res_row_next(), res_table_free(), stmt_get_result(), multi_t::type, type(), multi_t::u64, meta_user_t::usernum, and multi_t::val.

Referenced by meta_folders_update().

bool_t meta_data_fetch_mailbox_aliases (meta_user_t * user)

Get all the mailbox aliases for a specified user.

Parameters:

user a pointer to the partially populated meta user object to be queried, with its usernum field set.

Returns:

true on success or false on failure.

Definition at line 1775 of file datatier.c.

References alias_alloc(), meta_user_t::aliases, inx_alloc(), inx_cleanup(), inx_insert(), log_error, log_info, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, mm_free(), mm_wipe(), PLACER, res_field_block(), res_field_length(), res_field_uint64(), res_field_uint8(), res_row_next(), res_table_free(), st_char_get(), st_length_int(), stmt_get_result(), multi_t::u64, meta_user_t::usernum, and multi_t::val.

Referenced by meta_user_build(), and meta_user_update().

void meta_data_fetch_message_tags (meta_message_t * message)

Fetch the tags for a specified message from the database.

Note:

The results of the operation will be stored in the specified meta message object's "tags" member.

Parameters:

message the meta message object for which the tags will be looked up.

Returns:

This function returns no value.

Definition at line 590 of file datatier.c.

References ar_alloc(), ar_append(), ARRAY_TYPE_STRINGER, meta_message_t::messagenum, mm_wipe(), res_field_string(), res_row_count(), res_row_next(), res_table_free(), stmt_get_result(), and meta_message_t::tags.

Referenced by meta_data_fetch_messages(), and portal_endpoint_messages_tag().

bool_t meta_data_fetch_messages (meta_user_t * user)

Fetch all of a user's stored messages from the database and attach them to the meta user object.

Note:

Any of the user's existing messages will be destroyed first to allow for updates.

Parameters:

user the meta user object whose mail messages will be retrieved.

Returns:

true on success or false on failure.

Definition at line 629 of file `datatier.c`.

References `meta_message_t::created`, `meta_message_t::foldernum`, `inx_alloc()`, `inx_cleanup()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_insert()`, `log_error`, `log_info`, `log_pedantic`, `M_INX_LINKED`, `M_TYPE_UINT64`, `MAIL_STATUS_TAGGED`, `meta_message_t::messagenum`, `meta_user_t::messages`, `meta_check_message_encryption()`, `meta_data_fetch_message_tags()`, `meta_message_free()`, `mm_alloc()`, `mm_copy()`, `mm_free()`, `mm_wipe()`, `res_field_block()`, `res_field_length()`, `res_field_uint32()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `meta_message_t::server`, `meta_message_t::sigkey`, `meta_message_t::signum`, `meta_message_t::size`, `st_char_get()`, `meta_message_t::status`, `stmt_get_result()`, `multi_t::type`, `multi_t::u64`, `meta_user_t::username`, `meta_user_t::usernum`, and `multi_t::val`.

Referenced by `meta_messages_login_update()`, and `meta_messages_update()`.

bool_t meta_data_fetch_user (meta_user_t * user)

Populate a meta user object with user information stored in the database.

Note:

Fields fetched from the database include: flags, hashed password, lock & ssl status, and quota information.

Parameters:

user a pointer to the partially populated meta user object to be updated, with the `usernum` field already supplied.

Returns:

true on success or false on failure.

LOW: This user fetch function was intended only to refresh existing user configurations/preferences. In theory an inactivity lock would have been cleared when the session was created so we shouldn't need to even check that value at this stage.

Definition at line 1193 of file `datatier.c`.

References `meta_user_t::flags`, `meta_user_t::lock_status`, `meta_data_update_lock()`, `META_USER_ENCRYPT_DATA`, `META_USER_OVERQUOTA`, `META_USER_SSL`, `mm_wipe()`, `meta_user_t::passhash`, `res_field_int8()`, `res_field_string()`, `res_row_next()`, `res_table_free()`, `st_cleanup()`, `stmt_get_result()`, and `meta_user_t::usernum`.

Referenced by `meta_user_build()`, and `meta_user_update()`.

bool_t meta_data_flags_add (inx_t * messages, uint64_t usernum, uint64_t foldernum, uint32_t flags)

Add the specified flags mask to a collection of mail messages.

Parameters:

messages an `inx` holder containing the collection of messages to have their flags updated.

usernum the numerical id of the user to whom the target messages belong, for validation purposes.

foldernum the numerical id of the parent folder containing the messages to be updated, for validation purposes.
flags a mask of all flags that are to be added to any matching messages in the collection.

Returns:

true on success or false on failure.

Definition at line 176 of file `datatier.c`.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `log_pedantic`, `meta_message_t::messagenum`, `mm_wipe()`, and `stmt_exec()`.

Referenced by `adjust_message_encryption()`, `imap_fetch()`, `imap_update_flags()`, `portal_endpoint_messages_flag()`, and `portal_endpoint_messages_tag()`.

`bool_t meta_data_flags_remove (inx_t * messages, uint64_t usernum, uint64_t foldernum, uint32_t flags)`

Remove the specified flags mask from a collection of mail messages.

Parameters:

messages an inx holder containing the collection of messages to have their flags removed.

usernum the numerical id of the user to whom the target messages belong, for validation purposes.

foldernum the numerical id of the parent folder containing the messages to be updated, for validation purposes.

flags a mask of all flags that are to be stripped from any matching messages in the collection.

Returns:

true on success or false on failure.

Definition at line 104 of file `datatier.c`.

References `meta_message_t::foldernum`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `log_pedantic`, `meta_message_t::messagenum`, `mm_wipe()`, and `stmt_exec()`.

Referenced by `adjust_message_encryption()`, `imap_select()`, `imap_update_flags()`, `portal_endpoint_messages_flag()`, and `portal_endpoint_messages_tag()`.

`bool_t meta_data_flags_replace (inx_t * messages, uint64_t usernum, uint64_t foldernum, uint32_t flags)`

Remove all user (non-system) flags from a collection of mail messages, and set the specified flags mask for them.

Note:

The new mask can contain both user and system flags, but only user flags will be stripped from each message initially.

Parameters:

messages an inx holder containing the collection of messages to have their flags updated.

usernum the numerical of the user to whom the target messages belong, for validation purposes.

foldernum the numerical id of the parent folder containing the messages to be updated, for validation purposes.

flags a mask of all flags that are to be added to any matching messages in the collection.

Returns:

true on success or false on failure.

Definition at line 24 of file `datatier.c`.

References `meta_message_t::foldernum`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `log_pedantic`, `MAIL_STATUS_USER_FLAGS`, `meta_message_t::messagenum`, `mm_wipe()`, and `stmt_exec()`.

Referenced by `imap_update_flags()`, and `portal_endpoint_messages_flag()`.

`uint64_t meta_data_insert_folder (uint64_t usernum, stringer_t * name, uint64_t parent, uint32_t order)`

Insert a new mail folder into the database.

Parameters:

usernum the numerical id of the user to whom the new folder belongs.

name a managed string containing the name of the new mail folder.

parent the numerical id of the mail folder to be the parent of the new mail folder.

order the order number of this folder in its parent folder.

Returns:

0 on failure, or the numerical id of the newly inserted mail folder in the database on success.

Definition at line 329 of file `datatier.c`.

References `M_FOLDER_MESSAGES`, `mm_wipe()`, `st_char_get()`, `st_length_get()`, `stmt_insert()`, and `type()`.

Referenced by `imap_folder_create()`, and `imap_folder_rename()`.

`int_t meta_data_insert_tag (meta_message_t * message, stringer_t * tag)`

Insert a tag for a message into the database.

Parameters:

message the meta message object of the message to be tagged.

tag a managed string containing the name of the tag.

Returns:

0 on success or -1 on failure.

Definition at line 445 of file `datatier.c`.

References `meta_message_t::messagenum`, `mm_wipe()`, `st_char_get()`, `st_length_get()`, and `stmt_exec_affected()`.

Referenced by `portal_endpoint_messages_tag()`.

`int_t meta_data_truncate_tags (meta_message_t * message)`

Remove all tags associated with a message in the database.

Parameters:

message a pointer to the meta message object of the message to have all of its tags stripped.

Returns:

0 on success or -1 on failure.

Definition at line 476 of file `datatier.c`.

References meta_message_t::messagenum, mm_wipe(), and stmt_exec_affected().

Referenced by portal_endpoint_messages_tag().

uint64_t meta_data_update_folder_name (uint64_t usernum, uint64_t foldernum, stringer_t * name, uint64_t parent, uint32_t order)

Update the record for a message folder in the database.

Parameters:

usernum the numerical id of the user that owns the specified folder.

foldernum the id of the folder to have its properties adjusted.

name a managed string containing the new name of the specified folder.

parent the id of the new parent folder to be set for the specified message folder.

order the value of the order for the specified folder.

Returns:

1 on success, or <= 0 on failure.

Definition at line 275 of file datatier.c.

References M_FOLDER_MESSAGES, mm_wipe(), st_char_get(), st_length_get(), stmt_exec_affected(), and type().

Referenced by imap_folder_rename().

void meta_data_update_lock (uint64_t usernum, uint8_t lock)

Update a user's lock in the database.

Parameters:

usernum the numerical id of the user for whom the lock will be set.

lock the new value to which the specified user's lock will be set.

Returns:

This function returns no value.

Definition at line 408 of file datatier.c.

References log_pedantic, mm_wipe(), and stmt_exec_affected().

Referenced by meta_data_fetch_user(), meta_data_user_build(), and smtp_fetch_authorization().

void meta_data_update_log (meta_user_t * user, META_PROT prot)

Update the per-user entry in the Log table for the specified protocol.

Note:

This function will set the last session timestamp for the user, and increment the sessions counter in the database.

Parameters:

user the meta user object of the user making the logging request.

prot the protocol associated with the log request: META_PROT_POP, META_PROT_IMAP, or META_PROT_WEB.

Returns:

This function returns no value.

Definition at line 375 of file `datatier.c`.

References `log_pedantic`, `META_PROT_IMAP`, `META_PROT_POP`, `META_PROT_WEB`, `mm_wipe()`, `stmt_exec_affected()`, and `meta_user_t::usernum`.

Referenced by `imap_login()`, and `pop_pass()`.

`int_t meta_data_user_build (meta_user_t * user, stringer_t * passhash, stringer_t * passkey)`

Build a meta user object by username, hashed password, and hashed key storage password.

Parameters:

user a user meta object with the username field populated.

passhash a managed string with a multi-round hashed of the user's password for mysql database lookup.

passkeys a managed string with a single-pass hash of the user's password.

Returns:

-1 for unexpected program/system error, 0 for password auth failure, or 1 on success.

Definition at line 1264 of file `datatier.c`.

References `meta_user_t::flags`, `meta_user_t::lock_status`, `log_pedantic`, `meta_data_update_lock()`, `meta_data_user_build_storage_keys()`, `META_USER_ENCRYPT_DATA`, `META_USER_OVERQUOTA`, `META_USER_SSL`, `mm_wipe()`, `meta_user_t::passhash`, `res_field_int8()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `st_char_get()`, `st_dup()`, `st_empty()`, `st_free()`, `st_length_get()`, `st_length_int()`, `stmt_get_result()`, `meta_user_t::storage_privkey`, `meta_user_t::storage_pubkey`, `meta_user_t::username`, and `meta_user_t::usernum`.

Referenced by `meta_user_build()`.

`int_t meta_data_user_build_storage_keys (uint64_t usernum, stringer_t * passkey, stringer_t ** priv_out, stringer_t ** pub_out, bool dont_create, bool_t do_trans, uint32_t tid)`

Retrieve the on-disk mail storage key pair associated with the user, or create it if it doesn't exist.

Parameters:

usernum the user id of the target account.

passkey the single-round hashed password for storage key management, which can be NULL if *dont_create* is specified.

priv_out a pointer to a managed string to receive a copy of the ECIES private key if not NULL.

pub_out a pointer to a managed string to receive a copy of the ECIES public key if not NULL.

dont_create if true, ensures that storage keys won't be created if they don't already exist.

do_trans specifies whether or not a transaction id will be supplied for database operations.

tid if *do_trans* is set, the mysql transaction id to be used for all database operations.

Returns:

-1 on general failure, 0 if keys were successfully retrieved, or 1 if keys were generated.

Definition at line 1381 of file `datatier.c`.

References `base64_decode_mod()`, `base64_decode_opts()`, `CONTIGUOUS`, `ecies_key_create()`, `ecies_key_free()`, `ecies_key_private_bin()`, `ecies_key_public_bin()`, `log_info`, `log_pedantic`, `MANAGED_T`, `meta_data_user_save_storage_keys()`, `mm_free()`, `mm_sec_free()`, `mm_wipe()`, `res_field_block()`, `res_field_length()`, `res_row_next()`, `res_table_free()`, `scramble_decrypt()`, `SECURE`, `st_alloc_opts()`,

st_cleanup(), st_copy_in(), st_data_get(), st_empty(), st_free(), st_import(), stmt_get_result(), and stmt_get_result_conn().

Referenced by meta_data_user_build(), register_data_insert_user(), and smtp_store_message().

int_t meta_data_user_save_storage_keys (uint64_t *usernum*, stringer_t * *passkey*, stringer_t * *pubkey*, stringer_t * *privkey*, bool_t *do_trans*, uint32_t *tid*)

Persist the user's storage keys into the mysql database.

Parameters:

usernum the user id of the target account.

passkey the passkey that will be used to encrypt the user's private key symmetrically in the database.

pubkey the user's storage public key as a base64 encoded string.

privkey the user's encrypted storage private key as a base64 encoded string.

do_trans specifies whether or not a transaction id will be supplied for database operations.

tid if *do_trans* is set, the mysql transaction id to be used for all database operations.

Returns:

-1 on failure or 0 on success.

Definition at line 1576 of file datatier.c.

References base64_encode_mod(), log_info, mm_copy(), mm_wipe(), scramble_encrypt(), scramble_free(), scramble_total_length(), st_char_get(), st_free(), st_import(), st_length_get(), stmt_exec_affected(), and stmt_exec_affected_conn().

Referenced by meta_data_user_build_storage_keys().

meta_stats_tag_t* meta_folder_stats_tag_alloc (stringer_t * *tag*)

folders.c

folders.c

Parameters:

tag a managed string containing the name of the message tag.

Returns:

NULL on failure, or a newly allocated and initialized meta tag stat object on success.

Definition at line 20 of file folders.c.

References align(), FOREIGNDATA, JOINTED, log_pedantic, mm_alloc(), mm_copy(), PLACER_T, placer_t, st_data_get(), st_length_get(), STACK, and meta_stats_tag_t::tag.

Referenced by meta_folders_stats_tags().

meta_folder_t* meta_folders_by_name (inx_t * *folders*, stringer_t * *name*)

Get a folder by its fully qualified name.

Parameters:

folders a pointer to an inx holder containing a list of folders to be traversed in the search.

folder a pointer to the meta folder object that is the leaf node of the folder path.

Returns:

NULL on failure or a managed string containing the fully qualified folder name on success.

Definition at line 143 of file folders.c.

References `CONSTANT`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `meta_folders_name()`, `st_cmp_ci_eq()`, `st_cmp_cs_eq()`, and `st_free()`.

Referenced by `imap_append()`, `imap_copy()`, `imap_folder_create()`, `imap_folder_remove()`, `imap_folder_rename()`, `imap_folder_status()`, `imap_subscribe()`, and `portal_folder_mail_add()`.

meta_folder_t* meta_folders_by_number (inx_t * *folders*, uint64_t *number*)

Get a folder by number.

Parameters:

folders a pointer to an inx holder containing the folders to be searched.

number the numerical id of the folder to be retrieved.

Returns:

NULL on failure, or a pointer to the found meta folder object of the specified folder on success.

Definition at line 184 of file folders.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, and `inx_cursor_value_next()`.

Referenced by `meta_folders_name()`, `portal_endpoint_folders_rename()`, `portal_endpoint_folders_tags()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_tag()`, `portal_folder_mail_add()`, and `portal_folder_mail_remove()`.

int_t meta_folders_children (inx_t * *folders*, uint64_t *number*)

Get the number of direct child folders of a specified parent folder.

Parameters:

folders a pointer to an inx holder containing the the collection of folders to be traversed.

number the folder id of the parent folder to be scanned for children.

Returns:

the number of children folders in the specified parent folder, or 0 on failure.

Definition at line 212 of file folders.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, and `meta_folder_t::parent`.

Referenced by `imap_folder_remove()`.

stringer_t* meta_folders_name (inx_t * *list*, meta_folder_t * *folder*)

Get a folder's fully qualified name.

Note:

This function will prepend the entire ancestor path of a folder to its name, with each level delimited by periods.

Parameters:

list a pointer to an inx holder containing a list of folders to be searched for parent folders.
folder a pointer to the meta folder object that is the leaf node of the folder path.

Returns:

NULL on failure or a managed string containing the fully qualified folder name on success.

Definition at line 104 of file folders.c.

References FOLDER_RECURSION_LIMIT, log_pedantic, meta_folders_by_number(), meta_folder_t::name, ns_length_get(), meta_folder_t::parent, st_free(), st_import(), and st_merge.

Referenced by imap_folder_name_escaped(), imap_narrow_folders(), meta_folders_by_name(), portal_endpoint_folders_rename(), portal_folder_mail_add(), and portal_folder_mail_remove().

inx_t* meta_folders_stats_tags (inx_t * *messages*, uint64_t *folder*)

Create a collection of meta tag stats for a set of messages.

Note:

Each meta tag stat will contain the name of a message tag, along with the number of messages with which it was associated.

Parameters:

messages an inx holder containing the list of messages to be examined.
folder the numerical id of the parent folder that contains all target messages.

Returns:

NULL on failure, or an inx holder containing all of the messages' meta tag stats on success.

LOW: This is a very inefficient method for counting the number of times each tag appears.

Definition at line 45 of file folders.c.

References ar_field_st(), ar_length_get(), meta_stats_tag_t::count, meta_message_t::foldernum, inx_alloc(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_find(), inx_insert(), log_pedantic, M_INX_HASHED, M_TYPE_STRINGER, meta_folder_stats_tag_alloc(), mm_free(), multi_t::st, meta_message_t::tags, and multi_t::val.

Referenced by portal_endpoint_folders_tags().

int_t meta_folders_update (meta_user_t * *user*, META_LOCK_STATUS *locked*)

Update a user's message folders if necessary.

Note:

This function will try to retrieve the folders from the cache, if possible, or fall back to the database.

Parameters:

user a pointer to the meta user object that will have its message folders updated.
locked if META_NEED_LOCK is specified, the meta user object will be locked for operation.

Returns:

-1 on failure or 1 on success.

Definition at line 242 of file folders.c.

References meta_user_t::folders, meta_user_t::messages, meta_data_fetch_folders(), meta_messages_update_sequences(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), OBJECT_FOLDERS, meta_user_t::pop, meta_user_t::refs, serial_get(), serial_increment(), meta_user_t::serials, and meta_user_t::usernum.

Referenced by imap_session_update(), meta_get(), and sess_update().

int_t meta_get (stringer_t * username, stringer_t * mboxdomain, stringer_t * passhash, stringer_t * passkey, META_PROT flags, META_GET get, meta_user_t ** output)

users.c

users.c

Note:

If the user is not found, one will be created.

Parameters:

username the username to be looked up.

mboxdomain if not NULL, an optional managed string containing the domain suffix of a mailbox address to be verified first.

passhash a managed string with a multi-round hash of the user's password.

passkey a managed string with a single-round hash of the user's password.

flags a set of flags specifying the protocol used by the calling function. Values can be META_PROT_NONE, META_PROT_SMTP, META_PROT_POP, META_PROT_IMAP, META_PROT_WEB, or META_PROT_GENERIC.

get a set of flags specifying the data to be retrieved (META_GET_NONE, META_GET_MESSAGES, META_GET_FOLDERS, or META_GET_CONTACTS)

output the address of a meta user object that will store a pointer to the result of the lookup.

Returns:

-1 on error, 0 if the username information exists but there was an error, and 1 on success.

Definition at line 533 of file users.c.

References inx_find(), inx_insert(), inx_lock_write(), inx_unlock(), log_error, log_pedantic, M_TYPE_STRINGER, meta_contacts_update(), meta_data_check_mailbox(), meta_folders_update(), META_GET_CONTACTS, META_GET_FOLDERS, META_GET_MESSAGES, META_LOCKED, meta_message_folders_update(), meta_messages_update(), meta_remove(), meta_user_build(), meta_user_create(), meta_user_destroy(), meta_user_ref_add(), meta_user_unlock(), meta_user_wlock(), objects, meta_user_t::passhash, st_cleanup(), st_cmp_cs_eq(), st_empty(), st_free(), st_merge, and object_cache_t::users.

Referenced by imap_login(), pop_pass(), and portal_endpoint_auth().

int_t meta_message_folders_update (meta_user_t * user, META_LOCK_STATUS locked)

messages.c

messages.c

TODO: The serial number scheme needs to refreshing. There should be a serial number to indicate changes in the folder structure, and then individual serial numbers for each folder to indicate when the contents are changed. If that were the case this function really only needs to update the list of folders.

See also:

messages_update()

Parameters:

user a pointer to the meta user object requesting the folders.

locked if META_LOCKED, lock the meta user object for the duration of the request.

Returns:

-1 on failure, or 1 on success.

Definition at line 27 of file messages.c.

References meta_user_t::folders, inx_free(), meta_user_t::message_folders, messages_update(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), OBJECT_FOLDERS, serial_get(), serial_increment(), meta_user_t::serials, and meta_user_t::usernum.

Referenced by meta_get().

void meta_remove (stringer_t * username, META_PROT flags)

Lock a user's object in the cache and decrement their reference counter.

See also:

meta_user_ref_dec()

Parameters:

username a managed string containing the name of the user to be adjusted.

flags specifies the protocol bound to the reference counter to be decremented (META_PROT_WEB, META_PROT_IMAP, etc.)

Returns:

This function returns no value.

Definition at line 220 of file users.c.

References inx_find(), inx_lock_read(), inx_unlock(), M_TYPE_STRINGER, meta_user_ref_dec(), objects, st_empty(), and object_cache_t::users.

Referenced by imap_login(), imap_session_destroy(), meta_get(), pop_pass(), pop_session_destroy(), portal_endpoint_auth(), and sess_destroy().

int_t meta_user_build (meta_user_t * user, stringer_t * username, stringer_t * passhash, stringer_t * passkey, META_LOCK_STATUS locked)

Build a user's meta information from specified data parameters.

Note:

The user object will be pulled from the cache, if possible, or falls back to a database lookup by username+password.

Parameters:

user a pointer to the meta user object of the user that is to be filled.

passhash a managed string with a multi-round hash of the user's password.

passkey a managed string with a single-round hash of the user's password.

locked the meta lock status of the operation (if META_NEED_LOCK is supplied, the meta user object will be locked for the duration of the function).

Returns:

true on success or false on failure.

Definition at line 386 of file users.c.

References meta_data_fetch_mailbox_aliases(), meta_data_fetch_user(), meta_data_user_build(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), OBJECT_USER, meta_user_t::passhash,

serial_get(), serial_increment(), meta_user_t::serials, st_cleanup(), st_dupe(), st_empty(), meta_user_t::user, meta_user_t::username, and meta_user_t::usernum.

Referenced by meta_get().

meta_user_t* meta_user_create (void)

Allocate and initialize a meta user object.

Returns:

NULL on failure, or a pointer to the newly allocated meta user object on success.

Definition at line 290 of file users.c.

References meta_user_t::lock, log_pedantic, mm_alloc(), mm_free(), mutex_destroy(), mutex_init(), and meta_user_t::refs.

Referenced by meta_get().

void meta_user_destroy (meta_user_t * user)

Destroy a meta user object and free all of its underlying data.

Parameters:

user a pointer to the meta user object to be destroyed.

Returns:

This function returns no value.

Definition at line 185 of file users.c.

References meta_user_t::ads, meta_user_t::aliases, meta_user_t::contacts, meta_user_t::folders, inx_cleanup(), meta_user_t::lock, meta_user_t::message_folders, meta_user_t::messages, mm_free(), mutex_destroy(), meta_user_t::passhash, meta_user_t::refs, st_cleanup(), meta_user_t::storage_privkey, meta_user_t::storage_pubkey, and meta_user_t::username.

Referenced by meta_get(), and obj_cache_start().

int_t meta_user_prune (stringer_t * username)

Locate and remove a user's object from the cache if the reference counter is zero.

Parameters:

username a managed string containing the name of the user whose data should be removed from the object cache.

Returns:

returns 1 if the user object was successfully pruned, 0 if the user isn't found, a non-zero reference count causes a return of -1, and a value of -2 is returned if an error occurs while trying to delete the user from the object cache index.

Definition at line 248 of file users.c.

References count, inx_count(), inx_delete(), inx_find(), inx_lock_write(), inx_unlock(), log_error, M_TYPE_STRINGER, meta_user_ref_total(), objects, st_empty(), stats_adjust_by_name(), stats_set_by_name(), and object_cache_t::users.

void meta_user_ref_add (meta_user_t * user, META_PROT protocol)

Increment a meta user's reference counter for a specified protocol and update the activity timestamp.

Note:

META_PROT_GENERIC can be specified for protocol non-specific accounting purposes.

Parameters:

user a pointer to the meta user object to be adjusted.

protocol the protocol identifier for the session.

Returns:

This function returns no value.

Definition at line 22 of file users.c.

References meta_user_t::generic, meta_user_t::imap, meta_user_t::lock, META_PROT_GENERIC, META_PROT_IMAP, META_PROT_POP, META_PROT_SMTP, META_PROT_WEB, mutex_lock(), mutex_unlock(), meta_user_t::pop, meta_user_t::refs, meta_user_t::smtp, meta_user_t::stamp, and meta_user_t::web.

Referenced by meta_check_message_encryption(), meta_get(), and portal_endpoint_auth().

void meta_user_ref_dec (meta_user_t * user, META_PROT protocol)

Decrement a user's reference counter for a specified protocol and update the activity timestamp.

Note:

META_PROT_GENERIC can be specified for protocol non-specific accounting purposes.

Parameters:

user a pointer to the meta user object to be adjusted.

protocol the protocol identifier for the session (META_PROT_WEB, META_PROT_IMAP, META_PROT_POP, META_PROT_SMTP, META_PROT_GENERIC).

Returns:

This function returns no value.

Definition at line 54 of file users.c.

References meta_user_t::generic, meta_user_t::imap, meta_user_t::lock, META_PROT_GENERIC, META_PROT_IMAP, META_PROT_POP, META_PROT_SMTP, META_PROT_WEB, mutex_lock(), mutex_unlock(), meta_user_t::pop, meta_user_t::refs, meta_user_t::smtp, meta_user_t::stamp, and meta_user_t::web.

Referenced by decrypt_user_messages(), encrypt_user_messages(), meta_remove(), and portal_endpoint_logout().

time_t meta_user_ref_stamp (meta_user_t * user)

Get the activity timestamp for a meta user object.

Parameters:

user a pointer to the meta user object to be examined.

Returns:

a timestamp containing the last time the meta user object's reference count changed.

Definition at line 109 of file users.c.

References meta_user_t::lock, mutex_lock(), mutex_unlock(), meta_user_t::refs, and meta_user_t::stamp.

Referenced by obj_cache_prune().

uint64_t meta_user_ref_total (meta_user_t * user)

Get the total session reference count for a user for all protocols.

Parameters:

user a pointer to the meta user object to be examined.

Returns:

the total number of references held by the user.

Definition at line 84 of file users.c.

References meta_user_t::generic, meta_user_t::imap, meta_user_t::lock, mutex_lock(), mutex_unlock(), meta_user_t::pop, meta_user_t::refs, meta_user_t::smtp, and meta_user_t::web.

Referenced by meta_user_prune(), and obj_cache_prune().

void meta_user_rlock (meta_user_t * user)

Acquire a read lock for a meta user object.

Parameters:

user a pointer to the meta user object to be locked.

Returns:

This function returns no value.

Definition at line 134 of file users.c.

References meta_user_t::lock, and mutex_lock().

Referenced by imap_close(), imap_examine(), imap_expunge(), imap_fetch(), imap_list(), imap_login(), imap_lsub(), imap_search_messages(), imap_status(), pop_last(), pop_list(), pop_retr(), pop_stat(), pop_top(), pop_uidl(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(), portal_endpoint_folders_tags(), portal_endpoint_messages_flag(), and portal_endpoint_messages_tag().

bool_t meta_user_serial_check (meta_user_t * user, uint64_t object)

Check an object's serial number to see if it is up-to-date.

Note:

The object's serial number will be incremented regardless of whether it is consistent with the cache.

Parameters:

user the meta user object to whom the object belongs.

object the serial object to be checked for changes.

Returns:

0 if the object did not to be refreshed, or 1 if it doesn't match the internal checkpoint and should be updated.
Definition at line 502 of file users.c.

References meta_user_serial_get(), meta_user_serial_set(), serial_get(), serial_increment(), and meta_user_t::usernum.

Referenced by portal_endpoint_messages_tag(), and sess_serial_check().

uint64_t meta_user_serial_get (meta_user_t * user, uint64_t object)

Get an object serial number for a given meta user structure.

Parameters:

user the meta user structure to be examined.

object the object to query for the serial number.

Returns:

the serial number value of the specified object, or 0 on failure.

Definition at line 472 of file users.c.

References meta_user_t::contacts, meta_user_t::folders, meta_user_t::messages, OBJECT_CONTACTS, OBJECT_FOLDERS, OBJECT_MESSAGES, OBJECT_USER, meta_user_t::serials, and meta_user_t::user.

Referenced by meta_user_serial_check().

void meta_user_serial_set (meta_user_t * user, uint64_t object, uint64_t serial)

Set an object serial number for a given meta user structure.

Parameters:

user the meta user object to be adjusted.

object the object to receive the new serial number.

serial the new serial number to be set.

Returns:

This function returns no value.

Definition at line 445 of file users.c.

References meta_user_t::contacts, meta_user_t::folders, meta_user_t::messages, OBJECT_CONTACTS, OBJECT_FOLDERS, OBJECT_MESSAGES, OBJECT_USER, meta_user_t::serials, and meta_user_t::user.

Referenced by meta_user_serial_check().

void meta_user_unlock (meta_user_t * user)

Release the lock for a meta user object.

Parameters:

user a pointer to the meta user object to be unlocked.

Returns:

This function returns no value.

Definition at line 168 of file users.c.

References meta_user_t::lock, and mutex_unlock().

Referenced by imap_append(), imap_close(), imap_copy(), imap_create(), imap_delete(), imap_examine(), imap_expunge(), imap_fetch(), imap_list(), imap_login(), imap_lsub(), imap_rename(), imap_search_messages(), imap_select(), imap_session_destroy(), imap_session_update(), imap_status(), imap_store(), imap_subscribe(), meta_contacts_update(), meta_folders_update(), meta_get(), meta_message_folders_update(), meta_messages_copier(), meta_messages_login_update(), meta_messages_mover(), meta_messages_update(), meta_user_build(), meta_user_update(), pop_delete(), pop_last(), pop_list(), pop_pass(), pop_retr(), pop_session_destroy(), pop_session_reset(), pop_stat(), pop_top(), pop_uidl(), portal_endpoint_contacts_add(), portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(), portal_endpoint_contacts_move(), portal_endpoint_contacts_remove(), portal_endpoint_folders_rename(), portal_endpoint_folders_tags(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(), portal_endpoint_messages_tag(), portal_folder_contacts_add(), portal_folder_contacts_remove(), portal_folder_mail_add(), and portal_folder_mail_remove().

int_t meta_user_update (meta_user_t * user, META_LOCK_STATUS locked)

Update a meta user object from the cache.

Parameters:

user a pointer to the meta user object that should be updated.

locked if META_NEED_LOCK is specified, a writer's lock will be acquired for the meta user object.

Returns:

-1 on general failure, 0 if the user or its mailbox aliases could not be fetched, or 1 on success.

Definition at line 339 of file users.c.

References meta_data_fetch_mailbox_aliases(), meta_data_fetch_user(), META_NEED_LOCK, meta_user_unlock(), meta_user_wlock(), OBJECT_USER, serial_get(), serial_increment(), meta_user_t::serials, meta_user_t::user, and meta_user_t::usernum.

Referenced by imap_session_update(), and sess_update().

void meta_user_wlock (meta_user_t * user)

Acquire a write lock for a meta user object.

Parameters:

user a pointer to the meta user object to be locked.

Returns:

This function returns no value.

Definition at line 151 of file users.c.

References meta_user_t::lock, and mutex_lock().

Referenced by imap_append(), imap_close(), imap_copy(), imap_create(), imap_delete(), imap_examine(), imap_expunge(), imap_fetch(), imap_rename(), imap_select(), imap_session_destroy(), imap_session_update(), imap_store(), imap_subscribe(), meta_contacts_update(), meta_folders_update(), meta_get(), meta_message_folders_update(), meta_messages_copier(), meta_messages_login_update(), meta_messages_mover(), meta_messages_update(), meta_user_build(), meta_user_update(), pop_delete(), pop_pass(), pop_session_destroy(), pop_session_reset(), portal_endpoint_contacts_add(),

portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(), portal_endpoint_contacts_move(),
portal_endpoint_contacts_remove(), portal_endpoint_folders_rename(), portal_endpoint_messages_copy(),
portal_endpoint_messages_flag(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(),
portal_endpoint_messages_tag(), portal_folder_contacts_add(), portal_folder_contacts_remove(),
portal_folder_mail_add(), and portal_folder_mail_remove().

magma/objects/warehouse/domains.c File Reference

Functions for managing the list of system domain names.

```
#include "magma.h"
```

Functions

- void **domain_stop** (void)
- *Free the global list of domains.* **bool_t domain_start** (void)
- *Fetch and store the list of configured domains from the database.* **domain_t * domain_alloc** (**stringer_t** *domain, **int_t** restricted, **int_t** mailboxes, **int_t** wildcard, **int_t** dkim, **int_t** spf)
- *Allocate and initialize a new domain object.* **int_t domain_mailboxes** (**stringer_t** *domain)
- *Determine whether mailboxes are hosted locally for a specified domain.* **int_t domain_restricted** (**stringer_t** *domain)
- *Determine whether a specified domain is restricted to authenticated users.* **int_t domain_wildcard** (**stringer_t** *domain)
- *Determine whether a specified domain has wildcards enabled.* **int_t domain_dkim** (**stringer_t** *domain)
- *TODO: Eliminate dkim+spf and replace them with a `sign` flag.* **int_t domain_spf** (**stringer_t** *domain)

Determine whether SPF has been configured for a specified domain. Variables

- **inx_t * domains** = NULL

Detailed Description

Functions for managing the list of system domain names.

Definition in file **domains.c**.

Function Documentation

domain_t* domain_alloc (**stringer_t** * *domain*, **int_t** *restricted*, **int_t** *mailboxes*, **int_t** *wildcard*, **int_t** *dkim*, **int_t** *spf*)

Allocate and initialize a new domain object.

domains.c

Parameters:

domain a pointer to a managed string containing the name of the specified domain.
restricted if set, indicate that the domain is restricted to authenticated users only.
mailboxes if set, indicate that mailboxes are hosted locally for the domain.
wildcard if set, indicate that wildcards are enabled for the domain.
dkim if set, indicate that outbound messages for the domain should be signed via DKIM.
spf if set, indicate that SPF is configured for the domain.

Returns:

NULL on failure or a pointer to the newly initialized domain object on success.

Definition at line 52 of file domains.c.

References `align()`, `FOREIGNDATA`, `JOINTED`, `log_pedantic`, `mm_alloc()`, `mm_copy()`, `PLACER_T`, `placer_t`, `st_data_get()`, `st_length_get()`, and `STACK`.

Referenced by `warehouse_fetch_domains()`.

`int_t domain_dkim (stringer_t * domain)`

TODO: Eliminate `dkim+spf` and replace them with a ``sign`` flag.

Determine whether outbound messages should be signed via DKIM for a specified domain.

Parameters:

domain a pointer to a managed string containing the name of the domain to be queried.

Returns:

-1 on failure or if the domain wasn't found, 0 if DKIM signing is disabled, or 1 if outbound messages should be signed via DKIM for the domain.

Definition at line 136 of file `domains.c`.

References `inx_find()`, and `M_TYPE_STRINGER`.

`int_t domain_mailboxes (stringer_t * domain)`

Determine whether mailboxes are hosted locally for a specified domain.

Parameters:

domain a pointer to a managed string containing the name of the domain to be queried.

Returns:

-1 on failure or if the domain was not found, 0 if the domain is foreign, or 1 if the mailbox is hosted locally for the domain.

Definition at line 81 of file `domains.c`.

References `inx_find()`, and `M_TYPE_STRINGER`.

`int_t domain_restricted (stringer_t * domain)`

Determine whether a specified domain is restricted to authenticated users.

Parameters:

domain a pointer to a managed string containing the name of the domain to be queried.

Returns:

-1 on failure or if the domain was not found, 0 if restricted relay is disabled, or 1 if the domain is restricted to authenticated users only.

Definition at line 99 of file `domains.c`.

References `inx_find()`, and `M_TYPE_STRINGER`.

`int_t domain_spf (stringer_t * domain)`

Determine whether SPF has been configured for a specified domain.

Parameters:

domain a pointer to a managed string containing the name of the domain to be queried.

Returns:

-1 on failure or if the domain wasn't found, 0 if SPF is disabled, or 1 if SPF is actively configured for the domain.

Definition at line 154 of file domains.c.

References `inx_find()`, and `M_TYPE_STRINGER`.

bool_t domain_start (void)

Fetch and store the list of configured domains from the database.

Returns:

true if the domain retrieval was successful, or false on error.

Definition at line 33 of file domains.c.

References `warehouse_fetch_domains()`.

Referenced by `warehouse_start()`.

void domain_stop (void)

Free the global list of domains.

Returns:

This function returns no value.

Definition at line 21 of file domains.c.

References `inx_cleanup()`.

Referenced by `warehouse_stop()`.

int_t domain_wildcard (stringer_t * domain)

Determine whether a specified domain has wildcards enabled.

Parameters:

domain a pointer to a managed string containing the name of the domain to be queried.

Returns:

-1 on failure or if the domain wasn't found, 0 if wildcards are disabled, or 1 if the domain has wildcards enabled.

Definition at line 117 of file domains.c.

References `inx_find()`, and `M_TYPE_STRINGER`.

Referenced by `smtp_check_authorized_from()`, and `smtp_fetch_inbound()`.

Variable Documentation

inx_t* domains = NULL

Definition at line 15 of file domains.c.

magma/objects/warehouse/patterns.c File Reference

Functions used to manage the list of spam patterns that are scanned for detection in outbound messages.

```
#include "magma.h"
```

Functions

- **int_t pattern_check** (stringer_t *message)
- *Check to see if any of the entries in the patterns list are found in a body of text.* void **pattern_update** (void)
- *Update the patterns list from the database, but no more frequently than once daily.* void **pattern_stop** (void)
- *Destroy the patterns list.* **bool_t pattern_start** (void)

Initialize the patterns list. Variables

- **inx_t * patterns_list** = NULL
- **uint64_t patterns_stamp** = 0
- **pthread_mutex_t patterns_mutex** = PTHREAD_MUTEX_INITIALIZER

Detailed Description

Functions used to manage the list of spam patterns that are scanned for detection in outbound messages.

Definition in file **patterns.c**.

Function Documentation

int_t pattern_check (stringer_t * message)

Check to see if any of the entries in the patterns list are found in a body of text.

patterns.c

Parameters:

message a managed string containing the raw data of the text to be searched.

Returns:

-2 on pattern match, -1 if an error occurs, or 1 if none of the patterns in the patterns list were detected.

Definition at line 24 of file patterns.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `mutex_lock()`, `mutex_unlock()`, `patterns_mutex`, `st_search_ci()`, and `stats_adjust_by_name()`.

Referenced by `portal_outbound_checks()`, and `smtp_data_outbound()`.

bool_t pattern_start (void)

Initialize the patterns list.

Returns:

This function will always return true.
Definition at line 114 of file patterns.c.
References pattern_update().
Referenced by warehouse_start().

void pattern_stop (void)

Destroy the patterns list.

Returns:

This function returns no value.
Definition at line 99 of file patterns.c.
References inx_cleanup(), mutex_lock(), mutex_unlock(), and patterns_mutex.
Referenced by warehouse_stop().

void pattern_update (void)

Update the patterns list from the database, but no more frequently than once daily.

Returns:

This function returns no value.
Definition at line 67 of file patterns.c.
References inx_cleanup(), mutex_lock(), mutex_unlock(), patterns_mutex, patterns_stamp, time_datestamp(), and warehouse_fetch_patterns().
Referenced by pattern_start(), and warehouse_update().

Variable Documentation**inx_t* patterns_list = NULL**

Definition at line 15 of file patterns.c.

pthread_mutex_t patterns_mutex = PTHREAD_MUTEX_INITIALIZER

Definition at line 17 of file patterns.c.
Referenced by pattern_check(), pattern_stop(), and pattern_update().

uint64_t patterns_stamp = 0

Definition at line 16 of file patterns.c.
Referenced by pattern_update().

magma/objects/warehouse/warehouse.c File Reference

The warehouse management functions.

```
#include "magma.h"
```

Functions

- void **warehouse_update** (void)
- *Update the warehouse components.* void **warehouse_stop** (void)
- *Stop the warehouse facility.* **bool_t warehouse_start** (void)

Start the warehouse facility.

Detailed Description

The warehouse management functions.

Definition in file **warehouse.c**.

Function Documentation

bool_t warehouse_start (void)

Start the warehouse facility.

warehouse.c

Note:

This will initialize the domains and patterns lists.

Returns:

false if any of the warehouse components failed to load, or true on success.

Definition at line 45 of file warehouse.c.

References domain_start(), pattern_start(), and warehouse_stop().

Referenced by process_start().

void warehouse_stop (void)

Stop the warehouse facility.

Note:

This will destroy the patterns and domain lists.

Returns:

This function returns no value.

Definition at line 32 of file warehouse.c.

References domain_stop(), and pattern_stop().

Referenced by process_stop(), and warehouse_start().

void warehouse_update (void)

Update the warehouse components.

Note:

This will update the patterns list.

Returns:

This function returns no value.

Definition at line 20 of file warehouse.c.

References `pattern_update()`.

Referenced by `process_maint()`.

magma/objects/warehouse/warehouse.h File Reference

Functions to provide access to warehoused reference data needed to make intelligent decisions.

Data Structures

- struct `__attribute__`

Functions

- `inx_t * warehouse_fetch_domains` (void)
- *datatier.c* `inx_t * warehouse_fetch_patterns` (void)
- *Fetch the pattern list from the database.* `domain_t * domain_alloc` (`stringer_t *domain`, `int_t restricted`, `int_t mailboxes`, `int_t wildcard`, `int_t dkim`, `int_t spf`)
- *domains.c* `int_t domain_dkim` (`stringer_t *domain`)
- *TODO: Eliminate dkim+spf and replace them with a `sign` flag.* `int_t domain_mailboxes` (`stringer_t *domain`)
- *Determine whether mailboxes are hosted locally for a specified domain.* `int_t domain_restricted` (`stringer_t *domain`)
- *Determine whether a specified domain is restricted to authenticated users.* `int_t domain_spf` (`stringer_t *domain`)
- *Determine whether SPF has been configured for a specified domain.* `bool_t domain_start` (void)
- *Fetch and store the list of configured domains from the database.* `void domain_stop` (void)
- *Free the global list of domains.* `int_t domain_wildcard` (`stringer_t *domain`)
- *Determine whether a specified domain has wildcards enabled.* `int_t pattern_check` (`stringer_t *message`)
- *patterns.c* `bool_t pattern_start` (void)
- *Initialize the patterns list.* `void pattern_stop` (void)
- *Destroy the patterns list.* `void pattern_update` (void)
- *Update the patterns list from the database, but no more frequently than once daily.* `bool_t warehouse_start` (void)
- *warehouse.c* `void warehouse_stop` (void)
- *Stop the warehouse facility.* `void warehouse_update` (void)

Update the warehouse components.

Detailed Description

Functions to provide access to warehoused reference data needed to make intelligent decisions.

Definition in file `warehouse.h`.

Function Documentation

`domain_t* domain_alloc` (`stringer_t * domain`, `int_t restricted`, `int_t mailboxes`, `int_t wildcard`, `int_t dkim`, `int_t spf`)

`domains.c`

`domains.c`

Parameters:

domain a pointer to a managed string containing the name of the specified domain.

restricted if set, indicate that the domain is restricted to authenticated users only.
mailboxes if set, indicate that mailboxes are hosted locally for the domain.
wildcard if set, indicate that wildcards are enabled for the domain.
dkim if set, indicate that outbound messages for the domain should be signed via DKIM.
spf if set, indicate that SPF is configured for the domain.

Returns:

NULL on failure or a pointer to the newly initialized domain object on success.

Definition at line 52 of file domains.c.

References align(), FOREIGNDATA, JOINED, log_pedantic, mm_alloc(), mm_copy(), PLACER_T, placer_t, st_data_get(), st_length_get(), and STACK.

Referenced by warehouse_fetch_domains().

int_t domain_dkim (stringer_t * domain)

TODO: Eliminate dkim+spf and replace them with a `sign` flag.

Determine whether outbound messages should be signed via DKIM for a specified domain.

Parameters:

domain a pointer to a managed string containing the name of the domain to be queried.

Returns:

-1 on failure or if the domain wasn't found, 0 if DKIM signing is disabled, or 1 if outbound messages should be signed via DKIM for the domain.

Definition at line 136 of file domains.c.

References inx_find(), and M_TYPE_STRINGER.

int_t domain_mailboxes (stringer_t * domain)

Determine whether mailboxes are hosted locally for a specified domain.

Parameters:

domain a pointer to a managed string containing the name of the domain to be queried.

Returns:

-1 on failure or if the domain was not found, 0 if the domain is foreign, or 1 if the mailbox is hosted locally for the domain.

Definition at line 81 of file domains.c.

References inx_find(), and M_TYPE_STRINGER.

int_t domain_restricted (stringer_t * domain)

Determine whether a specified domain is restricted to authenticated users.

Parameters:

domain a pointer to a managed string containing the name of the domain to be queried.

Returns:

-1 on failure or if the domain was not found, 0 if restricted relay is disabled, or 1 if the domain is restricted to authenticated users only.

Definition at line 99 of file domains.c.

References `inx_find()`, and `M_TYPE_STRINGER`.

`int_t domain_spf (stringer_t * domain)`

Determine whether SPF has been configured for a specified domain.

Parameters:

domain a pointer to a managed string containing the name of the domain to be queried.

Returns:

-1 on failure or if the domain wasn't found, 0 if SPF is disabled, or 1 if SPF is actively configured for the domain.

Definition at line 154 of file domains.c.

References `inx_find()`, and `M_TYPE_STRINGER`.

`bool_t domain_start (void)`

Fetch and store the list of configured domains from the database.

Returns:

true if the domain retrieval was successful, or false on error.

Definition at line 33 of file domains.c.

References `warehouse_fetch_domains()`.

Referenced by `warehouse_start()`.

`void domain_stop (void)`

Free the global list of domains.

Returns:

This function returns no value.

Definition at line 21 of file domains.c.

References `inx_cleanup()`.

Referenced by `warehouse_stop()`.

`int_t domain_wildcard (stringer_t * domain)`

Determine whether a specified domain has wildcards enabled.

Parameters:

domain a pointer to a managed string containing the name of the domain to be queried.

Returns:

-1 on failure or if the domain wasn't found, 0 if wildcards are disabled, or 1 if the domain has wildcards enabled.

Definition at line 117 of file domains.c.

References `inx_find()`, and `M_TYPE_STRINGER`.

Referenced by `smtp_check_authorized_from()`, and `smtp_fetch_inbound()`.

int_t pattern_check (stringer_t * message)

patterns.c

patterns.c

Parameters:

message a managed string containing the raw data of the text to be searched.

Returns:

-2 on pattern match, -1 if an error occurs, or 1 if none of the patterns in the patterns list were detected.

Definition at line 24 of file patterns.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `mutex_lock()`, `mutex_unlock()`, `patterns_mutex`, `st_search_ci()`, and `stats_adjust_by_name()`.

Referenced by `portal_outbound_checks()`, and `smtp_data_outbound()`.

bool_t pattern_start (void)

Initialize the patterns list.

Returns:

This function will always return true.

Definition at line 114 of file patterns.c.

References `pattern_update()`.

Referenced by `warehouse_start()`.

void pattern_stop (void)

Destroy the patterns list.

Returns:

This function returns no value.

Definition at line 99 of file patterns.c.

References `inx_cleanup()`, `mutex_lock()`, `mutex_unlock()`, and `patterns_mutex`.

Referenced by `warehouse_stop()`.

void pattern_update (void)

Update the patterns list from the database, but no more frequently than once daily.

Returns:

This function returns no value.

Definition at line 67 of file patterns.c.

References `inx_cleanup()`, `mutex_lock()`, `mutex_unlock()`, `patterns_mutex`, `patterns_stamp`, `time_datestamp()`, and `warehouse_fetch_patterns()`.

Referenced by `pattern_start()`, and `warehouse_update()`.

inx_t* warehouse_fetch_domains (void)

datatier.c

datatier.c

Returns:

NULL on failure, or an `inx` object holding all the configured domains on success.

Definition at line 19 of file datatier.c.

References `domain_alloc()`, `inx_alloc()`, `inx_free()`, `inx_insert()`, `log_check`, `log_info`, `log_pedantic`, `M_INX_TREE`, `M_TYPE_STRINGER`, `MAGMA_HOSTNAME_MAX`, `mm_free()`, `PLACER`, `res_field_block()`, `res_field_int8()`, `res_field_length()`, `res_row_next()`, `res_table_free()`, `multi_t::st`, `stmt_get_result()`, and `multi_t::val`.

Referenced by `domain_start()`.

inx_t* warehouse_fetch_patterns (void)

Fetch the pattern list from the database.

Returns:

NULL on failure or a pointer to an `inx` holder containing a collection of managed strings with the patterns on success.

Definition at line 69 of file datatier.c.

References `inx_alloc()`, `inx_free()`, `inx_insert()`, `log_pedantic`, `M_INX_LINKED`, `M_TYPE_UINT64`, `res_field_string()`, `res_row_next()`, `res_table_free()`, `st_free()`, `stmt_get_result()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `pattern_update()`.

bool_t warehouse_start (void)

warehouse.c

warehouse.c

Note:

This will initialize the domains and patterns lists.

Returns:

false if any of the warehouse components failed to load, or true on success.

Definition at line 45 of file warehouse.c.

References `domain_start()`, `pattern_start()`, and `warehouse_stop()`.

Referenced by `process_start()`.

void warehouse_stop (void)

Stop the warehouse facility.

Note:

This will destroy the patterns and domain lists.

Returns:

This function returns no value.

Definition at line 32 of file warehouse.c.

References domain_stop(), and pattern_stop().

Referenced by process_stop(), and warehouse_start().

void warehouse_update (void)

Update the warehouse components.

Note:

This will update the patterns list.

Returns:

This function returns no value.

Definition at line 20 of file warehouse.c.

References pattern_update().

Referenced by process_maint().

magma/providers/checkers/allocations.h File Reference

Functions used to scan, analyze, check, and validate data.

Variables

- struct {
 - uint32_t status
 - uint32_t owner
 - uint32_t weight
 - chr_t * comment
 - } ip_v4_allocations_t []
-

Detailed Description

Functions used to scan, analyze, check, and validate data.

Allocation	Data
http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml	
http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.txt	
Network	Popularity
http://www.spamcop.net/w3m?action=map	
http://www.spamcop.net/w3m?action=map;format=text	
Last Updated 2010/11/09	
Definition in file allocations.h .	

Variable Documentation

chr_t* comment

Definition at line 34 of file allocations.h.

Referenced by mail_extract_address().

struct { ... } ip_v4_allocations_t[]

uint32_t owner

Definition at line 33 of file allocations.h.

uint32_t status

Definition at line 33 of file allocations.h.

Referenced by `__attribute__()`, `client_read()`, `client_read_line()`, `con_read()`, `con_read_line()`, `con_reverse_check()`, `dequeue()`, `dkim_check()`, `dkim_create()`, `http_requeue()`, `imap_examine()`, `imap_fetch()`, `imap_message_copier()`, `imap_parse_address_breaker()`, `imap_requeue()`, `imap_search()`, `imap_search_messages()`, `imap_select()`, `imap_status()`, `mail_add_forward_headers()`, `meta_messages_copier()`, `net_listen()`, `pop_requeue()`, `portal_endpoint_config_edit()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_edit()`, `portal_endpoint_contacts_move()`, `portal_endpoint_contacts_remove()`, `process_maint()`, `process_stop()`, `queue_signal()`, `signal_status()`, `smtp_data_finish()`, `smtp_data_read()`, `smtp_requeue()`, and `smtp_store_message()`.

uint32_t weight

Definition at line 33 of file `allocations.h`.

magma/providers/checkers/checkers.h File Reference

Functions used to scan, analyze, check, and validate data.

```
#include "network/network.h"
```

Defines

- `#define GET_IP_T_DEFINITION`
- `#define IP_RANDOMIZER_POOL 1024`
- `#define IP_RANDOMIZER_PUSH_MIN 4`
- `#define IP_RANDOMIZER_PUSH_MAX 16`

Enumerations

- `enum { UNALLOCATED = 0, ALLOCATED = 1, RESERVED = 2 }`
- `enum { REGISTRY = 1, DIRECT = 2 }`
- `enum { DOMESTIC = 0, FOREIGN = 1 }`

Functions

- `bool_t lib_load_clamav (void)`
- `clamav.c bool_t virus_start (void)`
- `const char * lib_version_clamav (void)`
- `int virus_engine_refresh (void)`
- `int virus_check (stringer_t *data)`
- *Virus scan a block of data.* `struct cl_engine * virus_engine_create (uint64_t *signatures)`
- `uint64_t virus_sigs_loaded (void)`
- *Get the number of virus signatures loaded by the ClamAV engine context.* `uint64_t virus_sigs_total (void)`
- *Get the number of official signatures available inside the ClamAV signature directory.* `void virus_engine_destroy (struct cl_engine **target)`
- *Destroy a ClamAV engine context.* `void virus_stop (void)`
- *Shut down the ClamAV library and free all its data and temporary files.* `int_t dkim_check (stringer_t *id, stringer_t *message)`
- `dkim.c stringer_t * dkim_create (stringer_t *id, stringer_t *message)`
- *Generate a DKIM signature for a message.* `void * dkim_memory_alloc (void *closure, size_t nbytes)`
- *A memory allocation routine stub for dkim.* `void dkim_memory_free (void *closure, void *ptr)`
- *A memory free routine stub for dkim.* `bool_t dkim_start (void)`
- *Start the dkim engine.* `void dkim_stop (void)`
- *Stop the dkim engine.* `bool_t lib_load_dkim (void)`
- *Initialize the dkim library and bind dynamically to the exported functions that are required.* `const chr_t * lib_version_dkim (void)`
- *Return the version string of the dkim library.* `int_t dspam_check (uint64_t usernum, stringer_t *message, stringer_t **signature)`
- `dspam.c bool_t dspam_start (void)`
- `void dspam_stop (void)`
- `bool_t dspam_train (uint64_t usernum, int_t disposition, stringer_t *signature)`
- `bool_t lib_load_dspam (void)`
- *Initialize the dspam library and bind dynamically to the exported functions that are required.* `chr_t * lib_version_dspam (void)`
- *Return the version string of the dspam library.* `bool_t lib_load_spf (void)`
- `spf.c const chr_t * lib_version_spf (void)`
- *Return the version string of the spf library.* `bool_t spf_start (void)`
- *Initialize the pool of spf server connections.* `int_t spf_check (ip_t *ip, stringer_t *helo, stringer_t *mailfrom)`

- *Validate an smtp request via spf.* void **spf_stop** (void)
Destroy the spf connection pool.
-

Detailed Description

Functions used to scan, analyze, check, and validate data.

Definition in file **checkers.h**.

Define Documentation

#define GET_IP_T_DEFINITION

Definition at line 16 of file checkers.h.

#define IP_RANDOMIZER_POOL 1024

Definition at line 19 of file checkers.h.

#define IP_RANDOMIZER_PUSH_MAX 16

Definition at line 22 of file checkers.h.

#define IP_RANDOMIZER_PUSH_MIN 4

Definition at line 21 of file checkers.h.

Enumeration Type Documentation

anonymous enum

Enumerator:

UNALLOCATED
ALLOCATED
RESERVED

Definition at line 24 of file checkers.h.

anonymous enum

Enumerator:

REGISTRY
DIRECT

Definition at line 30 of file checkers.h.

anonymous enum

Enumerator:

DOMESTIC
FOREIGN

Definition at line 35 of file checkers.h.

Function Documentation

int_t dkim_check (stringer_t * *id*, stringer_t * *message*)

dkim.c

dkim.c

Note:

This function also updates the provider.dkim.* statistics.

Parameters:

id a managed string containing a printable string id for this message.

message a managed string containing the mail message data.

Returns:

1 if the dkim verification was successful, or < 0 otherwise. -1: General internal or dkim-related failure occurred, or no signature was present. -2: The signature was bad or the signing key was revoked or key retrieval failed (try again later).

Definition at line 176 of file dkim.c.

References dkim_chunk_d, dkim_engine, dkim_eom_d, dkim_free_d, dkim_getresultstr_d, dkim_verify_d, log_pedantic, st_data_get(), st_length_get(), stats_adjust_by_name(), and status.

Referenced by smtp_accept_message().

stringer_t* dkim_create (stringer_t * *id*, stringer_t * *message*)

Generate a DKIM signature for a message.

Note:

For this function to work, the **magma.dkim.enabled** configuration option must be set. This function also updates the provider.dkim.signed statistic.

Parameters:

id a managed string containing a printable string id for this message.

message a managed string containing the mail message data.

Returns:

NULL on failure or if **magma.dkim.enabled** is false; otherwise, a managed string containing a DKIM-Signature header built using the dkim signature that was generated for the input message.

LOW: Should we track signing errors too?

Definition at line 118 of file dkim.c.

References CONTIGUOUS, magma_t::dkim, dkim_chunk_d, dkim_engine, dkim_eom_d, dkim_free_d, dkim_getresultstr_d, dkim_getsighdrx_d, DKIM_PROCESS_ALL, dkim_sign_d, magma_t::domain, magma_t::enabled, HEAP, log_pedantic, magma, NULLER_T, magma_t::privkey, magma_t::selector, st_alloc_opts(), st_cleanup(), st_data_get(), st_length_get(), st_merge, stats_adjust_by_name(), and status.

Referenced by mail_add_forward_headers(), mail_add_outbound_headers(), smtp_bounce(), and smtp_reply().

void* dkim_memory_alloc (void * *closure*, size_t *nbytes*)

A memory allocation routine stub for dkim.

Note:

This function is passed to dkim_init().

Parameters:

closure a void pointer to a memory closure passed to dkim_sign() and dkim_verify().

nbytes the size, in bytes, of the memory block to be allocated.

Returns:

a pointer to a newly allocated block of memory of specified size.

Definition at line 62 of file dkim.c.

References mm_alloc().

Referenced by dkim_start().

void dkim_memory_free (void * *closure*, void * *ptr*)

A memory free routine stub for dkim.

Note:

This function is passed to dkim_init().

Parameters:

closure a void pointer to a memory closure passed to dkim_sign() and dkim_verify().

ptr a pointer to the block of memory to be released.

Returns:

This function returns no value.

Definition at line 74 of file dkim.c.

References mm_free().

Referenced by dkim_start().

bool_t dkim_start (void)

Start the dkim engine.

Returns:

false on failure or true on success.

Definition at line 84 of file dkim.c.

References dkim_engine, dkim_init_d, dkim_memory_alloc(), dkim_memory_free(), and log_pedantic.

Referenced by process_start().

void dkim_stop (void)

Stop the dkim engine.

Returns:

This function returns no value.

Definition at line 98 of file dkim.c.

References dkim_close_d, and dkim_engine.

Referenced by process_stop().

int_t dspam_check (uint64_t usernum, stringer_t * message, stringer_t ** signature)

dspam.c

HIGH: Return a result structure with the disposition, confidence, probability and signature data. Then store the analysis result with the message. Either in the DB or by adding a custom header to the message. Return codes should become 0 for success, or a negative integer for errors. Add statistical updates to check and train functions. Result: result=\"%s\"; class=\"%s\"; probability=01.4f; confidence=02.2f; signature=lu; key=lu; Layman's terms: How spammy is this message

Definition at line 59 of file dspam.c.

References dspam_attach_d, dspam_create_d, dspam_destroy_d, dspam_detach_d, dspam_process_d, log_error, log_info, log_pedantic, MAGMA_SPOOL_DATA, PL_RESERVED, pool_get_obj(), pool_pull(), pool_release(), spool_path(), sql_ping(), st_char_get(), st_free(), st_import(), and stmt_rebuild().

Referenced by smtp_accept_message().

bool_t dspam_start (void)

Definition at line 43 of file dspam.c.

References dspam_init_driver_d.

Referenced by process_start().

void dspam_stop (void)

Definition at line 50 of file dspam.c.

References dspam_shutdown_driver_d.

Referenced by process_stop().

bool_t dspam_train (uint64_t usernum, int_t disposition, stringer_t * signature)

Note:

The disposition parameter marks whether or not the signature is currently marked as junk. So training the signature means that it will toggle its status.

Parameters:

usenum the numerical id of the user making the spam training request.

disposition if 0, dspam will mark the signature as junk; otherwise, mark as OK.

signature a managed string containing the spam signature to be trained.

Returns:

true on success or false on failure.

Definition at line 178 of file dspam.c.

References dspam_attach_d, dspam_create_d, dspam_destroy_d, dspam_detach_d, dspam_process_d, log_info, log_pedantic, MAGMA_SPOOL_DATA, PL_RESERVED, pool_get_obj(), pool_pull(), pool_release(), spool_path(), sql_ping(), st_char_get(), st_free(), st_length_get(), and stmt_rebuild().

Referenced by teacher_process().

bool_t lib_load_clamav (void)

clamav.c

clamav.c

Returns:

false on failure or true on success.

Definition at line 464 of file clamav.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

bool_t lib_load_dkim (void)

Initialize the dkim library and bind dynamically to the exported functions that are required.

Returns:

true on success or false on failure.

Definition at line 32 of file dkim.c.

References dkim_getsighdrx_d, dkim_libversion_d, dkim_version, lib_symbols(), and M_BIND.

Referenced by lib_load().

bool_t lib_load_dspam (void)

Initialize the dspam library and bind dynamically to the exported functions that are required.

Returns:

true on success or false on failure.

Definition at line 29 of file dspam.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

bool_t lib_load_spf (void)

spf.c

spf.c

Returns:

false on failure or true on success.

Definition at line 32 of file spf.c.

References lib_symbols(), M_BIND, SPF_get_lib_version_d, and spf_version.

Referenced by lib_load().

const char* lib_version_clamav (void)

Returns the version of ClamAV that was loaded at runtime.

Returns:

The ClamAV version as a constant string.

Definition at line 456 of file clamav.c.

References cl_retver_d.

Referenced by lib_load().

const chr_t* lib_version_dkim (void)

Return the version string of the dkim library.

Returns:

a pointer to a character string containing the dkim library version information.

Definition at line 24 of file dkim.c.

References dkim_version.

Referenced by lib_load().

chr_t* lib_version_dspam (void)

Return the version string of the dspam library.

Returns:

a pointer to a character string containing the dspam library version information.

Definition at line 21 of file dspam.c.

References dspam_version_d.

Referenced by lib_load().

const chr_t* lib_version_spf (void)

Return the version string of the spf library.

Returns:

a pointer to a character string containing the spf library version information.
Definition at line 24 of file spf.c.
References spf_version.
Referenced by lib_load().

int_t spf_check (ip_t * ip, stringer_t * helo, stringer_t * mailfrom)

Validate an smtp request via spf.

Parameters:

ip an ip address object containing the ip address of the connection to be checked.
helo a managed string containing the client supplied HELO request value.
mailfrom a managed string containing the client supplied MAIL FROM request value. TODO: We really need to change these return values to account for 0

Returns:

-2 on spf check fail, -1 on other failure, and 1 on spf pass.
Definition at line 117 of file spf.c.
References ip_t::family, ip_t::ip4, ip_t::ip6, log_pedantic, mail_domain_get(), PL_RESERVED, placer_t, pool_get_obj(), pool_pull(), pool_release(), SPF_request_free_d, SPF_request_new_d, SPF_request_query_mailfrom_d, SPF_request_set_env_from_d, SPF_request_set_helo_dom_d, SPF_request_set_ipv4_d, SPF_request_set_ipv6_d, SPF_response_free_d, SPF_response_reason_d, SPF_response_result_d, SPF_strerror_d, SPF_streason_d, SPF_strerror_d, st_char_get(), st_length_int(), and stats_adjust_by_name().
Referenced by smtp_rcpt_to().

bool_t spf_start (void)

Initialize the pool of spf server connections.

Returns:

false on failure or true on success.
Definition at line 57 of file spf.c.
References magma_t::iface, log_pedantic, magma, pool_alloc(), pool_set_obj(), magma_t::spf, and SPF_server_new_d.
Referenced by process_start().

void spf_stop (void)

Destroy the spf connection pool.

Returns:

This function returns no value.
Definition at line 89 of file spf.c.
References magma_t::iface, magma, pool_free(), pool_get_obj(), magma_t::spf, and SPF_server_free_d.

Referenced by process_stop().

int virus_check (stringer_t * *data*)

Virus scan a block of data.

Parameters:

data a managed string containing the block of data to be scanned.

Returns:

1 if the message passed the scan, or < 0 on failure. -1: general failure, or the virus scanner was not enabled.
-2: the data matches a worm, trojan, or virus. -3: the data matches a phishing attempt.

Definition at line 366 of file clamav.c.

References cl_scandesc_d, cl_strerror_d, CONSTANT, magma_t::iface, log_error, log_pedantic, magma, MAGMA_SPOOL_SCAN, ns_length_get(), PLACER, spool_mktemp(), st_cmp_ci_starts(), st_data_get(), st_length_get(), stats_increment_by_name(), magma_t::virus, virus_engine, and virus_lock.

Referenced by portal_outbound_checks(), smtp_accept_message(), and smtp_data_outbound().

struct cl_engine* virus_engine_create (uint64_t * *signatures*) [read]

Generates a new ClamAV engine context.

Parameters:

signatures An optional pointer which will be used to record the number of signatures loaded.

Returns:

Returns a pointer to the newly created context or NULL if an error occurs.

Definition at line 91 of file clamav.c.

References cl_engine_compile_d, cl_engine_free_d, cl_engine_new_d, cl_engine_set_num_d, cl_engine_set_str_d, cl_load_d, cl_strerror_d, magma_t::iface, log_error, magma, st_char_get(), magma_t::virus, and virus_spool.

Referenced by virus_engine_refresh(), and virus_start().

void virus_engine_destroy (struct cl_engine ** *target*)

Destroy a ClamAV engine context.

Parameters:

target the address of a pointer to a ClamAV engine context to be freed and reset.

Returns:

This function returns no value.

Definition at line 78 of file clamav.c.

References cl_engine_free_d, and log_check.

Referenced by virus_engine_refresh(), and virus_stop().

int virus_engine_refresh (void)

Checks the virus database directory for new signatures. If new signatures are detected an updated ClamAV engine context is created.

Returns:

Returns 1 if the engine context is updated, 0 if no updates are necessary and -1 in the event of an error.
Definition at line 296 of file clamav.c.

References `cl_statchkdir_d`, `cl_statfree_d`, `cl_statinidir_d`, `magma_t::iface`, `log_error`, `log_info`, `magma`, `mm_wipe()`, `stats_increment_by_name()`, `stats_set_by_name()`, `magma_t::virus`, `virus_engine`, `virus_engine_create()`, `virus_engine_destroy()`, `virus_lock`, `virus_sigs`, `virus_sigs_total()`, and `virus_stat`.

Referenced by `process_maint()`.

uint64_t virus_sigs_loaded (void)

Get the number of virus signatures loaded by the ClamAV engine context.

Returns:

the number of virus signatures loaded by the ClamAV engine context.
Definition at line 44 of file clamav.c.

References `virus_lock`, and `virus_sigs`.

uint64_t virus_sigs_total (void)

Get the number of official signatures available inside the ClamAV signature directory.

See also:

`magma.iface.virus.signatures`

Returns:

the number of official signatures available inside the ClamAV signature directory, or 0 on failure.
Definition at line 60 of file clamav.c.

References `cl_countsigs_d`, `cl_strerror_d`, `magma_t::iface`, `log_error`, `magma`, and `magma_t::virus`.

Referenced by `virus_engine_refresh()`, and `virus_start()`.

bool_t virus_start (void)

Initializes the global ClamAV engine context and configures it appropriately.

Returns:

Returns true if the ClamAV engine was loaded correctly.
Definition at line 201 of file clamav.c.

References `cl_init_d`, `cl_statfree_d`, `cl_statinidir_d`, `cl_strerror_d`, `magma_t::iface`, `log_critical`, `log_pedantic`, `magma`, `MAGMA_SPOOL_SCAN`, `mm_wipe()`, `magma_t::spool`, `spool_check()`, `spool_path()`, `st_char_get()`, `st_length_int()`, `stats_increment_by_name()`, `stats_set_by_name()`, `magma_t::virus`, `virus_engine`, `virus_engine_create()`, `virus_sigs`, `virus_sigs_total()`, `virus_spool`, and `virus_stat`.

Referenced by `process_start()`.

void virus_stop (void)

Shut down the ClamAV library and free all its data and temporary files.

Returns:

This function returns no value.

Definition at line 254 of file clamav.c.

References `cl_shutdown_d`, `cl_statfree_d`, `magma_t::iface`, `magma`, `st_free()`, `stats_set_by_name()`, `magma_t::virus`, `virus_engine`, `virus_engine_destroy()`, `virus_sigs`, `virus_spool`, and `virus_stat`.

Referenced by `process_stop()`.

magma/providers/checkers/clamav.c File Reference

Interface to the ClamAV library.

```
#include "magma.h"
```

Functions

- uint64_t **virus_sigs_loaded** (void)
- *Get the number of virus signatures loaded by the ClamAV engine context.* uint64_t **virus_sigs_total** (void)
- *Get the number of official signatures available inside the ClamAV signature directory.* void **virus_engine_destroy** (struct cl_engine **target)
- *Destroy a ClamAV engine context.* struct cl_engine * **virus_engine_create** (uint64_t *signatures)
- **bool_t virus_start** (void)
- void **virus_stop** (void)
- *Shut down the ClamAV library and free all its data and temporary files.* int **virus_engine_refresh** (void)
- int **virus_check** (stringer_t *data)
- *Virus scan a block of data.* const char * **lib_version_clamav** (void)
- **bool_t lib_load_clamav** (void)

Loads the external functions needed by the ClamAV interface. Variables

- stringer_t * **virus_spool** = NULL
- struct cl_stat **virus_stat**
- unsigned int **virus_sigs** = 0
- struct cl_engine * **virus_engine** = NULL
- pthread_rwlock_t **virus_lock** = PTHREAD_RWLOCK_INITIALIZER

Detailed Description

Interface to the ClamAV library.

Definition in file **clamav.c**.

Function Documentation

bool_t lib_load_clamav (void)

Loads the external functions needed by the ClamAV interface.

clamav.c

Returns:

false on failure or true on success.

Definition at line 464 of file clamav.c.

References `lib_symbols()`, and `M_BIND`.

Referenced by `lib_load()`.

const char* lib_version_clamav (void)

Returns the version of ClamAV that was loaded at runtime.

Returns:

The ClamAV version as a constant string.

Definition at line 456 of file clamav.c.

References `cl_retver_d`.

Referenced by `lib_load()`.

int virus_check (stringer_t * data)

Virus scan a block of data.

Parameters:

data a managed string containing the block of data to be scanned.

Returns:

1 if the message passed the scan, or < 0 on failure. -1: general failure, or the virus scanner was not enabled.

-2: the data matches a worm, trojan, or virus. -3: the data matches a phishing attempt.

Definition at line 366 of file clamav.c.

References `cl_scandesc_d`, `cl_strerror_d`, `CONSTANT`, `magma_t::iface`, `log_error`, `log_pedantic`, `magma`, `MAGMA_SPOOL_SCAN`, `ns_length_get()`, `PLACER`, `spool_mktemp()`, `st_cmp_ci_starts()`, `st_data_get()`, `st_length_get()`, `stats_increment_by_name()`, `magma_t::virus`, `virus_engine`, and `virus_lock`.

Referenced by `portal_outbound_checks()`, `smtp_accept_message()`, and `smtp_data_outbound()`.

struct cl_engine* virus_engine_create (uint64_t * signatures) [read]

Generates a new ClamAV engine context.

Parameters:

signatures An optional pointer which will be used to record the number of signatures loaded.

Returns:

Returns a pointer to the newly created context or NULL if an error occurs.

Definition at line 91 of file clamav.c.

References `cl_engine_compile_d`, `cl_engine_free_d`, `cl_engine_new_d`, `cl_engine_set_num_d`, `cl_engine_set_str_d`, `cl_load_d`, `cl_strerror_d`, `magma_t::iface`, `log_error`, `magma`, `st_char_get()`, `magma_t::virus`, and `virus_spool`.

Referenced by `virus_engine_refresh()`, and `virus_start()`.

void virus_engine_destroy (struct cl_engine ** target)

Destroy a ClamAV engine context.

Parameters:

target the address of a pointer to a ClamAV engine context to be freed and reset.

Returns:

This function returns no value.

Definition at line 78 of file clamav.c.

References `cl_engine_free_d`, and `log_check`.

Referenced by `virus_engine_refresh()`, and `virus_stop()`.

int virus_engine_refresh (void)

Checks the virus database directory for new signatures. If new signatures are detected an updated ClamAV engine context is created.

Returns:

Returns 1 if the engine context is updated, 0 if no updates are necessary and -1 in the event of an error.

Definition at line 296 of file `clamav.c`.

References `cl_statchkdir_d`, `cl_statfree_d`, `cl_statinidir_d`, `magma_t::iface`, `log_error`, `log_info`, `magma`, `mm_wipe()`, `stats_increment_by_name()`, `stats_set_by_name()`, `magma_t::virus`, `virus_engine`, `virus_engine_create()`, `virus_engine_destroy()`, `virus_lock`, `virus_sigs`, `virus_sigs_total()`, and `virus_stat`.

Referenced by `process_maint()`.

uint64_t virus_sigs_loaded (void)

Get the number of virus signatures loaded by the ClamAV engine context.

Returns:

the number of virus signatures loaded by the ClamAV engine context.

Definition at line 44 of file `clamav.c`.

References `virus_lock`, and `virus_sigs`.

uint64_t virus_sigs_total (void)

Get the number of official signatures available inside the ClamAV signature directory.

See also:

`magma.iface.virus.signatures`

Returns:

the number of official signatures available inside the ClamAV signature directory, or 0 on failure.

Definition at line 60 of file `clamav.c`.

References `cl_countsigs_d`, `cl_strerror_d`, `magma_t::iface`, `log_error`, `magma`, and `magma_t::virus`.

Referenced by `virus_engine_refresh()`, and `virus_start()`.

bool_t virus_start (void)

Initializes the global ClamAV engine context and configures it appropriately.

Returns:

Returns true if the ClamAV engine was loaded correctly.

Definition at line 201 of file `clamav.c`.

References `cl_init_d`, `cl_statfree_d`, `cl_statinidir_d`, `cl_strerror_d`, `magma_t::iface`, `log_critical`, `log_pedantic`, `magma`, `MAGMA_SPOOL_SCAN`, `mm_wipe()`, `magma_t::spool`, `spool_check()`, `spool_path()`, `st_char_get()`, `st_length_int()`, `stats_increment_by_name()`, `stats_set_by_name()`, `magma_t::virus`, `virus_engine`, `virus_engine_create()`, `virus_sigs`, `virus_sigs_total()`, `virus_spool`, and `virus_stat`.

Referenced by process_start().

void virus_stop (void)

Shut down the ClamAV library and free all its data and temporary files.

Returns:

This function returns no value.

Definition at line 254 of file clamav.c.

References cl_shutdown_d, cl_statfree_d, magma_t::iface, magma, st_free(), stats_set_by_name(), magma_t::virus, virus_engine, virus_engine_destroy(), virus_sigs, virus_spool, and virus_stat.

Referenced by process_stop().

Variable Documentation

struct cl_engine* virus_engine = NULL

The virus engine context pointer.

Definition at line 33 of file clamav.c.

Referenced by virus_check(), virus_engine_refresh(), virus_start(), and virus_stop().

pthread_rwlock_t virus_lock = PTHREAD_RWLOCK_INITIALIZER

The virus engine read/write lock.

Definition at line 38 of file clamav.c.

Referenced by virus_check(), virus_engine_refresh(), and virus_sigs_loaded().

unsigned int virus_sigs = 0

The number of signatures loaded by the virus engine.

Definition at line 28 of file clamav.c.

Referenced by virus_engine_refresh(), virus_sigs_loaded(), virus_start(), and virus_stop().

stringer_t* virus_spool = NULL

The virus engine spool directory.

Definition at line 18 of file clamav.c.

Referenced by virus_engine_create(), virus_start(), and virus_stop().

struct cl_stat virus_stat

The status of the signatures directory.

Definition at line 23 of file clamav.c.

Referenced by virus_engine_refresh(), virus_start(), and virus_stop().

magma/providers/checkers/dkim.c File Reference

Functions used to generate and verify Domain Keys Identified Mail (DKIM).

```
#include "magma.h"
```

Defines

- `#define DKIM_PROCESS_ALL -1L`

Functions

- `const chr_t * lib_version_dkim` (void)
- *Return the version string of the dkim library.* `bool_t lib_load_dkim` (void)
- *Initialize the dkim library and bind dynamically to the exported functions that are required.* `void * dkim_memory_alloc` (void *closure, size_t nbytes)
- *A memory allocation routine stub for dkim.* `void dkim_memory_free` (void *closure, void *ptr)
- *A memory free routine stub for dkim.* `bool_t dkim_start` (void)
- *Start the dkim engine.* `void dkim_stop` (void)
- *Stop the dkim engine.* `stringer_t * dkim_create` (stringer_t *id, stringer_t *message)
- *Generate a DKIM signature for a message.* `int_t dkim_check` (stringer_t *id, stringer_t *message)

Perform dkim verification of a signed message. Variables

- `chr_t dkim_version` [8]
- `DKIM_LIB * dkim_engine` = NULL

Detailed Description

Functions used to generate and verify Domain Keys Identified Mail (DKIM).

Definition in file `dkim.c`.

Define Documentation

`#define DKIM_PROCESS_ALL -1L`

Definition at line 18 of file `dkim.c`.

Referenced by `dkim_create()`.

Function Documentation

`int_t dkim_check` (stringer_t * *id*, stringer_t * *message*)

Perform dkim verification of a signed message.

`dkim.c`

Note:

This function also updates the provider.dkim.* statistics.

Parameters:

id a managed string containing a printable string id for this message.

message a managed string containing the mail message data.

Returns:

1 if the dkim verification was successful, or < 0 otherwise. -1: General internal or dkim-related failure occurred, or no signature was present. -2: The signature was bad or the signing key was revoked or key retrieval failed (try again later).

Definition at line 176 of file dkim.c.

References dkim_chunk_d, dkim_engine, dkim_eom_d, dkim_free_d, dkim_getresultstr_d, dkim_verify_d, log_pedantic, st_data_get(), st_length_get(), stats_adjust_by_name(), and status.

Referenced by smtp_accept_message().

stringer_t* dkim_create (stringer_t * *id*, stringer_t * *message*)

Generate a DKIM signature for a message.

Note:

For this function to work, the **magma.dkim.enabled** configuration option must be set. This function also updates the provider.dkim.signed statistic.

Parameters:

id a managed string containing a printable string id for this message.

message a managed string containing the mail message data.

Returns:

NULL on failure or if **magma.dkim.enabled** is false; otherwise, a managed string containing a DKIM-Signature header built using the dkim signature that was generated for the input message.

LOW: Should we track signing errors too?

Definition at line 118 of file dkim.c.

References CONTIGUOUS, magma_t::dkim, dkim_chunk_d, dkim_engine, dkim_eom_d, dkim_free_d, dkim_getresultstr_d, dkim_getsighdrx_d, DKIM_PROCESS_ALL, dkim_sign_d, magma_t::domain, magma_t::enabled, HEAP, log_pedantic, magma, NULLER_T, magma_t::privkey, magma_t::selector, st_alloc_opts(), st_cleanup(), st_data_get(), st_length_get(), st_merge, stats_adjust_by_name(), and status.

Referenced by mail_add_forward_headers(), mail_add_outbound_headers(), smtp_bounce(), and smtp_reply().

void* dkim_memory_alloc (void * *closure*, size_t *nbytes*)

A memory allocation routine stub for dkim.

Note:

This function is passed to dkim_init().

Parameters:

closure a void pointer to a memory closure passed to dkim_sign() and dkim_verify().

nbytes the size, in bytes, of the memory block to be allocated.

Returns:

a pointer to a newly allocated block of memory of specified size.

Definition at line 62 of file dkim.c.

References mm_alloc().

Referenced by dkim_start().

void dkim_memory_free (void * *closure*, void * *ptr*)

A memory free routine stub for dkim.

Note:

This function is passed to dkim_init().

Parameters:

closure a void pointer to a memory closure passed to dkim_sign() and dkim_verify().

ptr a pointer to the block of memory to be released.

Returns:

This function returns no value.

Definition at line 74 of file dkim.c.

References mm_free().

Referenced by dkim_start().

bool_t dkim_start (void)

Start the dkim engine.

Returns:

false on failure or true on success.

Definition at line 84 of file dkim.c.

References dkim_engine, dkim_init_d, dkim_memory_alloc(), dkim_memory_free(), and log_pedantic.

Referenced by process_start().

void dkim_stop (void)

Stop the dkim engine.

Returns:

This function returns no value.

Definition at line 98 of file dkim.c.

References dkim_close_d, and dkim_engine.

Referenced by process_stop().

bool_t lib_load_dkim (void)

Initialize the dkim library and bind dynamically to the exported functions that are required.

Returns:

true on success or false on failure.

Definition at line 32 of file dkim.c.

References dkim_getsighdrx_d, dkim_libversion_d, dkim_version, lib_symbols(), and M_BIND.

Referenced by lib_load().

const chr_t* lib_version_dkim (void)

Return the version string of the dkim library.

Returns:

a pointer to a character string containing the dkim library version information.

Definition at line 24 of file dkim.c.

References dkim_version.

Referenced by lib_load().

Variable Documentation

DKIM_LIB* dkim_engine = NULL

Definition at line 16 of file dkim.c.

Referenced by dkim_check(), dkim_create(), dkim_start(), and dkim_stop().

chr_t dkim_version[8]

Definition at line 15 of file dkim.c.

Referenced by lib_load_dkim(), and lib_version_dkim().

magma/providers/checkers/dspam.c File Reference

DSPAM interface functions.

```
#include "magma.h"
```

Functions

- **chr_t * lib_version_dspam** (void)
- *Return the version string of the dspam library.* **bool_t lib_load_dspam** (void)
- *Initialize the dspam library and bind dynamically to the exported functions that are required.* **bool_t dspam_start** (void)
- **void dspam_stop** (void)
- **int_t dspam_check** (uint64_t usernum, **stringer_t** *message, **stringer_t** **signature)
- *dspam.c* **bool_t dspam_train** (uint64_t usernum, **int_t** disposition, **stringer_t** *signature)

Variables

- **pool_t * sql_pool**

Detailed Description

DSPAM interface functions.

Definition in file **dspam.c**.

Function Documentation

int_t dspam_check (uint64_t *usernum*, **stringer_t** * *message*, **stringer_t** ** *signature*)

dspam.c

HIGH: Return a result structure with the disposition, confidence, probability and signature data. Then store the analysis result with the message. Either in the DB or by adding a custom header to the message. Return codes should become 0 for success, or a negative integer for errors. Add statistical updates to check and train functions. Result: result=\"%s\"; class=\"%s\"; probability=01.4f; confidence=02.2f; signature=lu; key=lu; Layman's terms: How spammy is this message

Definition at line 59 of file dspam.c.

References `dspam_attach_d`, `dspam_create_d`, `dspam_destroy_d`, `dspam_detach_d`, `dspam_process_d`, `log_error`, `log_info`, `log_pedantic`, `MAGMA_SPOOL_DATA`, `PL_RESERVED`, `pool_get_obj()`, `pool_pull()`, `pool_release()`, `spool_path()`, `sql_ping()`, `st_char_get()`, `st_free()`, `st_import()`, and `stmt_rebuild()`.

Referenced by `smtp_accept_message()`.

bool_t dspam_start (void)

Definition at line 43 of file dspam.c.

References `dspam_init_driver_d`.

Referenced by `process_start()`.

void dspam_stop (void)

Definition at line 50 of file dspam.c.

References dspam_shutdown_driver_d.

Referenced by process_stop().

bool_t dspam_train (uint64_t usernum, int_t disposition, stringer_t * signature)

Note:

The disposition parameter marks whether or not the signature is currently marked as junk. So training the signature means that it will toggle its status.

Parameters:

usenum the numerical id of the user making the spam training request.

disposition if 0, dspam will mark the signature as junk; otherwise, mark as OK.

signature a managed string containing the spam signature to be trained.

Returns:

true on success or false on failure.

Definition at line 178 of file dspam.c.

References dspam_attach_d, dspam_create_d, dspam_destroy_d, dspam_detach_d, dspam_process_d, log_info, log_pedantic, MAGMA_SPOOL_DATA, PL_RESERVED, pool_get_obj(), pool_pull(), pool_release(), spool_path(), sql_ping(), st_char_get(), st_free(), st_length_get(), and stmt_rebuild().

Referenced by teacher_process().

bool_t lib_load_dspam (void)

Initialize the dspam library and bind dynamically to the exported functions that are required.

Returns:

true on success or false on failure.

Definition at line 29 of file dspam.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

chr_t* lib_version_dspam (void)

Return the version string of the dspam library.

Returns:

a pointer to a character string containing the dspam library version information.

Definition at line 21 of file dspam.c.

References dspam_version_d.

Referenced by lib_load().

Variable Documentation

pool_t* sql_pool

Definition at line 149 of file mysql.c.

Referenced by mail_db_update_message_folder(), sql_query(), sql_query_conn(), stmt_exec(), stmt_exec_affected(), stmt_get_result(), stmt_insert(), stmt_rebuild(), stmt_start(), tran_commit(), tran_rollback(), and tran_start().

magma/providers/checkers/spf.c File Reference

The functions used to validate SPF information.

```
#include "magma.h"
```

Functions

- `const chr_t * lib_version_spf` (void)
- *Return the version string of the spf library.* `bool_t lib_load_spf` (void)
- *Initialize the spf library and bind dynamically to the exported functions that are required.* `bool_t spf_start` (void)
- *Initialize the pool of spf server connections.* `void spf_stop` (void)
- *Destroy the spf connection pool.* `int_t spf_check` (`ip_t *ip`, `stringer_t *helo`, `stringer_t *mailfrom`)

Validate an smtp request via spf. Variables

- `pool_t * spf_pool` = NULL
- `chr_t spf_version` [8]

Detailed Description

The functions used to validate SPF information.

Definition in file `spf.c`.

Function Documentation

`bool_t lib_load_spf` (void)

Initialize the spf library and bind dynamically to the exported functions that are required.

spf.c

Returns:

false on failure or true on success.

Definition at line 32 of file `spf.c`.

References `lib_symbols()`, `M_BIND`, `SPF_get_lib_version_d`, and `spf_version`.

Referenced by `lib_load()`.

`const chr_t* lib_version_spf` (void)

Return the version string of the spf library.

Returns:

a pointer to a character string containing the spf library version information.

Definition at line 24 of file `spf.c`.

References `spf_version`.

Referenced by `lib_load()`.

int_t spf_check (ip_t * *ip*, stringer_t * *helo*, stringer_t * *mailfrom*)

Validate an smtp request via spf.

Parameters:

ip an ip address object containing the ip address of the connection to be checked.

helo a managed string containing the client supplied HELO request value.

mailfrom a managed string containing the client supplied MAIL FROM request value. TODO: We really need to change these return values to account for 0

Returns:

-2 on spf check fail, -1 on other failure, and 1 on spf pass.

Definition at line 117 of file `spf.c`.

References `ip_t::family`, `ip_t::ip4`, `ip_t::ip6`, `log_pedantic`, `mail_domain_get()`, `PL_RESERVED`, `placer_t`, `pool_get_obj()`, `pool_pull()`, `pool_release()`, `SPF_request_free_d`, `SPF_request_new_d`, `SPF_request_query_mailfrom_d`, `SPF_request_set_env_from_d`, `SPF_request_set_helo_dom_d`, `SPF_request_set_ipv4_d`, `SPF_request_set_ipv6_d`, `SPF_response_free_d`, `SPF_response_reason_d`, `SPF_response_result_d`, `SPF_strerror_d`, `SPF_strerror_d`, `SPF_strerror_d`, `SPF_strerror_d`, `st_char_get()`, `st_length_int()`, and `stats_adjust_by_name()`.

Referenced by `smtp_rcpt_to()`.

bool_t spf_start (void)

Initialize the pool of spf server connections.

Returns:

false on failure or true on success.

Definition at line 57 of file `spf.c`.

References `magma_t::iface`, `log_pedantic`, `magma`, `pool_alloc()`, `pool_set_obj()`, `magma_t::spf`, and `SPF_server_new_d`.

Referenced by `process_start()`.

void spf_stop (void)

Destroy the spf connection pool.

Returns:

This function returns no value.

Definition at line 89 of file `spf.c`.

References `magma_t::iface`, `magma`, `pool_free()`, `pool_get_obj()`, `magma_t::spf`, and `SPF_server_free_d`.

Referenced by `process_stop()`.

Variable Documentation

pool_t* spf_pool = NULL

Definition at line 17 of file spf.c.

chr_t spf_version[8]

Definition at line 18 of file spf.c.

Referenced by lib_load_spf(), and lib_version_spf().

magma/providers/compress/bzip.c File Reference

The interface for the BZIP compression functions.

```
#include "magma.h"
```

Functions

- `const char * lib_version_bzip` (void)
- *Return the version string of the bzip library.* `bool_t lib_load_bzip` (void)
- *Initialize the bzip library and bind dynamically to the exported functions that are required.* `stringer_t * decompress_bzip` (`compress_t *compressed`)
- *Decompress data using the bzip engine.* `compress_t * compress_bzip` (`stringer_t *input`)

Compress data using the bzip engine. Variables

- `char bzip_version` [16]

Detailed Description

The interface for the BZIP compression functions.

Definition in file `bzip.c`.

Function Documentation

`compress_t* compress_bzip (stringer_t * input)`

Compress data using the bzip engine.

Parameters:

input a managed string containing the data to be compressed.

Returns:

NULL on failure, or a pointer to the head of the compressed data on success.

Definition at line 105 of file `bzip.c`.

References `BZ2_bzBuffToBuffCompress_d`, `compress_alloc()`, `compress_body_data()`, `COMPRESS_ENGINE_BZIP`, `compress_free()`, `decompress_bzip()`, `hash_adler32()`, `log_info`, `st_data_get()`, `st_free()`, and `st_length_get()`.

Referenced by `engine_compress()`, and `tank_store()`.

`stringer_t* decompress_bzip (compress_t * compressed)`

Decompress data using the bzip engine.

Parameters:

compressed a pointer to the head of the compressed data.

Returns:

NULL on failure, or a managed string containing the uncompressed data on success.
Definition at line 66 of file bzip.c.

References BZ2_bzBuffToBuffDecompress_d, compress_body_data(), COMPRESS_ENGINE_BZIP, hash_adler32(), log_info, st_alloc, st_data_get(), st_free(), and st_length_set().

Referenced by compress_bzip(), engine_decompress(), and tank_load().

bool_t lib_load_bzip (void)

Initialize the bzip library and bind dynamically to the exported functions that are required.

bzip.c**Returns:**

true on success or false on failure.

Definition at line 29 of file bzip.c.

References BZ2_bzlibVersion_d, bzip_version, lib_symbols(), log_pedantic, M_BIND, and ns_length_int().

Referenced by lib_load().

const char* lib_version_bzip (void)

Return the version string of the bzip library.

Returns:

a pointer to a character string containing the bzip library version information.

Definition at line 21 of file bzip.c.

References bzip_version.

Referenced by lib_load().

Variable Documentation**char bzip_version[16]**

Definition at line 15 of file bzip.c.

Referenced by lib_load_bzip(), and lib_version_bzip().

magma/providers/compress/compress.c File Reference

Interface to the compression functions.

```
#include "magma.h"
```

Functions

- `size_t compress_block_length` (void)
- *Get the compression engine's (LZO1X-1) block size.* `uint64_t compress_total_length` (`compress_t *buffer`)
- *Return the total length of a compressed header and body.* `uint64_t compress_orig_hash` (`compress_t *buffer`)
- *Get the hash value of a compressed buffer's original data.* `uint64_t compress_orig_length` (`compress_t *buffer`)
- *Get the original length of a compressed buffer's data.* `uint64_t compress_body_hash` (`compress_t *buffer`)
- *Get the hash value of a compressed buffer's (compressed) body.* `uint64_t compress_body_length` (`compress_t *buffer`)
- *Return the length of a compressed buffer's body.* `void * compress_body_data` (`compress_t *buffer`)
- *Return the compressed body of data associated with a compressed header.* `size_t compress_body_offset` (void)
- *Get the offset to the compressed body from the compressed header.* `compress_t * compress_import` (`stringer_t *s`)
- *Parse and validate a managed string with compressed data and return a compressed header.* `compress_t * compress_alloc` (`size_t length`)
- *Allocate a new compressed object.* `void compress_free` (`compress_t *buffer`)

Free a compressed object.

Detailed Description

Interface to the compression functions.

Definition in file `compress.c`.

Function Documentation

`compress_t* compress_alloc (size_t length)`

Allocate a new compressed object.

`compress.c`

Parameters:

length the size, in bytes, of the data after compression.

Returns:

a pointer to the head of the newly allocated compressed object.

Definition at line 145 of file `compress.c`.

References `mm_alloc()`.

Referenced by `compress_bzip()`, `compress_lzo()`, and `compress_zlib()`.

`size_t compress_block_length (void)`

Get the compression engine's (LZO1X-1) block size.

Note:

In the future this might get stored in the compression header and/or change depending on the engine being used.

Returns:

the size, in bytes, of the compression engine's block size.

Definition at line 20 of file compress.c.

Referenced by mail_load_header().

void* compress_body_data (compress_t * *buffer*)

Return the compressed body of data associated with a compressed header.

Parameters:

buffer the input compressed header.

Returns:

a pointer to the body (start) of the compressed data.

Definition at line 86 of file compress.c.

Referenced by compress_bzip(), compress_import(), compress_lzo(), compress_zlib(), decompress_bzip(), decompress_lzo(), and decompress_zlib().

uint64_t compress_body_hash (compress_t * *buffer*)

Get the hash value of a compressed buffer's (compressed) body.

Parameters:

buffer a pointer to the header of the compressed data.

Returns:

the hash value of the (compressed) of the compressed data.

Definition at line 63 of file compress.c.

uint64_t compress_body_length (compress_t * *buffer*)

Return the length of a compressed buffer's body.

Returns:

the total length in bytes of the compressed data body, excluding the compressed header.

Definition at line 74 of file compress.c.

Referenced by compress_total_length(), and tank_store().

size_t compress_body_offset (void)

Get the offset to the compressed body from the compressed header.

Parameters:

the size, in bytes, of the offset from the start of the compressed data body from the start of the compressed header.

Definition at line 95 of file compress.c.

void compress_free (compress_t * *buffer*)

Free a compressed object.

Parameters:

buffer a pointer to the head of the compressed object to be freed.

Returns:

This function returns no value.

Definition at line 155 of file compress.c.

References mm_free().

Referenced by compress_bzip(), compress_lzo(), compress_zlib(), mail_store_message(), and tank_store().

compress_t* compress_import (stringer_t * *s*)

Parse and validate a managed string with compressed data and return a compressed header.

Parameters:

s the managed string containing the compressed data.

Returns:

NULL on general failure or if the Adler 32 hash doesn't match, or a pointer to the data's compressed header on success.

Definition at line 105 of file compress.c.

References compress_body_data(), hash_adler32(), log_pedantic, st_data_get(), st_empty(), and st_length_get().

Referenced by mail_load_message().

uint64_t compress_orig_hash (compress_t * *buffer*)

Get the hash value of a compressed buffer's original data.

Parameters:

buffer a pointer to the head of the compressed data.

Returns:

the hash value of the original (uncompressed) data.

Definition at line 39 of file compress.c.

uint64_t compress_orig_length (compress_t * *buffer*)

Get the original length of a compressed buffer's data.

Parameters:

buffer a pointer to the head of the compressed data.

Returns:

the original length of the (uncompressed) data.

Definition at line 51 of file compress.c.

uint64_t compress_total_length (compress_t * *buffer*)

Return the total length of a compressed header and body.

Returns:

the total length, in bytes, of the compressed header and body.

Definition at line 29 of file compress.c.

References compress_body_length().

Referenced by mail_store_message(), and tank_store().

magma/providers/compress/compress.h File Reference

Compression interface functions/handlers.

Data Structures

- struct `__attribute__`

Typedefs

- typedef `stringer_t` `compress_t`

Enumerations

- enum { `COMPRESS_ENGINE_LZO` = 1, `COMPRESS_ENGINE_ZLIB` = 2, `COMPRESS_ENGINE_BZIP` = 4 }

Functions

- `bool_t lib_load_bzip` (void)
- *bzip.c* const char * `lib_version_bzip` (void)
- Return the version string of the bzip library. `compress_t * compress_bzip` (`stringer_t *input`)
- Compress data using the bzip engine. `stringer_t * decompress_bzip` (`compress_t *compressed`)
- Decompress data using the bzip engine. `compress_t * compress_alloc` (size_t length)
- *compress.c* size_t `compress_block_length` (void)
- Get the compression engine's (LZO1X-1) block size. void * `compress_body_data` (`compress_t *buffer`)
- Return the compressed body of data associated with a compressed header. uint64_t `compress_body_hash` (`compress_t *buffer`)
- Get the hash value of a compressed buffer's (compressed) body. uint64_t `compress_body_length` (`compress_t *buffer`)
- Return the length of a compressed buffer's body. size_t `compress_body_offset` (void)
- Get the offset to the compressed body from the compressed header. void `compress_free` (`compress_t *buffer`)
- Free a compressed object. `compress_t * compress_import` (`stringer_t *s`)
- Parse and validate a managed string with compressed data and return a compressed header. uint64_t `compress_orig_hash` (`compress_t *buffer`)
- Get the hash value of a compressed buffer's original data. uint64_t `compress_orig_length` (`compress_t *buffer`)
- Get the original length of a compressed buffer's data. uint64_t `compress_total_length` (`compress_t *buffer`)
- Return the total length of a compressed header and body. `compress_t * engine_compress` (uint8_t engine, `stringer_t *s`)
- *engine.c* `stringer_t * engine_decompress` (`compress_t *buffer`)
- `bool_t lib_load_lzo` (void)
- *lzo.c* `compress_t * compress_lzo` (`stringer_t *input`)
- Compress data using the lzo engine. const char * `lib_version_lzo` (void)
- Return the version string of the lzo library. `stringer_t * decompress_block_lzo` (`stringer_t *block`)
- Decompress a single block of data using the lzo engine. `stringer_t * decompress_lzo` (`compress_t *compressed`)
- Decompress data using the lzo engine. `bool_t lib_load_zlib` (void)
- *zlib.c* const char * `lib_version_zlib` (void)
- Return the version string of zlib. `compress_t * compress_zlib` (`stringer_t *input`)
- Compress a block of data using the zlib engine. `stringer_t * decompress_zlib` (`compress_t *compressed`)

Decompress a block of data using the zlib engine. Variables

- enum { ... } `COMPRESS_ENGINE`

Detailed Description

Compression interface functions/handlers.

Definition in file **compress.h**.

Typedef Documentation

typedef stringer_t compress_t

Definition at line 38 of file compress.h.

Enumeration Type Documentation

anonymous enum

Enumerator:

COMPRESS_ENGINE_LZO
COMPRESS_ENGINE_ZLIB
COMPRESS_ENGINE_BZIP

Definition at line 16 of file compress.h.

Function Documentation

compress_t* compress_alloc (size_t *length*)

compress.c

compress.c

Parameters:

length the size, in bytes, of the data after compression.

Returns:

a pointer to the head of the newly allocated compressed object.

Definition at line 145 of file compress.c.

References mm_alloc().

Referenced by compress_bzip(), compress_lzo(), and compress_zlib().

size_t compress_block_length (void)

Get the compression engine's (LZO1X-1) block size.

Note:

In the future this might get stored in the compression header and/or change depending on the engine being used.

Returns:

the size, in bytes, of the compression engine's block size.

Definition at line 20 of file compress.c.

Referenced by mail_load_header().

void* compress_body_data (compress_t * *buffer*)

Return the compressed body of data associated with a compressed header.

Parameters:

buffer the input compressed header.

Returns:

a pointer to the body (start) of the compressed data.

Definition at line 86 of file compress.c.

Referenced by compress_bzip(), compress_import(), compress_lzo(), compress_zlib(), decompress_bzip(), decompress_lzo(), and decompress_zlib().

uint64_t compress_body_hash (compress_t * *buffer*)

Get the hash value of a compressed buffer's (compressed) body.

Parameters:

buffer a pointer to the header of the compressed data.

Returns:

the hash value of the (compressed) of the compressed data.

Definition at line 63 of file compress.c.

uint64_t compress_body_length (compress_t * *buffer*)

Return the length of a compressed buffer's body.

Returns:

the total length in bytes of the compressed data body, excluding the compressed header.

Definition at line 74 of file compress.c.

Referenced by compress_total_length(), and tank_store().

size_t compress_body_offset (void)

Get the offset to the compressed body from the compressed header.

Parameters:

the size, in bytes, of the offset from the start of the compressed data body from the start of the compressed header.

Definition at line 95 of file compress.c.

compress_t* compress_bzip (stringer_t * *input*)

Compress data using the bzip engine.

Parameters:

input a managed string containing the data to be compressed.

Returns:

NULL on failure, or a pointer to the head of the compressed data on success.

Definition at line 105 of file bzip.c.

References BZ2_bzBuffToBuffCompress_d, compress_alloc(), compress_body_data(), COMPRESS_ENGINE_BZIP, compress_free(), decompress_bzip(), hash_adler32(), log_info, st_data_get(), st_free(), and st_length_get().

Referenced by engine_compress(), and tank_store().

void compress_free (compress_t * *buffer*)

Free a compressed object.

Parameters:

buffer a pointer to the head of the compressed object to be freed.

Returns:

This function returns no value.

Definition at line 155 of file compress.c.

References mm_free().

Referenced by compress_bzip(), compress_lzo(), compress_zlib(), mail_store_message(), and tank_store().

compress_t* compress_import (stringer_t * *s*)

Parse and validate a managed string with compressed data and return a compressed header.

Parameters:

s the managed string containing the compressed data.

Returns:

NULL on general failure or if the Adler 32 hash doesn't match, or a pointer to the data's compressed header on success.

Definition at line 105 of file compress.c.

References compress_body_data(), hash_adler32(), log_pedantic, st_data_get(), st_empty(), and st_length_get().

Referenced by mail_load_message().

compress_t* compress_lzo (stringer_t * *input*)

Compress data using the lzo engine.

Parameters:

input a managed string containing the data to be compressed.

Returns:

NULL on failure, or a pointer to the head of the compressed data on success.

Definition at line 137 of file lzo.c.

References compress_alloc(), compress_body_data(), COMPRESS_ENGINE_LZO, compress_free(), decompress_lzo(), hash_adler32(), log_info, lzo1x_1_compress_d, mm_alloc(), mm_free(), st_data_get(), st_free(), and st_length_get().

Referenced by engine_compress(), mail_store_message(), and tank_store().

uint64_t compress_orig_hash (compress_t * *buffer*)

Get the hash value of a compressed buffer's original data.

Parameters:

buffer a pointer to the head of the compressed data.

Returns:

the hash value of the original (uncompressed) data.

Definition at line 39 of file compress.c.

uint64_t compress_orig_length (compress_t * *buffer*)

Get the original length of a compressed buffer's data.

Parameters:

buffer a pointer to the head of the compressed data.

Returns:

the original length of the (uncompressed) data.

Definition at line 51 of file compress.c.

uint64_t compress_total_length (compress_t * *buffer*)

Return the total length of a compressed header and body.

Returns:

the total length, in bytes, of the compressed header and body.

Definition at line 29 of file compress.c.

References `compress_body_length()`.

Referenced by `mail_store_message()`, and `tank_store()`.

`compress_t* compress_zlib (stringer_t * input)`

Compress a block of data using the zlib engine.

Parameters:

input a managed string containing the data to be compressed.

Returns:

NULL on failure, or a pointer to the compressed data header on success..

Definition at line 89 of file `zlib.c`.

References `compress2_d`, `compress_alloc()`, `compress_body_data()`, `COMPRESS_ENGINE_ZLIB`, `compress_free()`, `compressBound_d`, `decompress_zlib()`, `hash_adler32()`, `log_info`, `st_data_get()`, `st_free()`, and `st_length_get()`.

Referenced by `engine_compress()`, and `tank_store()`.

`stringer_t* decompress_block_lzo (stringer_t * block)`

Decompress a single block of data using the lzo engine.

Parameters:

block a managed string containing the block of compressed data.

Returns:

NULL on failure, or a managed string containing the uncompressed data on success.

Definition at line 51 of file `lzo.c`.

References `log_info`, `lzo1x_decompress_safe_d`, `st_alloc`, `st_data_get()`, `st_free()`, `st_length_get()`, and `st_length_set()`.

Referenced by `mail_load_header()`.

`stringer_t* decompress_bzip (compress_t * compressed)`

Decompress data using the bzip engine.

Parameters:

compressed a pointer to the head of the compressed data.

Returns:

NULL on failure, or a managed string containing the uncompressed data on success.

Definition at line 66 of file `bzip.c`.

References `BZ2_bzBuffToBuffDecompress_d`, `compress_body_data()`, `COMPRESS_ENGINE_BZIP`, `hash_adler32()`, `log_info`, `st_alloc`, `st_data_get()`, `st_free()`, and `st_length_set()`.

Referenced by `compress_bzip()`, `engine_decompress()`, and `tank_load()`.

stringer_t* decompress_lzo (compress_t * *compressed*)

Decompress data using the lzo engine.

Parameters:

compressed a pointer to the head of the compressed data.

Returns:

NULL on failure, or a managed string containing the uncompressed data on success.

Definition at line 91 of file lzo.c.

References compress_body_data(), COMPRESS_ENGINE_LZO, hash_adler32(), log_info, lzo1x_decompress_safe_d, st_alloc, st_data_get(), st_free(), and st_length_set().

Referenced by compress_lzo(), engine_decompress(), mail_load_header(), mail_load_message(), and tank_load().

stringer_t* decompress_zlib (compress_t * *compressed*)

Decompress a block of data using the zlib engine.

Parameters:

compressed a pointer to the head of the compressed data.

Returns:

NULL on failure (e.g. corruption), or a managed string containing the uncompressed data on success.

Definition at line 50 of file zlib.c.

References compress_body_data(), COMPRESS_ENGINE_ZLIB, hash_adler32(), log_info, st_alloc, st_data_get(), st_free(), st_length_set(), and uncompress_d.

Referenced by compress_zlib(), engine_decompress(), and tank_load().

compress_t* engine_compress (uint8_t *engine*, stringer_t * *s*)

engine.c

Definition at line 15 of file engine.c.

References compress_bzip(), COMPRESS_ENGINE_BZIP, COMPRESS_ENGINE_LZO, COMPRESS_ENGINE_ZLIB, compress_lzo(), compress_zlib(), and log_pedantic.

stringer_t* engine_decompress (compress_t * *buffer*)

Definition at line 40 of file engine.c.

References COMPRESS_ENGINE_BZIP, COMPRESS_ENGINE_LZO, COMPRESS_ENGINE_ZLIB, decompress_bzip(), decompress_lzo(), decompress_zlib(), and log_pedantic.

bool_t lib_load_bzip (void)

bzip.c

bzip.c

Returns:

true on success or false on failure.

Definition at line 29 of file bzip.c.

References BZ2_bzlibVersion_d, bzip_version, lib_symbols(), log_pedantic, M_BIND, and ns_length_int().

Referenced by lib_load().

bool_t lib_load_lzo (void)

lzo.c

lzo.c

Returns:

true on success or false on failure.

Definition at line 20 of file lzo.c.

References __lzo_init_v2_d, lib_symbols(), log_critical, and M_BIND.

Referenced by lib_load().

bool_t lib_load_zlib (void)

zlib.c

zlib.c

Returns:

true on success or false on failure.

Definition at line 32 of file zlib.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

const char* lib_version_bzip (void)

Return the version string of the bzip library.

Returns:

a pointer to a character string containing the bzip library version information.

Definition at line 21 of file bzip.c.

References bzip_version.

Referenced by lib_load().

const char* lib_version_lzo (void)

Return the version string of the lzo library.

Returns:

a pointer to a character string containing the lzo library version information.

Definition at line 42 of file lzo.c.

References lzo_version_string_d.

Referenced by lib_load().

const char* lib_version_zlib (void)

Return the version string of zlib.

Returns:

a pointer to a character string containing the zlib version information.

Definition at line 24 of file zlib.c.

References zlibVersion_d.

Referenced by lib_load().

Variable Documentation

enum { ... } COMPRESS_ENGINE

magma/providers/compress/engine.c File Reference

Generic engine independent compression interfaces.

```
#include "magma.h"
```

Functions

- **compress_t * engine_compress** (uint8_t engine, **stringer_t** *s)
 - *engine.c* **stringer_t * engine_decompress** (compress_t *buffer)
-

Detailed Description

Generic engine independent compression interfaces.

Definition in file **engine.c**.

Function Documentation

compress_t* engine_compress (uint8_t *engine*, **stringer_t** * s)

engine.c

Definition at line 15 of file engine.c.

References `compress_bzip()`, `COMPRESS_ENGINE_BZIP`, `COMPRESS_ENGINE_LZO`, `COMPRESS_ENGINE_ZLIB`, `compress_lzo()`, `compress_zlib()`, and `log_pedantic`.

stringer_t* engine_decompress (compress_t * *buffer*)

Definition at line 40 of file engine.c.

References `COMPRESS_ENGINE_BZIP`, `COMPRESS_ENGINE_LZO`, `COMPRESS_ENGINE_ZLIB`, `decompress_bzip()`, `decompress_lzo()`, `decompress_zlib()`, and `log_pedantic`.

magma/providers/compress/lzo.c File Reference

The interface for the LZO compression functions.

```
#include "magma.h"
```

Functions

- **bool_t lib_load_lzo** (void)
 - *Initialize the lzo library and bind dynamically to the exported functions that are required.* **const char * lib_version_lzo** (void)
 - *Return the version string of the lzo library.* **stringer_t * decompress_block_lzo** (**stringer_t *block**)
 - *Decompress a single block of data using the lzo engine.* **stringer_t * decompress_lzo** (**compress_t *compressed**)
 - *Decompress data using the lzo engine.* **compress_t * compress_lzo** (**stringer_t *input**)
- Compress data using the lzo engine.*
-

Detailed Description

The interface for the LZO compression functions.

Definition in file **lzo.c**.

Function Documentation

compress_t* compress_lzo (stringer_t * *input*)

Compress data using the lzo engine.

Parameters:

input a managed string containing the data to be compressed.

Returns:

NULL on failure, or a pointer to the head of the compressed data on success.

Definition at line 137 of file lzo.c.

References `compress_alloc()`, `compress_body_data()`, `COMPRESS_ENGINE_LZO`, `compress_free()`, `decompress_lzo()`, `hash_adler32()`, `log_info`, `lzo1x_1_compress_d`, `mm_alloc()`, `mm_free()`, `st_data_get()`, `st_free()`, and `st_length_get()`.

Referenced by `engine_compress()`, `mail_store_message()`, and `tank_store()`.

stringer_t* decompress_block_lzo (stringer_t * *block*)

Decompress a single block of data using the lzo engine.

Parameters:

block a managed string containing the block of compressed data.

Returns:

NULL on failure, or a managed string containing the uncompressed data on success.

Definition at line 51 of file lzo.c.

References `log_info`, `lzo1x_decompress_safe_d`, `st_alloc`, `st_data_get()`, `st_free()`, `st_length_get()`, and `st_length_set()`.

Referenced by `mail_load_header()`.

stringer_t* decompress_lzo (compress_t * *compressed*)

Decompress data using the lzo engine.

Parameters:

compressed a pointer to the head of the compressed data.

Returns:

NULL on failure, or a managed string containing the uncompressed data on success.

Definition at line 91 of file lzo.c.

References `compress_body_data()`, `COMPRESS_ENGINE_LZO`, `hash_adler32()`, `log_info`, `lzo1x_decompress_safe_d`, `st_alloc`, `st_data_get()`, `st_free()`, and `st_length_set()`.

Referenced by `compress_lzo()`, `engine_decompress()`, `mail_load_header()`, `mail_load_message()`, and `tank_load()`.

bool_t lib_load_lzo (void)

Initialize the lzo library and bind dynamically to the exported functions that are required.

lzo.c**Returns:**

true on success or false on failure.

Definition at line 20 of file lzo.c.

References `__lzo_init_v2_d`, `lib_symbols()`, `log_critical`, and `M_BIND`.

Referenced by `lib_load()`.

const char* lib_version_lzo (void)

Return the version string of the lzo library.

Returns:

a pointer to a character string containing the lzo library version information.

Definition at line 42 of file lzo.c.

References `lzo_version_string_d`.

Referenced by `lib_load()`.

magma/providers/compress/zlib.c File Reference

The interface for the ZLIB compression functions.

```
#include "magma.h"
```

Functions

- `const char * lib_version_zlib (void)`
- *Return the version string of zlib.* `bool_t lib_load_zlib (void)`
- *Initialize the zlib library and bind dynamically to the exported functions that are required.* `stringer_t * decompress_zlib (compress_t *compressed)`
- *Decompress a block of data using the zlib engine.* `compress_t * compress_zlib (stringer_t *input)`

Compress a block of data using the zlib engine. Variables

- `int(* deflateEnd_d)(z_stream strm) = NULL`
- `int(* deflate_d)(z_stream strm, int flush) = NULL`
- `int(* deflateInit2_d)(z_stream strm, int level, int method, int windowBits, int memLevel, int strategy, const char *version, int stream_size) = NULL`

Detailed Description

The interface for the ZLIB compression functions.

Definition in file **zlib.c**.

Function Documentation

compress_t* compress_zlib (stringer_t * input)

Compress a block of data using the zlib engine.

Parameters:

input a managed string containing the data to be compressed.

Returns:

NULL on failure, or a pointer to the compressed data header on success..

Definition at line 89 of file **zlib.c**.

References `compress2_d`, `compress_alloc()`, `compress_body_data()`, `COMPRESS_ENGINE_ZLIB`, `compress_free()`, `compressBound_d`, `decompress_zlib()`, `hash_adler32()`, `log_info`, `st_data_get()`, `st_free()`, and `st_length_get()`.

Referenced by `engine_compress()`, and `tank_store()`.

stringer_t* decompress_zlib (compress_t * compressed)

Decompress a block of data using the zlib engine.

Parameters:

compressed a pointer to the head of the compressed data.

Returns:

NULL on failure (e.g. corruption), or a managed string containing the uncompressed data on success.

Definition at line 50 of file `zlib.c`.

References `compress_body_data()`, `COMPRESS_ENGINE_ZLIB`, `hash_adler32()`, `log_info`, `st_alloc`, `st_data_get()`, `st_free()`, `st_length_set()`, and `uncompress_d`.

Referenced by `compress_zlib()`, `engine_decompress()`, and `tank_load()`.

bool_t lib_load_zlib (void)

Initialize the zlib library and bind dynamically to the exported functions that are required.

zlib.c

Returns:

true on success or false on failure.

Definition at line 32 of file `zlib.c`.

References `lib_symbols()`, and `M_BIND`.

Referenced by `lib_load()`.

const char* lib_version_zlib (void)

Return the version string of zlib.

Returns:

a pointer to a character string containing the zlib version information.

Definition at line 24 of file `zlib.c`.

References `zlibVersion_d`.

Referenced by `lib_load()`.

Variable Documentation

int(* deflate_d)(z_stream strm, int flush) = NULL

int(* deflateEnd_d)(z_stream strm) = NULL

int(* deflateInit2_d)(z_stream strm, int level, int method, int windowBits, int memLevel, int strategy, const char *version, int stream_size) = NULL

magma/providers/consumers/consumers.h File Reference

Interfaces for clients that consume/load/retrieve data across various network protocols.

Data Structures

- struct **serialization_t**

Functions

- **int_t** **cache_add** (**stringer_t** *key, **stringer_t** *object, **time_t** expiration)
- *cache.c* **int_t** **cache_append** (**stringer_t** *key, **stringer_t** *object, **time_t** expiration)
- Append data to the value of a cached key on a memcached server. **int_t** **cache_delete** (**stringer_t** *key)
- Delete a key from memcached immediately. **void** **cache_flush** (**void**)
- **stringer_t** * **cache_get** (**stringer_t** *key)
- Retrieve data from memcached by key. **uint64_t** **cache_get_u64** (**stringer_t** *key)
- Retrieve a 64 bit value from memcached by key. **uint64_t** **cache_increment** (**stringer_t** *key, **uint64_t** offset, **uint64_t** initial, **time_t** expiration)
- Increment the value stored with a key by memcached by a specified offset. **int_t** **cache_set** (**stringer_t** *key, **stringer_t** *object, **time_t** expiration)
- Set a value in memcached by key. **int_t** **cache_set_u64** (**stringer_t** *key, **uint64_t** value, **time_t** expiration)
- Set a 64 bit value in memcached by key. **int_t** **cache_silent_add** (**stringer_t** *key, **stringer_t** *object, **time_t** expiration)
- Add a new key/value pair to the memcached server, but suppress any error messages. **bool_t** **cache_start** (**void**)
- **void** **cache_stop** (**void**)
- Destroy the memcached client pool and all active connections. **bool_t** **lib_load_cache** (**void**)
- Initialize the memcached library and bind dynamically to the exported functions that are required. **const char** * **lib_version_cache** (**void**)
- Return the version string of the memcached library. **bool_t** **serialize_sz** (**stringer_t** **data, **size_t** number)
- Serialization. **bool_t** **serialize_ssz** (**stringer_t** **data, **ssize_t** number)
- Serialize an **ssize_t** into a managed string. **bool_t** **serialize_int16** (**stringer_t** **data, **int16_t** number)
- Serialize a signed 16-bit integer into a managed string. **bool_t** **serialize_int32** (**stringer_t** **data, **int32_t** number)
- Serialize a signed 32-bit integer into a managed string. **bool_t** **serialize_int64** (**stringer_t** **data, **int64_t** number)
- Serialize a signed 64-bit integer into a managed string. **bool_t** **serialize_st** (**stringer_t** **data, **stringer_t** *string)
- Serialize the contents of a managed string into another managed string. **bool_t** **serialize_uint64** (**stringer_t** **data, **int64_t** number)
- Serialize an unsigned 64-bit integer into a managed string. **bool_t** **serialize_uint16** (**stringer_t** **data, **uint16_t** number)
- Serialize an unsigned 16-bit integer into a managed string. **bool_t** **serialize_uint32** (**stringer_t** **data, **uint32_t** number)
- Serialize an unsigned 32-bit integer into a managed string. **bool_t** **serialize_ns** (**stringer_t** **data, **char** *string, **size_t** length)
- Serialize a block of memory into a managed string. **int_t** **deserialize_count_digits** (**chr_t** *data, **size_t** remaining)
- Count the number of bytes before a semicolon or non-numeric character is found. **bool_t** **deserialize_sz** (**serialization_t** *serial, **size_t** *number)
- Deserialize an **size_t** from a serialized object, and update its position. **bool_t** **deserialize_ssz** (**serialization_t** *serial, **ssize_t** *number)
- Deserialize an **ssize_t** from a serialized object, and update its position. **bool_t** **deserialize_int32** (**serialization_t** *serial, **int_t** *number)

- *Deserialize a signed 32-bit integer from a serialized object, and update its position.* **bool_t deserialize_int64** (**serialization_t** *serial, long int ***number**)
 - **bool_t deserialize_st** (**serialization_t** *serial, **stringer_t** **string)
 - *Deserialize a managed string from a serialized object, and update its position.* **bool_t deserialize_int16** (**serialization_t** *serial, short int ***number**)
 - **bool_t deserialize_uint32** (**serialization_t** *serial, uint32_t ***number**)
 - *Deserialize an unsigned 32-bit integer from a serialized object, and update its position.* **bool_t deserialize_uint64** (**serialization_t** *serial, uint64_t ***number**)
 - *Deserialize an unsigned 64-bit integer from a serialized object, and update its position.* **bool_t deserialize_ns** (**serialization_t** *serial, **chr_t** **string, size_t ***length**)
 - *Deserialize a null-terminated string from a serialized object, and update its position.* **bool_t deserialize_uint16** (**serialization_t** *serial, short unsigned int ***number**)
-

Detailed Description

Interfaces for clients that consume/load/retrieve data across various network protocols.

Definition in file **consumers.h**.

Function Documentation

int_t cache_add (**stringer_t** * *key*, **stringer_t** * *object*, **time_t** *expiration*)

cache.c

cache.c

Parameters:

key a managed string with the name of the key to be added to the cache.

object a managed string containing the initial value of the new key.

expiration an expiration time, in seconds, for the newly cached data.

Definition at line 284 of file cache.c.

References `log_info`, `memcached_add_d`, `memcached_strerror_d`, `PL_RESERVED`, `pool_get_obj()`, `pool_pull()`, `pool_release()`, `st_char_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by `mail_load_header()`, and `mail_load_message()`.

int_t cache_append (**stringer_t** * *key*, **stringer_t** * *object*, **time_t** *expiration*)

Append data to the value of a cached key on a memcached server.

Parameters:

key a managed string with the name of the target memcached key.

object a managed string containing the value of the data being appended to the original key.

expiration an expiration time, in seconds, for the modified cached data.

Definition at line 329 of file cache.c.

References `log_info`, `memcached_add_d`, `memcached_append_d`, `memcached_strerror_d`, `PL_RESERVED`, `pool_get_obj()`, `pool_pull()`, `pool_release()`, `st_char_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by stamp_counter_increment().

int_t cache_delete (stringer_t * key)

Delete a key from memcached immediately.

Note:

memcached_delete()

Parameters:

key the name of the key to be deleted from the memcached server.

Returns:

0 on failure, or MEMCACHED_SUCCESS on success.

Definition at line 359 of file cache.c.

References log_info, memcached_delete_d, memcached_strerror_d, PL_RESERVED, pool_get_obj(), pool_pull(), pool_release(), st_char_get(), st_length_get(), and st_length_int().

Referenced by lock_release().

void cache_flush (void)

Definition at line 130 of file cache.c.

References log_info, memcached_flush_d, memcached_strerror_d, PL_RESERVED, pool_get_obj(), pool_pull(), and pool_release().

Referenced by main().

stringer_t* cache_get (stringer_t * key)

Retrieve data from memcached by key.

Parameters:

key a managed string containing a key to be passed to memcached.

Returns:

NULL on failure, or a pointer to a managed string containing the cached data on success.

Definition at line 151 of file cache.c.

References data, length, log_info, memcached_get_d, memcached_strerror_d, mm_free(), PL_RESERVED, pool_get_obj(), pool_pull(), pool_release(), st_char_get(), st_import(), st_length_get(), and st_length_int().

Referenced by mail_load_header(), mail_load_message(), register_session_get(), smtp_check_greylist(), smtp_fetch_autoreply(), stamp_counter_check(), and teacher_data_get().

uint64_t cache_get_u64 (stringer_t * key)

Retrieve a 64 bit value from memcached by key.

Parameters:

key a managed string containing a key to be passed to memcached.

Returns:

NULL on failure, or the 64 bit value cached data on success.

Definition at line 191 of file cache.c.

References data, length, log_info, memcached_get_d, memcached_strerror_d, mm_free(), PL_RESERVED, pool_get_obj(), pool_pull(), pool_release(), st_char_get(), st_length_get(), and st_length_int().

Referenced by contact_abuse_checks(), register_abuse_checks(), and smtp_reply().

uint64_t cache_increment (stringer_t * key, uint64_t offset, uint64_t initial, time_t expiration)

Increment the value stored with a key by memcached by a specified offset.

See also:

memcached_increment_with_initial()

Parameters:

key a managed string containing the name of the memcached key to be adjusted.

offset the offset by which the specified key's value should be incremented.

initial the initial value to seed the key if it does not already exist.

expiration the time, in seconds, when the cached key will expire.

Returns:

0 on failure, or the new incremented value of the data associated with the key, on success.

Definition at line 385 of file cache.c.

References log_pedantic, memcached_increment_with_initial_d, memcached_strerror_d, PL_RESERVED, pool_get_obj(), pool_pull(), pool_release(), st_char_get(), st_length_get(), and st_length_int().

Referenced by contact_abuse_increment_history(), register_abuse_increment_history(), serial_get(), and serial_increment().

int_t cache_set (stringer_t * key, stringer_t * object, time_t expiration)

Set a value in memcached by key.

Parameters:

key a managed string containing a key to be passed to memcached.

object a managed string containing the new data to be associated with the supplied key.

expiration the expiration time of the cached data.

Returns:

0 on failure, or 1 on success.

Definition at line 237 of file cache.c.

References log_info, memcached_set_d, memcached_strerror_d, PL_RESERVED, pool_get_obj(), pool_pull(), pool_release(), st_char_get(), st_length_get(), and st_length_int().

Referenced by register_session_cache(), serial_reset(), smtp_check_greylist(), smtp_fetch_autoreply(), and teacher_data_save().

int_t cache_set_u64 (stringer_t * key, uint64_t value, time_t expiration)

Set a 64 bit value in memcached by key.

Parameters:

key a managed string containing a key to be passed to memcached.

value the new value to be associated with the supplied key.

expiration the expiration time of the cached data.

Returns:

NULL on failure, or the retrieved unsigned 64 bit cached data on success.

Definition at line 261 of file cache.c.

References log_info, memcached_set_d, memcached_strerror_d, PL_RESERVED, pool_get_obj(), pool_pull(), pool_release(), st_char_get(), st_length_get(), and st_length_int().

Referenced by smtp_reply().

int_t cache_silent_add (stringer_t * key, stringer_t * object, time_t expiration)

Add a new key/value pair to the memcached server, but suppress any error messages.

Parameters:

key a managed string with the name of the key to be added to the cache.

object a managed string containing the initial value of the new key.

expiration an expiration time, in seconds, for the newly cached data.

Definition at line 307 of file cache.c.

References memcached_add_d, PL_RESERVED, pool_get_obj(), pool_pull(), pool_release(), st_char_get(), and st_length_get().

Referenced by lock_get().

bool_t cache_start (void)

Definition at line 51 of file cache.c.

References magma_t::cache, magma_t::iface, log_critical, magma, MAGMA_CACHE_INSTANCES, memcached_behavior_set_d, memcached_create_d, memcached_free_d, memcached_server_add_with_weight_d, memcached_strerror_d, pool_alloc(), and pool_set_obj().

Referenced by process_start().

void cache_stop (void)

Destroy the memcached client pool and all active connections.

Returns:

This function returns no value.

Definition at line 110 of file cache.c.

References magma_t::cache, magma_t::iface, magma, memcached_free_d, pool_free(), and pool_get_obj().

Referenced by process_stop().

int_t deserialize_count_digits (chr_t * *data*, size_t *remaining*) [inline]

Count the number of bytes before a semicolon or non-numeric character is found.

Parameters:

data a pointer to a null-terminated string to be scanned.
remaining the maximum number of characters to be scanned.

Returns:

-1 if no match is found, or the length, in bytes, of the data preceding the matched character.

Definition at line 21 of file deserialization.c.

Referenced by `deserialize_int16()`, `deserialize_int32()`, `deserialize_int64()`, `deserialize_uint16()`, `deserialize_uint32()`, and `deserialize_uint64()`.

bool_t deserialize_int16 (serialization_t * *serial*, short int * *number*)

bool_t deserialize_int32 (serialization_t * *serial*, int_t * *number*)

Deserialize a signed 32-bit integer from a serialized object, and update its position.

Parameters:

serial a pointer to a serialized object from which the data will be extracted.
number a pointer to a signed 32-bit integer which will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure.

Definition at line 143 of file deserialization.c.

References `serialization_t::data`, `deserialize_count_digits()`, `int32_conv_bl()`, `length`, `log_pedantic`, `serialization_t::position`, `st_char_get()`, and `st_length_get()`.

Referenced by `teacher_data_get()`.

bool_t deserialize_int64 (serialization_t * *serial*, long int * *number*)

Referenced by `deserialize_ssz()`.

bool_t deserialize_ns (serialization_t * *serial*, chr_t ** *string*, size_t * *length*)

Deserialize a null-terminated string from a serialized object, and update its position.

Note:

A null-terminated string is stored as a 64 bit length field, followed by the string contents.

Parameters:

serial a pointer to a serialized object from which the data will be extracted.
string a pointer to the address of a null-terminated string that will receive the de-serialized data at the serialized object's current position.
length a pointer to a `size_t` that will receive the length of the de-serialized null-terminated string on success.

Returns:

true on success or false on failure..

Definition at line 357 of file deserialization.c.

References bytes, serialization_t::data, deserialize_sz(), log_pedantic, mm_copy(), ns_alloc(), serialization_t::position, st_char_get(), and st_length_get().

bool_t deserialize_ssz (serialization_t * *serial*, ssize_t * *number*)

Deserialize an ssize_t from a serialized object, and update its position.

See also:

deserialize_int64

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to an ssize_t variable which will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure.

Definition at line 53 of file deserialization.c.

References deserialize_int64().

bool_t deserialize_st (serialization_t * *serial*, stringer_t ** *string*)

Deserialize a managed string from a serialized object, and update its position.

Note:

A managed string is stored as a 64 bit length field, followed by opaque data.

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to the address of a managed string, will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure..

Definition at line 304 of file deserialization.c.

References serialization_t::data, deserialize_sz(), length, log_pedantic, mm_copy(), serialization_t::position, st_alloc, st_char_get(), st_length_get(), and st_length_set().

Referenced by register_session_get(), and teacher_data_get().

bool_t deserialize_sz (serialization_t * *serial*, size_t * *number*)

Deserialize an size_t from a serialized object, and update its position.

See also:

deserialize_uint64

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to an `size_t` variable which will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure.

Definition at line 42 of file `deserialization.c`.

References `deserialize_uint64()`.

Referenced by `deserialize_ns()`, and `deserialize_st()`.

`bool_t deserialize_uint16 (serialization_t * serial, short unsigned int * number)`

Referenced by `register_session_get()`.

`bool_t deserialize_uint32 (serialization_t * serial, uint32_t * number)`

Deserialize an unsigned 32-bit integer from a serialized object, and update its position.

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to an unsigned 32-bit integer which will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure.

Definition at line 103 of file `deserialization.c`.

References `serialization_t::data`, `deserialize_count_digits()`, `length`, `log_pedantic`, `serialization_t::position`, `st_char_get()`, `st_length_get()`, and `uint32_conv_bl()`.

`bool_t deserialize_uint64 (serialization_t * serial, uint64_t * number)`

Deserialize an unsigned 64-bit integer from a serialized object, and update its position.

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to an unsigned 64-bit integer which will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure.

Definition at line 63 of file `deserialization.c`.

References `serialization_t::data`, `deserialize_count_digits()`, `length`, `log_pedantic`, `serialization_t::position`, `st_char_get()`, `st_length_get()`, and `uint64_conv_bl()`.

Referenced by `deserialize_sz()`, `register_session_get()`, and `teacher_data_get()`.

`bool_t lib_load_cache (void)`

Initialize the memcached library and bind dynamically to the exported functions that are required.

Returns:

true on success or false on failure.

Definition at line 22 of file cache.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

const char* lib_version_cache (void)

Return the version string of the memcached library.

Returns:

a pointer to a character string containing the memcached library version information.

Definition at line 43 of file cache.c.

References memcached_lib_version_d.

Referenced by lib_load().

bool_t serialize_int16 (stringer_t ** data, int16_t number)

Serialize a signed 16-bit integer into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the signed 16-bit integer to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 44 of file serialization.c.

References int16_digits(), length, st_alloc, st_avail_get(), st_char_get(), st_length_get(), st_length_set(), and st_realloc().

bool_t serialize_int32 (stringer_t ** data, int32_t number)

Serialize a signed 32-bit integer into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the signed 32-bit integer to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 116 of file serialization.c.

References `int32_digits()`, `length`, `st_alloc`, `st_avail_get()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, and `st_realloc()`.

Referenced by `teacher_data_save()`.

bool_t serialize_int64 (stringer_t ** data, int64_t number)

Serialize a signed 64-bit integer into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the signed 64-bit integer to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 187 of file serialization.c.

References `int64_digits()`, `length`, `st_alloc`, `st_avail_get()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, and `st_realloc()`.

Referenced by `serialize_ssz()`.

bool_t serialize_ns (stringer_t ** data, char * string, size_t length)

Serialize a block of memory into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

string a pointer to the start of the memory block to be serialized.

length the length, in bytes, of the memory block to be serialized.

Returns:

1 if the block of memory was serialized and stored successfully, or -1 on failure.

Definition at line 307 of file serialization.c.

References `mm_copy()`, `serialize_ssz()`, `st_avail_get()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, and `st_realloc()`.

bool_t serialize_ssz (stringer_t ** data, ssize_t number)

Serialize an ssize_t into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the ssize_t to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 33 of file serialization.c.

References serialize_int64().

bool_t serialize_st (stringer_t ** data, stringer_t * string)

Serialize the contents of a managed string into another managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

string a pointer to a managed string containing the data to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 259 of file serialization.c.

References length, mm_copy(), serialize_ssz(), st_avail_get(), st_char_get(), st_length_get(), st_length_set(), and st_realloc().

Referenced by register_session_cache(), and teacher_data_save().

bool_t serialize_ssz (stringer_t ** data, size_t number)

Serialization.

Serialization.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the size_t to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 22 of file serialization.c.

References `serialize_uint64()`.

Referenced by `serialize_ns()`, and `serialize_st()`.

bool_t serialize_uint16 (stringer_t ** data, uint16_t number)

Serialize an unsigned 16-bit integer into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the unsigned 16-bit integer to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 80 of file serialization.c.

References `length`, `st_alloc`, `st_avail_get()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, `st_realloc()`, and `uint16_digits()`.

Referenced by `register_session_cache()`.

bool_t serialize_uint32 (stringer_t ** data, uint32_t number)

Serialize an unsigned 32-bit integer into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the unsigned 32-bit integer to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 152 of file serialization.c.

References `length`, `st_alloc`, `st_avail_get()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, `st_realloc()`, and `uint32_digits()`.

bool_t serialize_uint64 (stringer_t ** data, int64_t number)

Serialize an unsigned 64-bit integer into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the unsigned 64-bit integer to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 223 of file serialization.c.

References `length`, `st_alloc`, `st_avail_get()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, `st_realloc()`, and `uint64_digits()`.

Referenced by `register_session_cache()`, `serialize_sz()`, and `teacher_data_save()`.

magma/providers/consumers/deserialization.c File Reference

Distributed cache interface for de-serializing various data types.

```
#include "magma.h"
```

Functions

- **int_t deserialize_count_digits** (**chr_t** ***data**, **size_t** *remaining*)
- *Count the number of bytes before a semicolon or non-numeric character is found.* **bool_t deserialize_sz** (**serialization_t** ***serial**, **size_t** ***number**)
- *Deserialize an size_t from a serialized object, and update its position.* **bool_t deserialize_ssz** (**serialization_t** ***serial**, **ssize_t** ***number**)
- *Deserialize an ssize_t from a serialized object, and update its position.* **bool_t deserialize_uint64** (**serialization_t** ***serial**, **uint64_t** ***number**)
- *Deserialize an unsigned 64-bit integer from a serialized object, and update its position.* **bool_t deserialize_uint32** (**serialization_t** ***serial**, **uint32_t** ***number**)
- *Deserialize an unsigned 32-bit integer from a serialized object, and update its position.* **bool_t deserialize_int32** (**serialization_t** ***serial**, **int_t** ***number**)
- *Deserialize a signed 32-bit integer from a serialized object, and update its position.* **bool_t deserialize_int64** (**serialization_t** ***serial**, **long** ***number**)
- *Deserialize a signed 64-bit integer from a serialized object, and update its position.* **bool_t deserialize_int16** (**serialization_t** ***serial**, **short** ***number**)
- *Deserialize a signed 16 bit integer from a serialized object, and update its position.* **bool_t deserialize_uint16** (**serialization_t** ***serial**, **unsigned short** ***number**)
- *Deserialize an unsigned 16 bit integer from a serialized object, and update its position.* **bool_t deserialize_st** (**serialization_t** ***serial**, **stringer_t** ****string**)
- *Deserialize a managed string from a serialized object, and update its position.* **bool_t deserialize_ns** (**serialization_t** ***serial**, **chr_t** ****string**, **size_t** ***length**)

Deserialize a null-terminated string from a serialized object, and update its position.

Detailed Description

Distributed cache interface for de-serializing various data types.

Definition in file **deserialization.c**.

Function Documentation

int_t deserialize_count_digits (**chr_t** * **data**, **size_t** *remaining*) [**inline**]

Count the number of bytes before a semicolon or non-numeric character is found.

Parameters:

data a pointer to a null-terminated string to be scanned.

remaining the maximum number of characters to be scanned.

Returns:

-1 if no match is found, or the length, in bytes, of the data preceding the matched character.

Definition at line 21 of file **deserialization.c**.

Referenced by `deserialize_int16()`, `deserialize_int32()`, `deserialize_int64()`, `deserialize_uint16()`, `deserialize_uint32()`, and `deserialize_uint64()`.

`bool_t deserialize_int16 (serialization_t * serial, short * number)`

Deserialize a signed 16 bit integer from a serialized object, and update its position.

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to a signed 16-bit integer which will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure.

Definition at line 223 of file `deserialization.c`.

References `serialization_t::data`, `deserialize_count_digits()`, `int16_conv_bl()`, `length`, `log_pedantic`, `serialization_t::position`, `st_char_get()`, and `st_length_get()`.

`bool_t deserialize_int32 (serialization_t * serial, int_t * number)`

Deserialize a signed 32-bit integer from a serialized object, and update its position.

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to a signed 32-bit integer which will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure.

Definition at line 143 of file `deserialization.c`.

References `serialization_t::data`, `deserialize_count_digits()`, `int32_conv_bl()`, `length`, `log_pedantic`, `serialization_t::position`, `st_char_get()`, and `st_length_get()`.

Referenced by `teacher_data_get()`.

`bool_t deserialize_int64 (serialization_t * serial, long * number)`

Deserialize a signed 64-bit integer from a serialized object, and update its position.

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to a signed 64-bit integer which will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure.

Definition at line 183 of file `deserialization.c`.

References `serialization_t::data`, `deserialize_count_digits()`, `int64_conv_bl()`, `length`, `log_pedantic`, `serialization_t::position`, `st_char_get()`, and `st_length_get()`.

bool_t deserialize_ns (serialization_t * *serial*, chr_t ** *string*, size_t * *length*)

Deserialize a null-terminated string from a serialized object, and update its position.

Note:

A null-terminated string is stored as a 64 bit length field, followed by the string contents.

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

string a pointer to the address of a null-terminated string that will receive the de-serialized data at the serialized object's current position.

length a pointer to a size_t that will receive the length of the de-serialized null-terminated string on success.

Returns:

true on success or false on failure..

Definition at line 357 of file deserialization.c.

References bytes, serialization_t::data, deserialize_sz(), log_pedantic, mm_copy(), ns_alloc(), serialization_t::position, st_char_get(), and st_length_get().

bool_t deserialize_ssz (serialization_t * *serial*, ssize_t * *number*)

Deserialize an ssize_t from a serialized object, and update its position.

See also:

deserialize_int64

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to an ssize_t variable which will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure.

Definition at line 53 of file deserialization.c.

References deserialize_int64().

bool_t deserialize_st (serialization_t * *serial*, stringer_t ** *string*)

Deserialize a managed string from a serialized object, and update its position.

Note:

A managed string is stored as a 64 bit length field, followed by opaque data.

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to the address of a managed string, will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure..

Definition at line 304 of file deserialization.c.

References `serialization_t::data`, `deserialize_sz()`, `length`, `log_pedantic`, `mm_copy()`, `serialization_t::position`, `st_alloc`, `st_char_get()`, `st_length_get()`, and `st_length_set()`.

Referenced by `register_session_get()`, and `teacher_data_get()`.

bool_t deserialize_sz (serialization_t * *serial*, size_t * *number*)

Deserialize an `size_t` from a serialized object, and update its position.

See also:

`deserialize_uint64`

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to an `size_t` variable which will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure.

Definition at line 42 of file `deserialization.c`.

References `deserialize_uint64()`.

Referenced by `deserialize_ns()`, and `deserialize_st()`.

bool_t deserialize_uint16 (serialization_t * *serial*, unsigned short * *number*)

Deserialize an unsigned 16 bit integer from a serialized object, and update its position.

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to an unsigned 16-bit integer which will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure..

Definition at line 263 of file `deserialization.c`.

References `serialization_t::data`, `deserialize_count_digits()`, `length`, `log_pedantic`, `serialization_t::position`, `st_char_get()`, `st_length_get()`, and `uint16_conv_bl()`.

bool_t deserialize_uint32 (serialization_t * *serial*, uint32_t * *number*)

Deserialize an unsigned 32-bit integer from a serialized object, and update its position.

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to an unsigned 32-bit integer which will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure.

Definition at line 103 of file `deserialization.c`.

References `serialization_t::data`, `deserialize_count_digits()`, `length`, `log_pedantic`, `serialization_t::position`, `st_char_get()`, `st_length_get()`, and `uint32_conv_bl()`.

`bool_t deserialize_uint64 (serialization_t * serial, uint64_t * number)`

Deserialize an unsigned 64-bit integer from a serialized object, and update its position.

Parameters:

serial a pointer to a serialized object from which the data will be extracted.

number a pointer to an unsigned 64-bit integer which will receive the de-serialized data at the serialized object's current position.

Returns:

true on success or false on failure.

Definition at line 63 of file `deserialization.c`.

References `serialization_t::data`, `deserialize_count_digits()`, `length`, `log_pedantic`, `serialization_t::position`, `st_char_get()`, `st_length_get()`, and `uint64_conv_bl()`.

Referenced by `deserialize_sz()`, `register_session_get()`, and `teacher_data_get()`.

magma/providers/consumers/serialization.c File Reference

Distributed cache interface for serializing various data types.

```
#include "magma.h"
```

Functions

- **bool_t serialize_sz** (stringer_t **data, size_t number)
- *Serialize a size_t into a managed string.* **bool_t serialize_ssz** (stringer_t **data, ssize_t number)
- *Serialize an ssize_t into a managed string.* **bool_t serialize_int16** (stringer_t **data, int16_t number)
- *Serialize a signed 16-bit integer into a managed string.* **bool_t serialize_uint16** (stringer_t **data, uint16_t number)
- *Serialize an unsigned 16-bit integer into a managed string.* **bool_t serialize_int32** (stringer_t **data, int32_t number)
- *Serialize a signed 32-bit integer into a managed string.* **bool_t serialize_uint32** (stringer_t **data, uint32_t number)
- *Serialize an unsigned 32-bit integer into a managed string.* **bool_t serialize_int64** (stringer_t **data, int64_t number)
- *Serialize a signed 64-bit integer into a managed string.* **bool_t serialize_uint64** (stringer_t **data, int64_t number)
- *Serialize an unsigned 64-bit integer into a managed string.* **bool_t serialize_st** (stringer_t **data, stringer_t *string)
- *Serialize the contents of a managed string into another managed string.* **bool_t serialize_ns** (stringer_t **data, char *string, size_t length)

Serialize a block of memory into a managed string.

Detailed Description

Distributed cache interface for serializing various data types.

Definition in file **serialization.c**.

Function Documentation

bool_t serialize_int16 (stringer_t ** data, int16_t number)

Serialize a signed 16-bit integer into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the signed 16-bit integer to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 44 of file serialization.c.

References `int16_digits()`, `length`, `st_alloc`, `st_avail_get()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, and `st_realloc()`.

`bool_t serialize_int32 (stringer_t ** data, int32_t number)`

Serialize a signed 32-bit integer into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the signed 32-bit integer to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 116 of file `serialization.c`.

References `int32_digits()`, `length`, `st_alloc`, `st_avail_get()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, and `st_realloc()`.

Referenced by `teacher_data_save()`.

`bool_t serialize_int64 (stringer_t ** data, int64_t number)`

Serialize a signed 64-bit integer into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the signed 64-bit integer to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 187 of file `serialization.c`.

References `int64_digits()`, `length`, `st_alloc`, `st_avail_get()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, and `st_realloc()`.

Referenced by `serialize_ssz()`.

`bool_t serialize_ns (stringer_t ** data, char * string, size_t length)`

Serialize a block of memory into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

string a pointer to the start of the memory block to be serialized.

length the length, in bytes, of the memory block to be serialized.

Returns:

1 if the block of memory was serialized and stored successfully, or -1 on failure.

Definition at line 307 of file serialization.c.

References `mm_copy()`, `serialize_sz()`, `st_avail_get()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, and `st_realloc()`.

bool_t serialize_ssz (stringer_t ** data, ssize_t number)

Serialize an `ssize_t` into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the `ssize_t` to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 33 of file serialization.c.

References `serialize_int64()`.

bool_t serialize_st (stringer_t ** data, stringer_t * string)

Serialize the contents of a managed string into another managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

string a pointer to a managed string containing the data to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 259 of file serialization.c.

References `length`, `mm_copy()`, `serialize_sz()`, `st_avail_get()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, and `st_realloc()`.

Referenced by register_session_cache(), and teacher_data_save().

bool_t serialize_sz (stringer_t ** data, size_t number)

Serialize a size_t into a managed string.

Serialization.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the size_t to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 22 of file serialization.c.

References serialize_uint64().

Referenced by serialize_ns(), and serialize_st().

bool_t serialize_uint16 (stringer_t ** data, uint16_t number)

Serialize an unsigned 16-bit integer into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the unsigned 16-bit integer to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 80 of file serialization.c.

References length, st_alloc, st_avail_get(), st_char_get(), st_length_get(), st_length_set(), st_realloc(), and uint16_digits().

Referenced by register_session_cache().

bool_t serialize_uint32 (stringer_t ** data, uint32_t number)

Serialize an unsigned 32-bit integer into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the unsigned 32-bit integer to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 152 of file serialization.c.

References `length`, `st_alloc`, `st_avail_get()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, `st_realloc()`, and `uint32_digits()`.

bool_t serialize_uint64 (stringer_t ** data, int64_t number)

Serialize an unsigned 64-bit integer into a managed string.

Note:

This function will reallocate the output managed string if necessary, and append the serialized data, updating the length field to reflect the changes.

Parameters:

data a pointer to the address of a managed string to receive the output, which will be allocated for the caller if NULL is passed.

number the value of the unsigned 64-bit integer to be serialized.

Returns:

true if the specified value was serialized and stored successfully, or false on failure.

Definition at line 223 of file serialization.c.

References `length`, `st_alloc`, `st_avail_get()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, `st_realloc()`, and `uint64_digits()`.

Referenced by `register_session_cache()`, `serialize_sz()`, and `teacher_data_save()`.

magma/providers/cryptography/ciphers.c File Reference

Functions used to handle cryptographic ciphers.

```
#include "magma.h"
```

Functions

- **cipher_t * cipher_name** (stringer_t *name)
 - *Look up a cipher type by name.* **cipher_t * cipher_id** (int_t id)
 - *Look up a cipher type by its nid.* **int_t cipher_numeric_id** (cipher_t *c)
 - *Return the numerical ID (nid) of a specified cipher.* **int_t cipher_key_length** (cipher_t *c)
 - *Determine the key length of a specified cipher.* **int_t cipher_vector_length** (cipher_t *c)
 - *Determine the IV length of a specified cipher.* **int_t cipher_block_length** (cipher_t *c)
- Determine the block length of a specified cipher.*
-

Detailed Description

Functions used to handle cryptographic ciphers.

Definition in file **ciphers.c**.

Function Documentation

int_t cipher_block_length (cipher_t * c)

Determine the block length of a specified cipher.

Parameters:

c the input cipher type.

Returns:

-1 on failure, or the cipher's block length in bytes.

Definition at line 100 of file ciphers.c.

References `EVP_CIPHER_block_size_d`, `EVP_CIPHER_nid_d`, `log_pedantic`, and `OBJ_nid2sn_d`.

cipher_t* cipher_id (int_t id)

Look up a cipher type by its nid.

ciphers.c

Parameters:

id the numerical ID (nid) of the specified cipher.

Returns:

NULL on failure, or the relevant cipher data structure on success.

Definition at line 36 of file ciphers.c.

References `EVP_get_cipherbyname_d`, `log_pedantic`, and `OBJ_nid2sn_d`.

Referenced by `scramble_decrypt()`, and `scramble_encrypt()`.

int_t cipher_key_length (cipher_t * c)

Determine the key length of a specified cipher.

Parameters:

c the input cipher type.

Returns:

-1 on failure, or the cipher's key length in bytes.

Definition at line 68 of file `ciphers.c`.

References `EVP_CIPHER_key_length_d`, `EVP_CIPHER_nid_d`, `log_pedantic`, and `OBJ_nid2sn_d`.

Referenced by `symmetric_key()`.

cipher_t* cipher_name (stringer_t * name)

Look up a cipher type by name.

Parameters:

name a descriptive name of the cipher to be looked up.

Returns:

NULL on failure, or the relevant cipher data structure on success.

Definition at line 21 of file `ciphers.c`.

References `EVP_get_cipherbyname_d`, `log_pedantic`, `st_char_get()`, `st_empty()`, and `st_length_int()`.

int_t cipher_numeric_id (cipher_t * c)

Return the numerical ID (nid) of a specified cipher.

Parameters:

c the input cipher type.

Returns:

NID_undef on failure, or the nid of the specified cipher.

Definition at line 52 of file `ciphers.c`.

References `EVP_CIPHER_nid_d`, and `log_pedantic`.

Referenced by `scramble_encrypt()`.

int_t cipher_vector_length (cipher_t * c)

Determine the IV length of a specified cipher.

Parameters:

c the input cipher type.

Returns:

-1 on failure, or the cipher's IV length in bytes.

Definition at line 84 of file ciphers.c.

References `EVP_CIPHER_iv_length_d`, `EVP_CIPHER_nid_d`, `log_pedantic`, and `OBJ_nid2sn_d`.

Referenced by `symmetric_vector()`.

magma/providers/cryptography/cryptex.c File Reference

Functions for handling the secure data type.

```
#include "magma.h"
```

Functions

- `uint64_t cryptex_envelope_length (cryptex_t *cryptex)`
 - *Get the length of a cryptex envelope.* `uint64_t cryptex_hmac_length (cryptex_t *cryptex)`
 - *Get the length of a cryptex HMAC.* `uint64_t cryptex_body_length (cryptex_t *cryptex)`
 - *Get the length of a cryptex object's encrypted data body.* `uint64_t cryptex_original_length (cryptex_t *cryptex)`
 - *Get the original length of a cryptex object's data buffer.* `uint64_t cryptex_total_length (cryptex_t *cryptex)`
 - *Determine the total length of the data underlying a cryptex object.* `void * cryptex_envelope_data (cryptex_t *cryptex)`
 - *Get a pointer to the envelope data of a cryptex object.* `void * cryptex_mac_data (cryptex_t *cryptex)`
 - *Get a pointer to the HMAC data of a cryptex object.* `void * cryptex_body_data (cryptex_t *cryptex)`
 - *Get a pointer to the encrypted data body of a cryptex object.* `void * cryptex_alloc (uint64_t envelope, uint64_t hmac, uint64_t original, uint64_t body)`
 - *Allocate a new cryptex object.* `void cryptex_free (cryptex_t *cryptex)`
- Free a cryptex object.*
-

Detailed Description

Functions for handling the secure data type.

Definition in file **cryptex.c**.

Function Documentation

void* cryptex_alloc (uint64_t envelope, uint64_t hmac, uint64_t original, uint64_t body)

Allocate a new cryptex object.

Definition at line 114 of file cryptex.c.

References `log_pedantic`, and `mm_alloc()`.

Referenced by `ecies_encrypt()`.

void* cryptex_body_data (cryptex_t * cryptex)

Get a pointer to the encrypted data body of a cryptex object.

Parameters:

cryptex the input cryptex object.

Returns:

a pointer to the cryptex object's encrypted data body.

Definition at line 103 of file cryptex.c.

Referenced by `ecies_decrypt()`, and `ecies_encrypt()`.

uint64_t cryptex_body_length (cryptex_t * *cryptex*)

Get the length of a cryptex object's encrypted data body.

cryptex.c

Parameters:

cryptex a pointer to the head of the cryptex object to be examined.

Returns:

the length, in bytes, of the cryptex encrypted data body.

Definition at line 44 of file `cryptex.c`.

Referenced by `ecies_decrypt()`, and `ecies_encrypt()`.

void* cryptex_envelope_data (cryptex_t * *cryptex*)

Get a pointer to the envelope data of a cryptex object.

Parameters:

cryptex the input cryptex object.

Returns:

a pointer to the cryptex object's envelope data.

Definition at line 80 of file `cryptex.c`.

Referenced by `ecies_decrypt()`, and `ecies_encrypt()`.

uint64_t cryptex_envelope_length (cryptex_t * *cryptex*)

Get the length of a cryptex envelope.

Parameters:

cryptex a pointer to the head of the cryptex object to be examined.

Returns:

the length, in bytes, of the cryptex envelope.

Definition at line 20 of file `cryptex.c`.

Referenced by `ecies_decrypt()`.

void cryptex_free (cryptex_t * *cryptex*)

Free a cryptex object.

Parameters:

cryptex the cryptex object to be freed.

Returns:

This function returns no value.

Definition at line 138 of file cryptex.c.

Referenced by `adjust_message_encryption()`, `ecies_encrypt()`, and `mail_store_message()`.

uint64_t cryptex_hmac_length (cryptex_t * cryptex)

Get the length of a cryptex HMAC.

Parameters:

cryptex a pointer to the head of the cryptex object to be examined.

Returns:

the length, in bytes, of the cryptex HMAC.

Definition at line 32 of file cryptex.c.

Referenced by `ecies_decrypt()`, and `ecies_encrypt()`.

void* cryptex_mac_data (cryptex_t * cryptex)

Get a pointer to the HMAC data of a cryptex object.

Parameters:

cryptex the input cryptex object.

Returns:

a pointer to the cryptex object's hmac data.

Definition at line 91 of file cryptex.c.

Referenced by `ecies_decrypt()`, and `ecies_encrypt()`.

uint64_t cryptex_original_length (cryptex_t * cryptex)

Get the original length of a cryptex object's data buffer.

Parameters:

cryptex a pointer to the head of the cryptex object to be examined.

Returns:

the length, in bytes, of the cryptex object's original data.

Definition at line 56 of file cryptex.c.

Referenced by `ecies_decrypt()`.

uint64_t cryptex_total_length (cryptex_t * cryptex)

Determine the total length of the data underlying a cryptex object.

Parameters:

cryptex the input cryptex object.

Returns:

the total length of the associated cryptex data in bytes.

Definition at line 68 of file cryptex.c.

Referenced by `adjust_message_encryption()`, and `mail_store_message()`.

magma/providers/cryptography/cryptography.h File Reference

Functions used to perform cryptographic operations and provide truly random data.

Data Structures

- struct `__attribute__`
- struct `__attribute__`

Defines

- #define `BN_num_bytes_d(a)` $((\text{BN_num_bits_d}(a)+7)/8)$
- #define `OPENSSL_free_d(addr)` `CRYPTO_free_d(addr)`
- #define
`MAGMA_CIPHER_LIST` "EECDH+ECDSA+AESGCM:EECDH+aRSA+AESGCM:EECDH+ECDSA+SHA384:EECDH+ECDSA+SHA256:EECDH+aRSA+SHA384:EECDH+aRSA+SHA256:EECDH+aRSA+RC4:EDH:EDH+aRSA:RC4:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!SSLv2:!RC4-SHA:!SEED"
- #define
`MAGMA_CIPHER_LIST_RC4` "RC4:EECDH+ECDSA+AESGCM:EECDH+aRSA+AESGCM:EECDH+ECDSA+SHA384:EECDH+ECDSA+SHA256:EECDH+aRSA+SHA384:EECDH+aRSA+SHA256:EECDH+aRSA+RC4:EDH:EDH+aRSA:RC4:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!SSLv2:!RC4-SHA"
- #define
`MAGMA_CIPHER_LIST_PLANB` "RC4:kEDH:HIGH:!aNULL:!eNULL:!EXP:!LOW:!SSLv2:!MD5"
- #define
`MAGMA_CIPHER_LIST_PLANC` "EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+AESGCM:EECDH+aRSA+SHA384:EECDH+aRSA+SHA256:EECDH:+CAMELLIA256:+AES256:+CAMELLIA128:+AES128:+SSLv3:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!DSS:!RC4:!SEED:!ECDSA:CAMELLIA256-SHA:AES256-SHA:CAMELLIA128-SHA:AES128-SHA"
- #define `ECIES_HMAC` `NID_sha512`
- #define `ECIES_CURVE` `NID_sect571k1`
- #define `ECIES_CIPHER` `NID_aes_256_cbc`
- #define `ECIES_ENVELOPE` `NID_sha512`

Typedefs

- typedef void * `digest_t`
- typedef void * `cipher_t`
- typedef char * `cryptex_t`
- typedef `stringer_t` `scramble_t`

Enumerations

- enum `ECIES_KEY_TYPE` { `ECIES_PRIVATE_HEX` = 1, `ECIES_PRIVATE_BINARY` = 2, `ECIES_PUBLIC_HEX` = 4, `ECIES_PUBLIC_BINARY` = 8 }

Functions

- `cipher_t * cipher_id(int_t id)`
- *ciphers.c* `cipher_t * cipher_name(stringer_t *name)`
- *Look up a cipher type by name.* `int_t cipher_numeric_id(cipher_t *c)`
- *Return the numerical ID (nid) of a specified cipher.* `int_t cipher_block_length(cipher_t *c)`
- *Determine the block length of a specified cipher.* `int_t cipher_key_length(cipher_t *c)`
- *Determine the key length of a specified cipher.* `int_t cipher_vector_length(cipher_t *c)`

- Determine the IV length of a specified cipher. `uint64_t cryptex_body_length (cryptex_t *cryptex)`
- `cryptex.c` `uint64_t cryptex_envelope_length (cryptex_t *cryptex)`
- Get the length of a cryptex envelope. `uint64_t cryptex_hmac_length (cryptex_t *cryptex)`
- Get the length of a cryptex HMAC. `uint64_t cryptex_original_length (cryptex_t *cryptex)`
- Get the original length of a cryptex object's data buffer. `uint64_t cryptex_total_length (cryptex_t *cryptex)`
- Determine the total length of the data underlying a cryptex object. `void * cryptex_alloc (uint64_t envelope, uint64_t hmac, uint64_t original, uint64_t body)`
- Allocate a new cryptex object. `void * cryptex_body_data (cryptex_t *cryptex)`
- Get a pointer to the encrypted data body of a cryptex object. `void * cryptex_envelope_data (cryptex_t *cryptex)`
- Get a pointer to the envelope data of a cryptex object. `void * cryptex_mac_data (cryptex_t *cryptex)`
- Get a pointer to the HMAC data of a cryptex object. `void cryptex_free (cryptex_t *cryptex)`
- Free a cryptex object. `EC_GROUP * ecies_group (uint64_t curve, bool_t precompute)`
- `ecies.c` `EC_KEY * ecies_key_alloc (void)`
- Allocate a new ECIES key pair from the curve defined by `ECIES_CURVE`. `EC_KEY * ecies_key_create (void)`
- Generate a random ECIES key pair. `EC_KEY * ecies_key_private (uint64_t format, placer_t data)`
- `EC_KEY * ecies_key_public (uint64_t format, placer_t data)`
- `bool_t ecies_start (void)`
- `stringer_t * ecies_key_private_hex (EC_KEY *key)`
- Return an ECIES private key as a hex string. `char * ecies_key_private_bin (EC_KEY *key, size_t *olen)`
- Return an ECIES private key as binary data. `stringer_t * ecies_key_public_hex (EC_KEY *key)`
- Return an ECIES public key as a null-terminated hex string. `unsigned char * ecies_key_public_bin (EC_KEY *key, size_t *olen)`
- Return an ECIES public key as binary data. `cryptex_t * ecies_encrypt (stringer_t *key, ECIES_KEY_TYPE key_type, unsigned char *data, size_t length)`
- Encrypt a block of data using an ECIES public key. `unsigned char * ecies_decrypt (stringer_t *key, ECIES_KEY_TYPE key_type, cryptex_t *cryptex, size_t *length)`
- Decrypt a block of data using an ECIES private key. `void * ecies_envelope_derivation (const void *input, size_t ilen, void *output, size_t *olen)`
- `void ecies_key_free (EC_KEY *key)`
- Free an ECIES key pair. `void ecies_stop (void)`
- Destroy all initialized ECIES curve group information. `digest_t * digest_id (int_t id)`
- `digest.c` `digest_t * digest_name (stringer_t *name)`
- `stringer_t * digest_hash (digest_t *digest, stringer_t *s, stringer_t *output)`
- `stringer_t * digest_md4 (stringer_t *s, stringer_t *output)`
- `stringer_t * digest_md5 (stringer_t *s, stringer_t *output)`
- `stringer_t * digest_ripemd160 (stringer_t *s, stringer_t *output)`
- `stringer_t * digest_sha (stringer_t *s, stringer_t *output)`
- `stringer_t * digest_sha1 (stringer_t *s, stringer_t *output)`
- `stringer_t * digest_sha224 (stringer_t *s, stringer_t *output)`
- `stringer_t * digest_sha256 (stringer_t *s, stringer_t *output)`
- `stringer_t * digest_sha384 (stringer_t *s, stringer_t *output)`
- `stringer_t * digest_sha512 (stringer_t *s, stringer_t *output)`
- `bool_t lib_load_openssl (void)`
- `openssl.c` `const char * lib_version_openssl (void)`
- Return the version string of the openssl library. `int ssl_shutdown_get (SSL *ssl)`
- Checks whether an SSL tunnel has been shut down or not. `SSL * ssl_alloc (void *server, int sockd, int flags)`
- Create an SSL session for a file descriptor, and accept the client TLS/SSL handshake. `void * ssl_client_create (int_t sockd)`
- Establish an SSL client wrapper around a socket descriptor. `char * ssl_error_string (chr_t *buffer, int_t length)`
- Get a textual representation of the last openssl error message. `void ssl_free (SSL *ssl)`
- Shutdown and free an SSL connection. `void ssl_locking_callback (int mode, int n, const char *file, int line)`

- *The locking function callback necessary for all multi-threaded applications using openssl.* **int ssl_print** (SSL *ssl, const char *format, va_list args)
- *Write formatted data to an SSL connection.* **int ssl_read** (SSL *ssl, void *buffer, int length, **bool_t** block)
- *Read data from an SSL connection.* **bool_t ssl_server_create** (void *server)
- **void ssl_server_destroy** (void *server)
- *Destroy an SSL context associated with a server.* **bool_t ssl_start** (void)
- *Initialize the openssl facility.* **void ssl_stop** (void)
- *Stop the openssl library and cleanup all associated data structures.* **void ssl_thread_stop** (void)
- *Free the calling thread's SSL error queue.* **int ssl_write** (SSL *ssl, const void *buffer, int length)
- *Write data to an open SSL connection.* **bool_t ssl_verify_privkey** (const char *keyfile)
- *Verify that the filename contains a valid private key in PEM format.* **DH * ssl_dh_exchange_callback** (SSL *ssl, int is_export, int keylength)
- *Callback handler for the Diffie-Hellman parameter generation process necessary for PFS.* **int ssl_dh_generate_callback** (int p, int n, BN_GENCB *cb)
- *The DH param generation callback.* **EC_KEY * ssl_ecdh_exchange_callback** (SSL *ssl, int is_export, int keylength)
- *Callback handler for the EC Diffie-Hellman parameter generation process necessary for PFS.* **bool_t rand_start** (void)
- **random.c** **bool_t rand_thread_start** (void)
- **int16_t rand_get_int16** (void)
- **int32_t rand_get_int32** (void)
- **int64_t rand_get_int64** (void)
- *Generate a random signed 64 bit number.* **int8_t rand_get_int8** (void)
- **size_t rand_write** (**stringer_t** *s)
- *Fill a managed string with random data.* **stringer_t * rand_choices** (**chr_t** *choices, **size_t** len)
- *Get a random string of data of a specified size, populated with characters from a chosen set.* **uint16_t rand_get_uint16** (void)
- *Generate a random unsigned 16 bit number.* **uint32_t rand_get_uint32** (void)
- *Generate a random unsigned 32 bit number.* **uint64_t rand_get_uint64** (void)
- *Generate a random unsigned 64 bit number.* **uint8_t rand_get_uint8** (void)
- *Generate a random unsigned 8 bit number.* **void rand_stop** (void)
- **scramble_t * scramble_alloc** (**size_t** length)
- **scramble.c** **void * scramble_body_data** (**scramble_t** *buffer)
- *Get a pointer to the start of the encrypted data from a scrambled data header.* **uint64_t scramble_body_hash** (**scramble_t** *buffer)
- *Get a hash of a scrambled object's (encrypted) body data.* **uint64_t scramble_body_length** (**scramble_t** *buffer)
- *Get the length of a scrambled object's (encrypted) body data.* **stringer_t * scramble_decrypt** (**stringer_t** *key, **scramble_t** *input)
- *Un-scramble a block of data using an decryption key.* **scramble_t * scramble_encrypt** (**stringer_t** *key, **stringer_t** *input)
- *Scramble a block of data using an encryption key.* **void scramble_free** (**scramble_t** *buffer)
- *Free a scrambled data block.* **scramble_t * scramble_import** (**stringer_t** *s)
- *Return a managed string as a scrambled buffer, after validation.* **uint64_t scramble_orig_length** (**scramble_t** *buffer)
- *Get the length of the original (unencrypted) data underlying a scrambled object.* **uint64_t scramble_total_length** (**scramble_t** *buffer)
- *TODO: Create envelope functions to translate the scrambled buffer into a condensed/encoded stringer.* **void * scramble_vector_data** (**scramble_t** *buffer)
- *Get a pointer to the start of the IV from a scrambled data header.* **uint64_t scramble_vector_length** (**scramble_t** *buffer)
- *Get the length of a scrambled object's IV.* **stringer_t * symmetric_decrypt** (**cipher_t** *cipher, **stringer_t** *vector, **stringer_t** *key, **stringer_t** *input)

- *symmetric.c* **stringer_t * symmetric_encrypt** (**cipher_t** *cipher, **stringer_t** *vector, **stringer_t** *key, **stringer_t** *input)
- *Encrypt a block of data using a symmetric cipher.* **stringer_t * symmetric_key** (**cipher_t** *cipher, **stringer_t** *key, **stringer_t** *output)
- *Derive a symmetric key from a user's password.* **stringer_t * symmetric_vector** (**cipher_t** *cipher, **stringer_t** *output)

Generate an IV data suitable for the specified cipher.

Detailed Description

Functions used to perform cryptographic operations and provide truly random data.

Definition in file **cryptography.h**.

Define Documentation

#define BN_num_bytes_d(a) ((BN_num_bits_d(a)+7)/8)

Definition at line 17 of file cryptography.h.

Referenced by `ecies_key_private_bin()`.

#define ECIES_CIPHER NID_aes_256_cbc

Definition at line 30 of file cryptography.h.

Referenced by `ecies_decrypt()`, `ecies_encrypt()`, and `ecies_start()`.

#define ECIES_CURVE NID_sect571k1

Definition at line 29 of file cryptography.h.

Referenced by `ecies_key_alloc()`, and `ecies_start()`.

#define ECIES_ENVELOPE NID_sha512

Definition at line 31 of file cryptography.h.

Referenced by `ecies_start()`.

#define ECIES_HMAC NID_sha512

Definition at line 28 of file cryptography.h.

Referenced by `ecies_decrypt()`, `ecies_encrypt()`, and `ecies_start()`.

#define

MAGMA_CIPHER_LIST "EECDH+ECDSA+AESGCM:EECDH+aRSA+AESGCM:EECDH+ECDSA+SHA384:EECDH+ECDSA+SHA256:EECDH+aRSA+SHA384:EECDH+aRSA+SHA256:EECDH+aRSA+RC

```
4:EECDH:EDH+aRSA:RC4  
:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!SSLv2:!RC4-SHA:!SEED"
```

Definition at line 21 of file cryptography.h.

Referenced by ssl_server_create().

```
#define  
MAGMA_CIPHER_LIST_PLANB "RC4:kEDH:HIGH:!aNULL:!eNULL:!EXP:!LOW:!SSLv2:!MD5"
```

Definition at line 23 of file cryptography.h.

```
#define  
MAGMA_CIPHER_LIST_PLANC "EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA:AESGCM:EECDH+a  
RSA+SHA384:EECDH+aRSA+SHA256:EECDH:+CAMELLIA256:+AES256:+CAMELLIA128:+AES128  
:+SSLv3:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!DSS:!RC4:!SEED:!ECDSA:CAMELLIA256-  
SHA:AES256-SHA:CAMELLIA128-SHA:AES128-SHA"
```

Definition at line 24 of file cryptography.h.

```
#define  
MAGMA_CIPHER_LIST_RC4 "RC4:EECDH+ECDSA:AESGCM:EECDH+aRSA:AESGCM:EECDH+E  
CDSA+SHA384:EECDH+ECDSA+SHA256:EECDH+aRSA+SHA384:EECDH+aRSA+SHA256:EECDH+  
aRSA+RC4:EECDH:EDH+aRSA:RC4:!aNULL:!eNULL:!LOW:!3DES:!MD5:!EXP:!PSK:!SRP:!DSS:!SS  
Lv2:!RC4-SHA"
```

Definition at line 22 of file cryptography.h.

```
#define OPENSSL_free_d(addr) CRYPTO_free_d(addr)
```

Definition at line 18 of file cryptography.h.

Referenced by ecies_key_private_hex(), and ecies_key_public_hex().

Typedef Documentation

```
typedef void* cipher_t
```

Definition at line 68 of file cryptography.h.

```
typedef char* cryptex_t
```

Definition at line 69 of file cryptography.h.

```
typedef void* digest_t
```

Definition at line 67 of file cryptography.h.

typedef stringer_t scramble_t

Definition at line 70 of file cryptography.h.

Enumeration Type Documentation

enum ECIES_KEY_TYPE

Enumerator:

ECIES_PRIVATE_HEX
ECIES_PRIVATE_BINARY
ECIES_PUBLIC_HEX
ECIES_PUBLIC_BINARY

Definition at line 33 of file cryptography.h.

Function Documentation

int_t cipher_block_length (cipher_t * c)

Determine the block length of a specified cipher.

Parameters:

c the input cipher type.

Returns:

-1 on failure, or the cipher's block length in bytes.

Definition at line 100 of file ciphers.c.

References `EVP_CIPHER_block_size_d`, `EVP_CIPHER_nid_d`, `log_pedantic`, and `OBJ_nid2sn_d`.

cipher_t* cipher_id (int_t id)

`ciphers.c`

`ciphers.c`

Parameters:

id the numerical ID (nid) of the specified cipher.

Returns:

NULL on failure, or the relevant cipher data structure on success.

Definition at line 36 of file ciphers.c.

References `EVP_get_cipherbyname_d`, `log_pedantic`, and `OBJ_nid2sn_d`.

Referenced by `scramble_decrypt()`, and `scramble_encrypt()`.

int_t cipher_key_length (cipher_t * c)

Determine the key length of a specified cipher.

Parameters:

c the input cipher type.

Returns:

-1 on failure, or the cipher's key length in bytes.

Definition at line 68 of file ciphers.c.

References EVP_CIPHER_key_length_d, EVP_CIPHER_nid_d, log_pedantic, and OBJ_nid2sn_d.

Referenced by symmetric_key().

cipher_t* cipher_name (stringer_t * name)

Look up a cipher type by name.

Parameters:

name a descriptive name of the cipher to be looked up.

Returns:

NULL on failure, or the relevant cipher data structure on success.

Definition at line 21 of file ciphers.c.

References EVP_get_cipherbyname_d, log_pedantic, st_char_get(), st_empty(), and st_length_int().

int_t cipher_numeric_id (cipher_t * c)

Return the numerical ID (nid) of a specified cipher.

Parameters:

c the input cipher type.

Returns:

NID_undef on failure, or the nid of the specified cipher.

Definition at line 52 of file ciphers.c.

References EVP_CIPHER_nid_d, and log_pedantic.

Referenced by scramble_encrypt().

int_t cipher_vector_length (cipher_t * c)

Determine the IV length of a specified cipher.

Parameters:

c the input cipher type.

Returns:

-1 on failure, or the cipher's IV length in bytes.

Definition at line 84 of file ciphers.c.

References EVP_CIPHER_iv_length_d, EVP_CIPHER_nid_d, log_pedantic, and OBJ_nid2sn_d.

Referenced by symmetric_vector().

void* cryptex_alloc (uint64_t *envelope*, uint64_t *hmac*, uint64_t *original*, uint64_t *body*)

Allocate a new cryptex object.

Definition at line 114 of file cryptex.c.

References log_pedantic, and mm_alloc().

Referenced by ecies_encrypt().

void* cryptex_body_data (cryptex_t * *cryptex*)

Get a pointer to the encrypted data body of a cryptex object.

Parameters:

cryptex the input cryptex object.

Returns:

a pointer to the cryptex object's encrypted data body.

Definition at line 103 of file cryptex.c.

Referenced by ecies_decrypt(), and ecies_encrypt().

uint64_t cryptex_body_length (cryptex_t * *cryptex*)

cryptex.c

cryptex.c

Parameters:

cryptex a pointer to the head of the cryptex object to be examined.

Returns:

the length, in bytes, of the cryptex encrypted data body.

Definition at line 44 of file cryptex.c.

Referenced by ecies_decrypt(), and ecies_encrypt().

void* cryptex_envelope_data (cryptex_t * *cryptex*)

Get a pointer to the envelope data of a cryptex object.

Parameters:

cryptex the input cryptex object.

Returns:

a pointer to the cryptex object's envelope data.

Definition at line 80 of file cryptex.c.

Referenced by ecies_decrypt(), and ecies_encrypt().

uint64_t cryptex_envelope_length (cryptex_t * cryptex)

Get the length of a cryptex envelope.

Parameters:

cryptex a pointer to the head of the cryptex object to be examined.

Returns:

the length, in bytes, of the cryptex envelope.

Definition at line 20 of file cryptex.c.

Referenced by ecies_decrypt().

void cryptex_free (cryptex_t * cryptex)

Free a cryptex object.

Parameters:

cryptex the cryptex object to be freed.

Returns:

This function returns no value.

Definition at line 138 of file cryptex.c.

Referenced by adjust_message_encryption(), ecies_encrypt(), and mail_store_message().

uint64_t cryptex_hmac_length (cryptex_t * cryptex)

Get the length of a cryptex HMAC.

Parameters:

cryptex a pointer to the head of the cryptex object to be examined.

Returns:

the length, in bytes, of the cryptex HMAC.

Definition at line 32 of file cryptex.c.

Referenced by ecies_decrypt(), and ecies_encrypt().

void* cryptex_mac_data (cryptex_t * cryptex)

Get a pointer to the HMAC data of a cryptex object.

Parameters:

cryptex the input cryptex object.

Returns:

a pointer to the cryptex object's hmac data.

Definition at line 91 of file cryptex.c.

Referenced by ecies_decrypt(), and ecies_encrypt().

uint64_t cryptex_original_length (cryptex_t * cryptex)

Get the original length of a cryptex object's data buffer.

Parameters:

cryptex a pointer to the head of the cryptex object to be examined.

Returns:

the length, in bytes, of the cryptex object's original data.

Definition at line 56 of file cryptex.c.

Referenced by ecies_decrypt().

uint64_t cryptex_total_length (cryptex_t * cryptex)

Determine the total length of the data underlying a cryptex object.

Parameters:

cryptex the input cryptex object.

Returns:

the total length of the associated cryptex data in bytes.

Definition at line 68 of file cryptex.c.

Referenced by adjust_message_encryption(), and mail_store_message().

stringer_t* digest_hash (digest_t * digest, stringer_t * s, stringer_t * output)

Definition at line 36 of file digest.c.

References EVP_DigestFinal_d, EVP_DigestInit_ex_d, EVP_DigestUpdate_d, EVP_MD_CTX_cleanup_d, EVP_MD_CTX_init_d, EVP_MD_size_d, log_pedantic, st_alloc, st_avail_get(), st_data_get(), st_empty(), st_free(), st_length_get(), st_length_set(), st_valid_avail(), st_valid_destination(), and st_valid_tracked().

Referenced by digest_md4(), digest_md5(), digest_ripemd160(), digest_sha(), digest_sha1(), digest_sha224(), digest_sha256(), digest_sha384(), and digest_sha512().

digest_t* digest_id (int_t id)**digest.c**

Definition at line 25 of file digest.c.

References EVP_get_digestbyname_d, log_pedantic, and OBJ_nid2sn_d.

stringer_t* digest_md4 (stringer_t * s, stringer_t * output)

Definition at line 112 of file digest.c.

References digest_hash(), and EVP_md4_d.

stringer_t* digest_md5 (stringer_t * s, stringer_t * output)

Definition at line 116 of file digest.c.

References digest_hash(), and EVP_md5_d.

digest_t* digest_name (stringer_t * name)

Definition at line 15 of file digest.c.

References EVP_get_digestbyname_d, log_pedantic, st_char_get(), st_empty(), and st_length_int().

stringer_t* digest_ripemd160 (stringer_t * s, stringer_t * output)

Definition at line 144 of file digest.c.

References digest_hash(), and EVP_ripemd160_d.

stringer_t* digest_sha (stringer_t * s, stringer_t * output)

Definition at line 120 of file digest.c.

References digest_hash(), and EVP_sha_d.

stringer_t* digest_sha1 (stringer_t * s, stringer_t * output)

Definition at line 124 of file digest.c.

References digest_hash(), and EVP_sha1_d.

Referenced by symmetric_key().

stringer_t* digest_sha224 (stringer_t * s, stringer_t * output)

Definition at line 128 of file digest.c.

References digest_hash(), and EVP_sha224_d.

Referenced by symmetric_key().

stringer_t* digest_sha256 (stringer_t * s, stringer_t * output)

Definition at line 132 of file digest.c.

References digest_hash(), and EVP_sha256_d.

Referenced by symmetric_key().

stringer_t* digest_sha384 (stringer_t * s, stringer_t * output)

Definition at line 136 of file digest.c.

References digest_hash(), and EVP_sha384_d.

stringer_t* digest_sha512 (stringer_t * s, stringer_t * output)

Definition at line 140 of file digest.c.

References digest_hash(), and EVP_sha512_d.

Referenced by credential_alloc_auth().

unsigned char* ecies_decrypt (stringer_t * key, ECIES_KEY_TYPE key_type, cryptex_t * cryptex, size_t * length)

Decrypt a block of data using an ECIES private key.

Parameters:

key the ECIES private key in the specified format.

key_type the encoding type of the ECIES private key (ECIES_PRIVATE_BINARY or ECIES_PRIVATE_HEX).

cryptex a pointer to the head of the cryptex object with the encrypted data.

length a pointer to a size_t variable which will receive the final length of the unencrypted data.

Returns:

NULL on failure, or a pointer to a memory address containing the decrypted data on success..

Definition at line 595 of file ecies.c.

References cryptex_body_data(), cryptex_body_length(), cryptex_envelope_data(), cryptex_envelope_length(), cryptex_hmac_length(), cryptex_mac_data(), cryptex_original_length(), EC_KEY_free_d, EC_KEY_get0_public_key_d, ECDH_compute_key_d, ECIES_CIPHER, ecies_envelope_derivation(), ECIES_HMAC, ecies_key_private(), ecies_key_public(), ECIES_PRIVATE_BINARY, ECIES_PRIVATE_HEX, ECIES_PUBLIC_BINARY, ERR_error_string_d, ERR_get_error_d, EVP_CIPHER_CTX_cleanup_d, EVP_CIPHER_CTX_init_d, EVP_CIPHER_CTX_set_padding_d, EVP_CIPHER_key_length_d, EVP_DecryptFinal_ex_d, EVP_DecryptInit_ex_d, EVP_DecryptUpdate_d, EVP_get_cipherbyname_d, EVP_get_digestbyname_d, HMAC_CTX_cleanup_d, HMAC_CTX_init_d, HMAC_Final_d, HMAC_Init_ex_d, HMAC_Update_d, log_info, mm_alloc(), OBJ_nid2sn_d, pl_init(), and st_empty_out().

Referenced by adjust_message_encryption(), mail_load_header(), and mail_load_message().

cryptex_t* ecies_encrypt (stringer_t * key, ECIES_KEY_TYPE key_type, unsigned char * data, size_t length)

Encrypt a block of data using an ECIES public key.

Parameters:

key the ECIES public key in the specified format.

key_type the encoding type of the ECIES public key (ECIES_PUBLIC_BINARY or ECIES_PUBLIC_HEX).

data a pointer to the block of data to be encrypted.

length the length, in bytes, of the data to be encrypted.

Returns:

NULL on failure, or a pointer to the header of the cryptex object containing the encrypted data on success..
 Definition at line 424 of file ecies.c.

References cryptex_alloc(), cryptex_body_data(), cryptex_body_length(), cryptex_envelope_data(), cryptex_free(), cryptex_hmac_length(), cryptex_mac_data(), EC_KEY_free_d, EC_KEY_get0_group_d, EC_KEY_get0_public_key_d, EC_POINT_point2oct_d, ECDH_compute_key_d, ECIES_CIPHER, ecies_envelope_derivation(), ECIES_HMAC, ecies_key_create(), ecies_key_public(), ECIES_PUBLIC_BINARY, ECIES_PUBLIC_HEX, ERR_error_string_d, ERR_get_error_d, EVP_CIPHER_block_size_d, EVP_CIPHER_CTX_cleanup_d, EVP_CIPHER_CTX_init_d, EVP_CIPHER_CTX_set_padding_d, EVP_CIPHER_key_length_d, EVP_EncryptFinal_ex_d, EVP_EncryptInit_ex_d, EVP_EncryptUpdate_d, EVP_get_cipherbyname_d, EVP_get_digestbyname_d, EVP_MD_size_d, HMAC_CTX_cleanup_d, HMAC_CTX_init_d, HMAC_Final_d, HMAC_Init_ex_d, HMAC_Update_d, log_info, OBJ_nid2sn_d, pl_init(), and st_empty_out().

Referenced by adjust_message_encryption(), and mail_store_message().

void* ecies_envelope_derivation (const void * input, size_t ilen, void * output, size_t * olen)

Definition at line 407 of file ecies.c.

References ecies_envelope_evp, and EVP_Digest_d.

Referenced by ecies_decrypt(), and ecies_encrypt().

EC_GROUP* ecies_group (uint64_t curve, bool_t precompute)

ecies.c

Definition at line 94 of file ecies.c.

References EC_GROUP_free_d, EC_GROUP_new_by_curve_name_d, EC_GROUP_precompute_mult_d, EC_GROUP_set_point_conversion_form_d, ERR_error_string_d, ERR_get_error_d, and log_error.

Referenced by ecies_start().

EC_KEY* ecies_key_alloc (void)

Allocate a new ECIES key pair from the curve defined by ECIES_CURVE.

See also:

NID_sect571k1

Returns:

NULL on failure, or a pointer to the newly allocated key pair.

Definition at line 63 of file ecies.c.

References EC_GROUP_set_point_conversion_form_d, EC_KEY_free_d, EC_KEY_get0_group_d, EC_KEY_new_by_curve_name_d, EC_KEY_new_d, EC_KEY_set_group_d, ECIES_CURVE, ecies_curve_group, ERR_error_string_d, ERR_get_error_d, and log_info.

Referenced by ecies_key_create(), ecies_key_private(), and ecies_key_public().

EC_KEY* ecies_key_create (void)

Generate a random ECIES key pair.

Returns:

NULL on failure, or a new random ECIES key pair on success.

Definition at line 117 of file ecies.c.

References EC_KEY_free_d, EC_KEY_generate_key_d, ecies_key_alloc(), ERR_error_string_d, ERR_get_error_d, and log_info.

Referenced by ecies_encrypt(), and meta_data_user_build_storage_keys().

void ecies_key_free (EC_KEY * key)

Free an ECIES key pair.

Returns:

This function returns no value.

Definition at line 52 of file ecies.c.

References EC_KEY_free_d.

Referenced by meta_data_user_build_storage_keys().

EC_KEY* ecies_key_private (uint64_t format, placer_t data)

Definition at line 206 of file ecies.c.

References BN_bin2bn_d, BN_free_d, BN_hex2bn_d, EC_KEY_free_d, EC_KEY_set_private_key_d, ecies_key_alloc(), ECIES_PRIVATE_BINARY, ECIES_PRIVATE_HEX, ERR_error_string_d, ERR_get_error_d, log_info, number, pl_char_get(), pl_data_get(), and pl_length_get().

Referenced by ecies_decrypt().

char* ecies_key_private_bin (EC_KEY * key, size_t * olen)

Return an ECIES private key as binary data.

Parameters:

key the input ECIES key pair.

olen a pointer to store the length of the returned key.

Returns:

NULL on failure, or a pointer to the raw private key.

Definition at line 377 of file ecies.c.

References BN_bn2bin_d, BN_num_bytes_d, EC_KEY_get0_private_key_d, ERR_error_string_d, ERR_get_error_d, log_info, mm_sec_alloc(), and mm_sec_free().

Referenced by meta_data_user_build_storage_keys().

stringer_t* ecies_key_private_hex (EC_KEY * key)

Return an ECIES private key as a hex string.

Parameters:

key the input ECIES key pair.

Returns:

NULL on failure, or the hex-formatted private key as a managed string.

Definition at line 336 of file ecies.c.

References BN_bn2hex_d, CONTIGUOUS, EC_KEY_get0_private_key_d, ERR_error_string_d, ERR_get_error_d, log_info, MANAGED_T, mm_wipe(), ns_length_get(), OPENSSL_free_d, SECURE, st_alloc_opts(), st_copy_in(), and st_free().

EC_KEY* ecies_key_public (uint64_t *format*, placer_t *data*)

Definition at line 136 of file ecies.c.

References EC_KEY_check_key_d, EC_KEY_free_d, EC_KEY_get0_group_d, EC_KEY_set_public_key_d, EC_POINT_free_d, EC_POINT_hex2point_d, EC_POINT_new_d, EC_POINT_oct2point_d, ecies_key_alloc(), ECIES_PUBLIC_BINARY, ECIES_PUBLIC_HEX, ERR_error_string_d, ERR_get_error_d, log_info, pl_char_get(), pl_data_get(), and pl_length_get().

Referenced by ecies_decrypt(), and ecies_encrypt().

unsigned char* ecies_key_public_bin (EC_KEY * *key*, size_t * *olen*)

Return an ECIES public key as binary data.

Parameters:

key the input ECIES key pair.

olen a pointer to store the length of the returned key.

Returns:

NULL on failure, or a pointer to the raw public key.

Definition at line 298 of file ecies.c.

References EC_KEY_get0_group_d, EC_KEY_get0_public_key_d, EC_POINT_point2oct_d, ERR_error_string_d, ERR_get_error_d, log_info, mm_alloc(), and mm_free().

Referenced by meta_data_user_build_storage_keys().

stringer_t* ecies_key_public_hex (EC_KEY * *key*)

Return an ECIES public key as a null-terminated hex string.

Parameters:

key the input ECIES key pair.

Returns:

NULL on failure, or the hex-formatted public key as a null-terminated string.

Definition at line 262 of file ecies.c.

References EC_KEY_get0_group_d, EC_KEY_get0_public_key_d, EC_POINT_point2hex_d, ERR_error_string_d, ERR_get_error_d, log_info, ns_length_get(), OPENSSL_free_d, and st_import().

bool_t ecies_start (void)

Definition at line 20 of file ecies.c.

References ECIES_CIPHER, ecies_cipher_evp, ECIES_CURVE, ecies_curve_group, ECIES_ENVELOPE, ecies_envelope_evp, ecies_group(), ECIES_HMAC, ecies_hmac_evp, EVP_get_cipherbyname_d, EVP_get_digestbyname_d, log_error, and OBJ_nid2sn_d.

Referenced by process_start().

void ecies_stop (void)

Destroy all initialized ECIES curve group information.

Returns:

This function returns no value.

Definition at line 35 of file ecies.c.

References EC_GROUP_free_d, and ecies_curve_group.

Referenced by process_stop().

bool_t lib_load_openssl (void)

openssl.c

openssl.c

Returns:

true on success or false on failure.

Definition at line 31 of file openssl.c.

References lib_symbols(), M_BIND, ns_length_get(), pl_char_get(), pl_length_int(), placer_t, ssl_version, and tok_get_ns().

Referenced by lib_load().

const char* lib_version_openssl (void)

Return the version string of the openssl library.

Returns:

a pointer to a character string containing the libopenssl version information.

Definition at line 22 of file openssl.c.

References ssl_version.

Referenced by lib_load().

stringer_t* rand_choices (chr_t * choices, size_t len)

Get a random string of data of a specified size, populated with characters from a chosen set.

Parameters:

choices a pointer to a null-terminated string containing a pool of characters from which the contents of the random data will be selected.

len the length, in bytes, of the random string that will be returned to the caller.

Returns:

NULL on failure or a pointer to a managed string containing *len* bytes of random data on success.

Definition at line 28 of file random.c.

References `log_pedantic`, `ns_length_get()`, `RAND_bytes_d`, `st_alloc`, `st_data_get()`, `st_free()`, and `st_length_set()`.

Referenced by `mail_create_message()`, `register_print_captcha()`, `register_session_generate()`, `smtp_bounce()`, and `smtp_reply()`.

int16_t rand_get_int16 (void)

Definition at line 221 of file random.c.

References `buflen`, `bufptr`, `log_pedantic`, `RAND_bytes_d`, `rand_ctx`, and `ssl_error_string()`.

int32_t rand_get_int32 (void)

Definition at line 203 of file random.c.

References `buflen`, `bufptr`, `log_pedantic`, `RAND_bytes_d`, `rand_ctx`, and `ssl_error_string()`.

int64_t rand_get_int64 (void)

Generate a random signed 64 bit number.

Note:

This function attempts to generate random data securely, but falls back on the pseudo-random number generator.

Returns:

the newly generated signed 64 bit integer.

See also:

`RAND_bytes()`

Definition at line 183 of file random.c.

References `buflen`, `bufptr`, `log_pedantic`, `RAND_bytes_d`, `rand_ctx`, and `ssl_error_string()`.

Referenced by `portal_message_source()`.

int8_t rand_get_int8 (void)

Definition at line 240 of file random.c.

References `buflen`, `bufptr`, `log_pedantic`, `RAND_bytes_d`, `rand_ctx`, and `ssl_error_string()`.

uint16_t rand_get_uint16 (void)

Generate a random unsigned 16 bit number.

Note:

This function attempts to generate random data securely, but falls back on the pseudo-random number generator.

Returns:

the newly generated unsigned 16 bit integer.

See also:

RAND_bytes()

Definition at line 144 of file random.c.

References buflen, bufptr, log_pedantic, RAND_bytes_d, rand_ctx, and ssl_error_string().

uint32_t rand_get_uint32 (void)

Generate a random unsigned 32 bit number.

Note:

This function attempts to generate random data securely, but falls back on the pseudo-random number generator.

Returns:

the newly generated unsigned 32 bit integer.

See also:

RAND_bytes()

Definition at line 126 of file random.c.

References buflen, bufptr, log_pedantic, RAND_bytes_d, rand_ctx, and ssl_error_string().

Referenced by process_maint(), register_captcha_generate(), register_captcha_random_font(), register_captcha_write_noise(), and smtp_client_connect().

uint64_t rand_get_uint64 (void)

Generate a random unsigned 64 bit number.

Note:

This function attempts to generate random data securely, but falls back on the pseudo-random number generator.

Returns:

the newly generated unsigned 64 bit integer.

See also:

RAND_bytes()

Definition at line 105 of file random.c.

References buflen, bufptr, log_pedantic, RAND_bytes_d, rand_ctx, and ssl_error_string().

Referenced by mail_mime_generate_boundary(), sess_key(), smtp_store_spamsig(), and spool_mktemp().

uint8_t rand_get_uint8 (void)

Generate a random unsigned 8 bit number.

Note:

This function attempts to generate random data securely, but falls back on the pseudo-random number generator.

Returns:

the newly generated unsigned 8 bit integer.

See also:

RAND_bytes()

Definition at line 163 of file random.c.

References buflen, bufptr, log_pedantic, RAND_bytes_d, rand_ctx, and ssl_error_string().

bool_t rand_start (void)

random.c

random.c

Note:

The default seed source for cryptographically secure generation routines is the system device /dev/random.

Returns:

false on failure, true on success.

Definition at line 276 of file random.c.

References magma_t::cryptography, magma_t::iface, magma, rand_ctx, RAND_load_file_d, RAND_status_d, and thread_get_thread_id().

Referenced by process_start().

void rand_stop (void)

Definition at line 295 of file random.c.

References RAND_cleanup_d.

Referenced by process_stop().

bool_t rand_thread_start (void)

Definition at line 260 of file random.c.

References magma_t::cryptography, magma_t::iface, magma, rand_ctx, and thread_get_thread_id().

size_t rand_write (stringer_t * s)

Fill a managed string with random data.

Note:

This function generates random data using the cryptographically strong Openssl function RAND_bytes().

Parameters:

s the input managed string.

Returns:

0 on failure, or the total number of bytes written to the managed string.

See also:

RAND_bytes()

Definition at line 68 of file random.c.

References log_pedantic, RAND_bytes_d, st_avail_get(), st_data_get(), st_length_get(), st_length_set(), st_valid_avail(), st_valid_destination(), and st_valid_tracked().

Referenced by symmetric_vector().

scramble_t* scramble_alloc (size_t *length*)

scramble.c

scramble.c

Parameters:

length the length, in bytes, of the scrambled data buffer (should include the IV and encrypted body length).

Returns:

a pointer to a newly allocated scrambled data header.

Definition at line 144 of file scramble.c.

References mm_alloc().

Referenced by scramble_encrypt().

void* scramble_body_data (scramble_t * *buffer*)

Get a pointer to the start of the encrypted data from a scrambled data header.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

a pointer to the start of the scrambled data header's associated encrypted data body.

Definition at line 84 of file scramble.c.

References scramble_vector_length().

Referenced by scramble_decrypt(), scramble_encrypt(), and scramble_import().

uint64_t scramble_body_hash (scramble_t * *buffer*)

Get a hash of a scrambled object's (encrypted) body data.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

the 64-bit hash of the specified scrambled object's body data.
Definition at line 36 of file scramble.c.

uint64_t scramble_body_length (scramble_t * *buffer*)

Get the length of a scrambled object's (encrypted) body data.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

the length, in bytes, of the scrambled object's body data.
Definition at line 60 of file scramble.c.
Referenced by scramble_decrypt(), and scramble_total_length().

stringer_t* scramble_decrypt (stringer_t * *key*, scramble_t * *input*)

Un-scramble a block of data using an decryption key.

Parameters:

key a managed string containing the symmetric decryption key.
input a pointer to the scrambled data header to be decrypted.

Returns:

NULL on failure, or a managed string containing the verified decrypted data.
Definition at line 225 of file scramble.c.

References cipher_id(), hash_adler32(), log_info, log_pedantic, MANAGEDBUF, PLACER, scramble_body_data(), scramble_body_length(), scramble_vector_data(), scramble_vector_length(), st_free(), st_length_get(), symmetric_decrypt(), and symmetric_key().

Referenced by meta_data_user_build_storage_keys(), and sess_get().

scramble_t* scramble_encrypt (stringer_t * *key*, stringer_t * *input*)

Scramble a block of data using an encryption key.

Note:

This function is configured to use AES 256 in CBC mode.

Parameters:

key a managed string containing the symmetric encryption key.
input a managed string containing the data to be encrypted.

Returns:

NULL on failure, or a pointer to a scrambled data header containing the encrypted data and its metadata.
Definition at line 168 of file scramble.c.

References `cipher_id()`, `cipher_numeric_id()`, `hash_adler32()`, `log_pedantic`, `MANAGEDBUF`, `mm_copy()`, `scramble_alloc()`, `scramble_body_data()`, `scramble_vector_data()`, `st_data_get()`, `st_free()`, `st_length_get()`, `symmetric_encrypt()`, `symmetric_key()`, and `symmetric_vector()`.

Referenced by `meta_data_user_save_storage_keys()`, and `sess_token()`.

void scramble_free (scramble_t * *buffer*)

Free a scrambled data block.

Parameters:

buffer a pointer to the scrambled data header to be freed.

Returns:

This function returns no value.

Definition at line 154 of file `scramble.c`.

References `mm_free()`.

Referenced by `meta_data_user_save_storage_keys()`, and `sess_token()`.

scramble_t* scramble_import (stringer_t * *s*)

Return a managed string as a scrambled buffer, after validation.

Parameters:

s a managed string containing the serialized scrambled data.

Returns:

NULL on failure, or a pointer to the scrambled data header on success.

Definition at line 104 of file `scramble.c`.

References `hash_adler32()`, `log_pedantic`, `scramble_body_data()`, `st_data_get()`, `st_empty()`, and `st_length_get()`.

Referenced by `sess_get()`.

uint64_t scramble_orig_length (scramble_t * *buffer*)

Get the length of the original (unencrypted) data underlying a scrambled object.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

the length, in bytes, of the scrambled object's original data.

Definition at line 48 of file `scramble.c`.

uint64_t scramble_total_length (scramble_t * *buffer*)

TODO: Create envelope functions to translate the scrambled buffer into a condensed/encoded stringer.

Get the total length of a scrambled object.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

the total length, in bytes, of the scrambled object.

Definition at line 22 of file `scramble.c`.

References `scramble_body_length()`, and `scramble_vector_length()`.

Referenced by `meta_data_user_save_storage_keys()`, and `sess_token()`.

void* scramble_vector_data (scramble_t * *buffer*)

Get a pointer to the start of the IV from a scrambled data header.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

a pointer to the start of the scrambled data header's associated IV block.

Definition at line 94 of file `scramble.c`.

Referenced by `scramble_decrypt()`, and `scramble_encrypt()`.

uint64_t scramble_vector_length (scramble_t * *buffer*)

Get the length of a scrambled object's IV.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

the length, in bytes, of the scrambled object's IV.

Definition at line 72 of file `scramble.c`.

Referenced by `scramble_body_data()`, `scramble_decrypt()`, and `scramble_total_length()`.

SSL* ssl_alloc (void * *server*, int *sockd*, int *flags*)

Create an SSL session for a file descriptor, and accept the client TLS/SSL handshake.

See also:

`SSL_accept()`

`BIO_new_socket()`

Parameters:

server a server object which contains the underlying SSL context.

sockd the file descriptor of the TCP connection to be made SSL-ready.

flags passed to `BIO_new_socket()`, determines whether the socket is shut down when the BIO is freed.

Definition at line 282 of file `openssl.c`.

References `BIO_new_socket_d`, `buflen`, `bufptr`, `server_t::context`, `log_pedantic`, `server_t::ssl`, `SSL_accept_d`, `ssl_error_string()`, `SSL_free_d`, `SSL_new_d`, and `SSL_set_bio_d`.

Referenced by `imap_starttls()`, `pop_starttls()`, `protocol_secure()`, and `smtp_starttls()`.

void* ssl_client_create (int_t sockd)

Establish an SSL client wrapper around a socket descriptor.

Parameters:

sockd the file descriptor of the socket to have its transport security level upgraded.

Returns:

NULL on failure or a pointer to the SSL handle of the file descriptor if SSL negotiation was successful.

LOW: Add requisite config options and sandbox resources to verify server SSL certificates.

Definition at line 348 of file `openssl.c`.

References `BIO_new_socket_d`, `log_pedantic`, `MEMORYBUF`, `SSL_connect_d`, `SSL_CTX_ctrl_d`, `SSL_CTX_free_d`, `SSL_CTX_new_d`, `ssl_error_string()`, `SSL_free_d`, `SSL_new_d`, `SSL_set_bio_d`, and `SSLv23_client_method_d`.

Referenced by `client_secure()`.

DH* ssl_dh_exchange_callback (SSL * ssl, int is_export, int keylength)

Callback handler for the Diffie-Hellman parameter generation process necessary for PFS.

Parameters:

ssl the SSL session for which the callback was triggered.

is_export LOL. We're definitely ignoring this parameter.

keylength the length, in bits, of the DH key to be generated.

Returns:

a pointer to a Diffie-Hellman key of proper size with parameters generated, or NULL on failure.

Definition at line 613 of file `openssl.c`.

References `DH_free_d`, `DH_generate_parameters_ex_d`, `DH_new_d`, `log_error`, `log_pedantic`, and `ssl_dh_generate_callback()`.

Referenced by `ssl_server_create()`.

int ssl_dh_generate_callback (int p, int n, BN_GENCB * cb)

The DH param generation callback.

Definition at line 699 of file `openssl.c`.

Referenced by `ssl_dh_exchange_callback()`.

EC_KEY* ssl_ecdh_exchange_callback (SSL * ssl, int is_export, int keylength)

Callback handler for the EC Diffie-Hellman parameter generation process necessary for PFS.

Parameters:

ssl the SSL session for which the callback was triggered.
is_export LOL. We're definitely ignoring this parameter.
keylength the length, in bits, of the ECCH key to be generated.

Returns:

a pointer to a ECDH key of proper size with parameters generated, or NULL on failure.
 Definition at line 663 of file openssl.c.

References EC_GROUP_set_point_conversion_form_d, EC_KEY_get0_group_d,
 EC_KEY_new_by_curve_name_d, and log_error.

Referenced by ssl_server_create().

char* ssl_error_string (chr_t * *buffer*, int_t *length*)

Get a textual representation of the last openssl error message.

Parameters:

buffer a buffer that will receive the last openssl error message.
length the size, in bytes, of the buffer that will contain the last openssl error message.

Returns:

NULL on failure, or a pointer to the buffer where the last openssl error message has been stored.
 Definition at line 87 of file openssl.c.

References ERR_error_string_n_d, ERR_get_error_d, and log_pedantic.

Referenced by rand_get_int16(), rand_get_int32(), rand_get_int64(), rand_get_int8(), rand_get_uint16(),
 rand_get_uint32(), rand_get_uint64(), rand_get_uint8(), ssl_alloc(), ssl_client_create(), ssl_verify_privkey(),
 and ssl_write().

void ssl_free (SSL * *ssl*)

Shutdown and free an SSL connection.

Parameters:

ssl the SSL connection to be shut down.

Returns:

This function returns no value.
 Definition at line 235 of file openssl.c.
 References ERR_remove_state_d, log_pedantic, SSL_free_d, and SSL_shutdown_d.
 Referenced by client_close(), con_destroy(), and protocol_secure().

void ssl_locking_callback (int *mode*, int *n*, const char * *file*, int *line*)

The locking function callback necessary for all multi-threaded applications using openssl.

See also:

CRYPTO_set_locking_callback()

Parameters:

mode a bitmask specifying the requested openssl locking operation (CRYPTO_LOCK, CRYPTO_WRITE, CRYPTO_READ, or CRYPTO_UNLOCK).

n the zero-based index of the lock that is the target of the requested operation.

file a null-terminated string containing the filename of the function setting the lock, for debugging purposes.

line the line number of the function setting the lock, for debugging purposes.

Returns:

This function returns no value.

Definition at line 491 of file openssl.c.

References mutex_lock(), mutex_unlock(), and ssl_locks.

Referenced by ssl_start().

int ssl_print (SSL * *ssl*, const char * *format*, va_list *args*)

Write formatted data to an SSL connection.

Parameters:

ssl the SSL connection to which the data will be written.

format a format string specifying the data to be written to the SSL connection.

va_list a variable argument list containing the data parameters associated with the format string.

Returns:

-1 on error, or the number of bytes written to the SSL connection.

Definition at line 175 of file openssl.c.

References buflen, bufptr, length, mm_alloc(), mm_free(), and ssl_write().

int ssl_read (SSL * *ssl*, void * *buffer*, int *length*, bool_t *block*)

Read data from an SSL connection.

Parameters:

ssl the SSL connection from which the data will be read.

buffer a pointer to the buffer where the read data will be stored.

length the length, in bytes, of the amount of data to be read.

block a boolean variable specifying whether the read operation should block.

Returns:

-1 on failure, 0 if the connection has been closed, or the number of bytes read from the connection on success.

Definition at line 110 of file openssl.c.

References buflen, bufptr, ERR_error_string_n_d, log_pedantic, SSL_get_error_d, SSL_peek_d, and SSL_read_d.

Referenced by client_read(), client_read_line(), con_read(), and con_read_line().

bool_t ssl_server_create (void * *server*)

Definition at line 398 of file openssl.c.

References `server_t::certificate`, `server_t::context`, `log_critical`, `MAGMA_CIPHER_LIST`, `server_t::ssl`, `SSL_CTX_check_private_key_d`, `SSL_CTX_ctrl_d`, `SSL_CTX_new_d`, `SSL_CTX_set_cipher_list_d`, `SSL_CTX_set_tmp_dh_callback_d`, `SSL_CTX_set_tmp_ecdh_callback_d`, `SSL_CTX_use_certificate_chain_file_d`, `SSL_CTX_use_PrivateKey_file_d`, `ssl_dh_exchange_callback()`, `ssl_ecdh_exchange_callback()`, and `SSLv23_server_method_d`.

Referenced by `servers_encryption_start()`.

void ssl_server_destroy (void * server)

Destroy an SSL context associated with a server.

Parameters:

server the server to be deactivated.

Returns:

This function returns no value.

Definition at line 328 of file `openssl.c`.

References `server_t::context`, `log_pedantic`, `server_t::ssl`, and `SSL_CTX_free_d`.

Referenced by `servers_encryption_stop()`.

int ssl_shutdown_get (SSL * ssl)

Checks whether an SSL tunnel has been shut down or not.

See also:

`SSL_get_shutdown()`

Parameters:

ssl the SSL connection to be shut down.

Returns:

0 if the connection is alive and well, or `SSL_SENT_SHUTDOWN/SSL_RECEIVED_SHUTDOWN`

Definition at line 265 of file `openssl.c`.

References `SSL_get_shutdown_d`.

Referenced by `con_read()`.

bool_t ssl_start (void)

Initialize the openssl facility.

Note:

First, this function initializes the mutexes necessary for the locking function callback that openssl uses for shared data structures in multi-threaded applications. Next, the DKIM key is retrieved if the **magma.dkim.enabled** configuration variable is set.

Returns:

true if openssl was initialized successfully, or false on failure.

Definition at line 525 of file `openssl.c`.

References CRYPTO_num_locks_d, CRYPTO_set_id_callback_d, CRYPTO_set_locking_callback_d, magma_t::dkim, magma_t::enabled, file_load(), file_world_accessible(), log_critical, magma, mm_alloc(), mutex_init(), OPENSSL_add_all_algorithms_noconf_d, magma_t::privkey, SSL_library_init_d, SSL_load_error_strings_d, ssl_locking_callback(), ssl_locks, ssl_thread_id_callback(), ssl_verify_privkey(), st_char_get(), and st_free().

Referenced by process_start().

void ssl_stop (void)

Stop the openssl library and cleanup all associated data structures.

Returns:

This function returns no values.

Definition at line 448 of file openssl.c.

References ASN1_STRING_TABLE_cleanup_d, BIO_sock_cleanup_d, COMP_zlib_cleanup_d, CONF_modules_unload_d, CRYPTO_cleanup_all_ex_data_d, CRYPTO_num_locks_d, CRYPTO_set_id_callback_d, CRYPTO_set_locking_callback_d, ENGINE_cleanup_d, ERR_free_strings_d, ERR_remove_state_d, EVP_cleanup_d, mm_free(), mutex_destroy(), OBJ_cleanup_d, OBJ_NAME_cleanup_d, sk_pop_free_d, and ssl_locks.

Referenced by process_stop().

void ssl_thread_stop (void)

Free the calling thread's SSL error queue.

See also:

ssl_thread_stop()

Returns:

This function returns no value.

Definition at line 223 of file openssl.c.

References ERR_remove_state_d.

Referenced by thread_stop().

bool_t ssl_verify_privkey (const char * *keyfile*)

Verify that the filename contains a valid private key in PEM format.

Parameters:

keyfile the pathname of the private key.

Returns:

true if the the key is valid, or false if it is not.

Definition at line 586 of file openssl.c.

References log_pedantic, MEMORYBUF, SSL_CTX_free_d, SSL_CTX_new_d, SSL_CTX_use_PrivateKey_file_d, ssl_error_string(), and SSLv23_client_method_d.

Referenced by ssl_start().

int ssl_write (SSL * *ssl*, const void * *buffer*, int *length*)

Write data to an open SSL connection.

Parameters:

ssl the SSL connection to which the data will be written.
buffer a pointer to the buffer containing the data to be written.
length the length, in bytes, of the data to be written.

Returns:

-1 on error, or the number of bytes written to the SSL connection.

Definition at line 145 of file openssl.c.

References *buflen*, *bufptr*, *log_pedantic*, *ssl_error_string()*, *SSL_get_error_d*, and *SSL_write_d*.

Referenced by *client_write()*, *con_write_bl()*, and *ssl_print()*.

stringer_t* symmetric_decrypt (cipher_t * *cipher*, stringer_t * *vector*, stringer_t * *key*, stringer_t * *input*)

symmetric.c

Definition at line 108 of file symmetric.c.

References *ERR_error_string_d*, *ERR_get_error_d*, *EVP_CIPHER_CTX_block_size_d*, *EVP_CIPHER_CTX_cleanup_d*, *EVP_CIPHER_CTX_init_d*, *EVP_CIPHER_CTX_iv_length_d*, *EVP_CIPHER_CTX_key_length_d*, *EVP_DecryptFinal_ex_d*, *EVP_DecryptInit_ex_d*, *EVP_DecryptUpdate_d*, *log_pedantic*, *st_alloc*, *st_data_get()*, *st_empty_out()*, *st_free()*, and *st_length_set()*.

Referenced by *scramble_decrypt()*.

stringer_t* symmetric_encrypt (cipher_t * *cipher*, stringer_t * *vector*, stringer_t * *key*, stringer_t * *input*)

Encrypt a block of data using a symmetric cipher.

Parameters:

cipher the cipher type being used to encrypt the data.
vector the IV used for the encryption operation.
key the symmetric encryption key used to encrypt the data.
input a managed string containing the data to be encrypted.

Returns:

NULL on failure, or a pointer to a managed string containing the encrypted data.

Definition at line 24 of file symmetric.c.

References *EVP_CIPHER_CTX_block_size_d*, *EVP_CIPHER_CTX_cleanup_d*, *EVP_CIPHER_CTX_init_d*, *EVP_CIPHER_CTX_iv_length_d*, *EVP_CIPHER_CTX_key_length_d*, *EVP_EncryptFinal_ex_d*, *EVP_EncryptInit_ex_d*, *EVP_EncryptUpdate_d*, *log_pedantic*, *st_alloc*, *st_data_get()*, *st_empty_out()*, *st_free()*, and *st_length_set()*.

Referenced by *scramble_encrypt()*.

stringer_t* symmetric_key (cipher_t * *cipher*, stringer_t * *key*, stringer_t * *output*)

Derive a symmetric key from a user's password.

Note:

Depending on the length of the password, either SHA1, SHA224, or SHA256 may be used as the hashing algorithm.

If output is passed as NULL, a new managed string will be allocated to hold the output of the operation.

Parameters:

cipher the specified cipher type.

key the user's password, as a managed string.

output the output managed string to receive the digested password. Can be NULL.

Returns:

NULL on failure, or a pointer to the managed string containing the digested password.

Definition at line 234 of file symmetric.c.

References `cipher_key_length()`, `digest_sha1()`, `digest_sha224()`, `digest_sha256()`, `st_length_get()`, `st_length_set()`, `st_output()`, and `st_valid_tracked()`.

Referenced by `scramble_decrypt()`, and `scramble_encrypt()`.

stringer_t* symmetric_vector (cipher_t * *cipher*, stringer_t * *output*)

Generate an IV data suitable for the specified cipher.

Note:

If output is NULL, a new managed string will be allocated and returned by the function.

Parameters:

cipher the cipher type to be used.

output the managed string that will store the IV data.

Returns:

NULL on failure, or a pointer to the managed string that contains the IV data.

Definition at line 199 of file symmetric.c.

References `cipher_vector_length()`, `log_pedantic`, `PLACER`, `rand_write()`, `st_data_get()`, `st_length_set()`, `st_output()`, and `st_valid_tracked()`.

Referenced by `scramble_encrypt()`.

magma/providers/cryptography/digest.c File Reference

Functions used to create a secure one-way hash of an arbitrary input buffer.

```
#include "magma.h"
```

Functions

- **digest_t * digest_name** (stringer_t *name)
- **digest_t * digest_id** (int_t id)
- *digest.c* **stringer_t * digest_hash** (digest_t *digest, stringer_t *s, stringer_t *output)
- **stringer_t * digest_md4** (stringer_t *s, stringer_t *output)
- **stringer_t * digest_md5** (stringer_t *s, stringer_t *output)
- **stringer_t * digest_sha** (stringer_t *s, stringer_t *output)
- **stringer_t * digest_sha1** (stringer_t *s, stringer_t *output)
- **stringer_t * digest_sha224** (stringer_t *s, stringer_t *output)
- **stringer_t * digest_sha256** (stringer_t *s, stringer_t *output)
- **stringer_t * digest_sha384** (stringer_t *s, stringer_t *output)
- **stringer_t * digest_sha512** (stringer_t *s, stringer_t *output)
- **stringer_t * digest_ripemd160** (stringer_t *s, stringer_t *output)

Detailed Description

Functions used to create a secure one-way hash of an arbitrary input buffer.

Definition in file **digest.c**.

Function Documentation

stringer_t* digest_hash (digest_t * *digest*, stringer_t * *s*, stringer_t * *output*)

Definition at line 36 of file *digest.c*.

References *EVP_DigestFinal_d*, *EVP_DigestInit_ex_d*, *EVP_DigestUpdate_d*, *EVP_MD_CTX_cleanup_d*, *EVP_MD_CTX_init_d*, *EVP_MD_size_d*, *log_pedantic*, *st_alloc*, *st_avail_get()*, *st_data_get()*, *st_empty()*, *st_free()*, *st_length_get()*, *st_length_set()*, *st_valid_avail()*, *st_valid_destination()*, and *st_valid_tracked()*.

Referenced by *digest_md4()*, *digest_md5()*, *digest_ripemd160()*, *digest_sha()*, *digest_sha1()*, *digest_sha224()*, *digest_sha256()*, *digest_sha384()*, and *digest_sha512()*.

digest_t* digest_id (int_t *id*)

digest.c

Definition at line 25 of file *digest.c*.

References *EVP_get_digestbyname_d*, *log_pedantic*, and *OBJ_nid2sn_d*.

stringer_t* digest_md4 (stringer_t * *s*, stringer_t * *output*)

Definition at line 112 of file digest.c.

References digest_hash(), and EVP_md4_d.

stringer_t* digest_md5 (stringer_t * s, stringer_t * output)

Definition at line 116 of file digest.c.

References digest_hash(), and EVP_md5_d.

digest_t* digest_name (stringer_t * name)

Definition at line 15 of file digest.c.

References EVP_get_digestbyname_d, log_pedantic, st_char_get(), st_empty(), and st_length_int().

stringer_t* digest_ripemd160 (stringer_t * s, stringer_t * output)

Definition at line 144 of file digest.c.

References digest_hash(), and EVP_ripemd160_d.

stringer_t* digest_sha (stringer_t * s, stringer_t * output)

Definition at line 120 of file digest.c.

References digest_hash(), and EVP_sha_d.

stringer_t* digest_sha1 (stringer_t * s, stringer_t * output)

Definition at line 124 of file digest.c.

References digest_hash(), and EVP_sha1_d.

Referenced by symmetric_key().

stringer_t* digest_sha224 (stringer_t * s, stringer_t * output)

Definition at line 128 of file digest.c.

References digest_hash(), and EVP_sha224_d.

Referenced by symmetric_key().

stringer_t* digest_sha256 (stringer_t * s, stringer_t * output)

Definition at line 132 of file digest.c.

References digest_hash(), and EVP_sha256_d.

Referenced by symmetric_key().

stringer_t* digest_sha384 (stringer_t * s, stringer_t * output)

Definition at line 136 of file digest.c.

References `digest_hash()`, and `EVP_sha384_d`.

`stringer_t* digest_sha512 (stringer_t * s, stringer_t * output)`

Definition at line 140 of file digest.c.

References `digest_hash()`, and `EVP_sha512_d`.

Referenced by `credential_alloc_auth()`.

magma/providers/cryptography/ecies.c File Reference

ECIES encryption/decryption functions.

```
#include "magma.h"
```

Functions

- **bool_t ecies_start** (void)
- void **ecies_stop** (void)
- *Destroy all initialized ECIES curve group information.* void **ecies_key_free** (EC_KEY *key)
- *Free an ECIES key pair.* EC_KEY * **ecies_key_alloc** (void)
- *Allocate a new ECIES key pair from the curve defined by ECIES_CURVE.* EC_GROUP * **ecies_group** (uint64_t curve, **bool_t** precompute)
- *ecies.c* EC_KEY * **ecies_key_create** (void)
- *Generate a random ECIES key pair.* EC_KEY * **ecies_key_public** (uint64_t format, **placer_t** data)
- EC_KEY * **ecies_key_private** (uint64_t format, **placer_t** data)
- **stringer_t** * **ecies_key_public_hex** (EC_KEY *key)
- *Return an ECIES public key as a null-terminated hex string.* unsigned char * **ecies_key_public_bin** (EC_KEY *key, size_t *olen)
- *Return an ECIES public key as binary data.* **stringer_t** * **ecies_key_private_hex** (EC_KEY *key)
- *Return an ECIES private key as a hex string.* char * **ecies_key_private_bin** (EC_KEY *key, size_t *olen)
- *Return an ECIES private key as binary data.* void * **ecies_envelope_derivation** (const void *input, size_t ilen, void *output, size_t *olen)
- **cryptex_t** * **ecies_encrypt** (**stringer_t** *key, ECIES_KEY_TYPE key_type, unsigned char *data, size_t length)
- *Encrypt a block of data using an ECIES public key.* unsigned char * **ecies_decrypt** (**stringer_t** *key, ECIES_KEY_TYPE key_type, **cryptex_t** *cryptex, size_t *length)

Decrypt a block of data using an ECIES private key. Variables

- EC_GROUP * **ecies_curve_group** = NULL
- const EVP_MD * **ecies_hmac_evp** = NULL
- const EVP_MD * **ecies_envelope_evp** = NULL
- const EVP_CIPHER * **ecies_cipher_evp** = NULL

Detailed Description

ECIES encryption/decryption functions.

Definition in file **ecies.c**.

Function Documentation

unsigned char* **ecies_decrypt** (**stringer_t** * key, ECIES_KEY_TYPE key_type, **cryptex_t** * cryptex, size_t * length)

Decrypt a block of data using an ECIES private key.

Parameters:

key the ECIES private key in the specified format.

key_type the encoding type of the ECIES private key (ECIES_PRIVATE_BINARY or ECIES_PRIVATE_HEX).

cryptex a pointer to the head of the cryptex object with the encrypted data.

length a pointer to a size_t variable which will receive the final length of the unencrypted data.

Returns:

NULL on failure, or a pointer to a memory address containing the decrypted data on success..

Definition at line 595 of file ecies.c.

References cryptex_body_data(), cryptex_body_length(), cryptex_envelope_data(), cryptex_envelope_length(), cryptex_hmac_length(), cryptex_mac_data(), cryptex_original_length(), EC_KEY_free_d, EC_KEY_get0_public_key_d, ECDH_compute_key_d, ECIES_CIPHER, ecies_envelope_derivation(), ECIES_HMAC, ecies_key_private(), ecies_key_public(), ECIES_PRIVATE_BINARY, ECIES_PRIVATE_HEX, ECIES_PUBLIC_BINARY, ERR_error_string_d, ERR_get_error_d, EVP_CIPHER_CTX_cleanup_d, EVP_CIPHER_CTX_init_d, EVP_CIPHER_CTX_set_padding_d, EVP_CIPHER_key_length_d, EVP_DecryptFinal_ex_d, EVP_DecryptInit_ex_d, EVP_DecryptUpdate_d, EVP_get_cipherbyname_d, EVP_get_digestbyname_d, HMAC_CTX_cleanup_d, HMAC_CTX_init_d, HMAC_Final_d, HMAC_Init_ex_d, HMAC_Update_d, log_info, mm_alloc(), OBJ_nid2sn_d, pl_init(), and st_empty_out().

Referenced by adjust_message_encryption(), mail_load_header(), and mail_load_message().

cryptex_t* ecies_encrypt (stringer_t * key, ECIES_KEY_TYPE key_type, unsigned char * data, size_t length)

Encrypt a block of data using an ECIES public key.

Parameters:

key the ECIES public key in the specified format.

key_type the encoding type of the ECIES public key (ECIES_PUBLIC_BINARY or ECIES_PUBLIC_HEX).

data a pointer to the block of data to be encrypted.

length the length, in bytes, of the data to be encrypted.

Returns:

NULL on failure, or a pointer to the header of the cryptex object containing the encrypted data on success..

Definition at line 424 of file ecies.c.

References cryptex_alloc(), cryptex_body_data(), cryptex_body_length(), cryptex_envelope_data(), cryptex_free(), cryptex_hmac_length(), cryptex_mac_data(), EC_KEY_free_d, EC_KEY_get0_group_d, EC_KEY_get0_public_key_d, EC_POINT_point2oct_d, ECDH_compute_key_d, ECIES_CIPHER, ecies_envelope_derivation(), ECIES_HMAC, ecies_key_create(), ecies_key_public(), ECIES_PUBLIC_BINARY, ECIES_PUBLIC_HEX, ERR_error_string_d, ERR_get_error_d, EVP_CIPHER_block_size_d, EVP_CIPHER_CTX_cleanup_d, EVP_CIPHER_CTX_init_d, EVP_CIPHER_CTX_set_padding_d, EVP_CIPHER_key_length_d, EVP_EncryptFinal_ex_d, EVP_EncryptInit_ex_d, EVP_EncryptUpdate_d, EVP_get_cipherbyname_d, EVP_get_digestbyname_d, EVP_MD_size_d, HMAC_CTX_cleanup_d, HMAC_CTX_init_d, HMAC_Final_d, HMAC_Init_ex_d, HMAC_Update_d, log_info, OBJ_nid2sn_d, pl_init(), and st_empty_out().

Referenced by adjust_message_encryption(), and mail_store_message().

void* ecies_envelope_derivation (const void * input, size_t ilen, void * output, size_t * olen)

Definition at line 407 of file ecies.c.

References ecies_envelope_evp, and EVP_Digest_d.

Referenced by ecies_decrypt(), and ecies_encrypt().

EC_GROUP* ecies_group (uint64_t *curve*, bool_t *precompute*)

ecies.c

Definition at line 94 of file ecies.c.

References EC_GROUP_free_d, EC_GROUP_new_by_curve_name_d, EC_GROUP_precompute_mult_d, EC_GROUP_set_point_conversion_form_d, ERR_error_string_d, ERR_get_error_d, and log_error.

Referenced by ecies_start().

EC_KEY* ecies_key_alloc (void)

Allocate a new ECIES key pair from the curve defined by ECIES_CURVE.

See also:

NID_sect571k1

Returns:

NULL on failure, or a pointer to the newly allocated key pair.

Definition at line 63 of file ecies.c.

References EC_GROUP_set_point_conversion_form_d, EC_KEY_free_d, EC_KEY_get0_group_d, EC_KEY_new_by_curve_name_d, EC_KEY_new_d, EC_KEY_set_group_d, ECIES_CURVE, ecies_curve_group, ERR_error_string_d, ERR_get_error_d, and log_info.

Referenced by ecies_key_create(), ecies_key_private(), and ecies_key_public().

EC_KEY* ecies_key_create (void)

Generate a random ECIES key pair.

Returns:

NULL on failure, or a new random ECIES key pair on success.

Definition at line 117 of file ecies.c.

References EC_KEY_free_d, EC_KEY_generate_key_d, ecies_key_alloc(), ERR_error_string_d, ERR_get_error_d, and log_info.

Referenced by ecies_encrypt(), and meta_data_user_build_storage_keys().

void ecies_key_free (EC_KEY * *key*)

Free an ECIES key pair.

Returns:

This function returns no value.

Definition at line 52 of file ecies.c.

References EC_KEY_free_d.

Referenced by meta_data_user_build_storage_keys().

EC_KEY* ecies_key_private (uint64_t *format*, *placert_t data*)

Definition at line 206 of file ecies.c.

References BN_bin2bn_d, BN_free_d, BN_hex2bn_d, EC_KEY_free_d, EC_KEY_set_private_key_d, ecies_key_alloc(), ECIES_PRIVATE_BINARY, ECIES_PRIVATE_HEX, ERR_error_string_d, ERR_get_error_d, log_info, number, pl_char_get(), pl_data_get(), and pl_length_get().

Referenced by ecies_decrypt().

char* ecies_key_private_bin (EC_KEY * *key*, size_t * *olen*)

Return an ECIES private key as binary data.

Parameters:

key the input ECIES key pair.

olen a pointer to store the length of the returned key.

Returns:

NULL on failure, or a pointer to the raw private key.

Definition at line 377 of file ecies.c.

References BN_bn2bin_d, BN_num_bytes_d, EC_KEY_get0_private_key_d, ERR_error_string_d, ERR_get_error_d, log_info, mm_sec_alloc(), and mm_sec_free().

Referenced by meta_data_user_build_storage_keys().

stringer_t* ecies_key_private_hex (EC_KEY * *key*)

Return an ECIES private key as a hex string.

Parameters:

key the input ECIES key pair.

Returns:

NULL on failure, or the hex-formatted private key as a managed string.

Definition at line 336 of file ecies.c.

References BN_bn2hex_d, CONTIGUOUS, EC_KEY_get0_private_key_d, ERR_error_string_d, ERR_get_error_d, log_info, MANAGED_T, mm_wipe(), ns_length_get(), OPENSSL_free_d, SECURE, st_alloc_opts(), st_copy_in(), and st_free().

EC_KEY* ecies_key_public (uint64_t *format*, *placert_t data*)

Definition at line 136 of file ecies.c.

References EC_KEY_check_key_d, EC_KEY_free_d, EC_KEY_get0_group_d, EC_KEY_set_public_key_d, EC_POINT_free_d, EC_POINT_hex2point_d, EC_POINT_new_d, EC_POINT_oct2point_d,

ecies_key_alloc(), ECIES_PUBLIC_BINARY, ECIES_PUBLIC_HEX, ERR_error_string_d, ERR_get_error_d, log_info, pl_char_get(), pl_data_get(), and pl_length_get().

Referenced by ecies_decrypt(), and ecies_encrypt().

unsigned char* ecies_key_public_bin (EC_KEY * key, size_t * olen)

Return an ECIES public key as binary data.

Parameters:

key the input ECIES key pair.

olen a pointer to store the length of the returned key.

Returns:

NULL on failure, or a pointer to the raw public key.

Definition at line 298 of file ecies.c.

References EC_KEY_get0_group_d, EC_KEY_get0_public_key_d, EC_POINT_point2oct_d, ERR_error_string_d, ERR_get_error_d, log_info, mm_alloc(), and mm_free().

Referenced by meta_data_user_build_storage_keys().

stringer_t* ecies_key_public_hex (EC_KEY * key)

Return an ECIES public key as a null-terminated hex string.

Parameters:

key the input ECIES key pair.

Returns:

NULL on failure, or the hex-formatted public key as a null-terminated string.

Definition at line 262 of file ecies.c.

References EC_KEY_get0_group_d, EC_KEY_get0_public_key_d, EC_POINT_point2hex_d, ERR_error_string_d, ERR_get_error_d, log_info, ns_length_get(), OPENSSL_free_d, and st_import().

bool_t ecies_start (void)

Definition at line 20 of file ecies.c.

References ECIES_CIPHER, ecies_cipher_evp, ECIES_CURVE, ecies_curve_group, ECIES_ENVELOPE, ecies_envelope_evp, ecies_group(), ECIES_HMAC, ecies_hmac_evp, EVP_get_cipherbyname_d, EVP_get_digestbyname_d, log_error, and OBJ_nid2sn_d.

Referenced by process_start().

void ecies_stop (void)

Destroy all initialized ECIES curve group information.

Returns:

This function returns no value.

Definition at line 35 of file ecies.c.

References EC_GROUP_free_d, and ecies_curve_group.

Referenced by process_stop().

Variable Documentation

const EVP_CIPHER* ecies_cipher_evp = NULL

Definition at line 18 of file ecies.c.

Referenced by ecies_start().

EC_GROUP* ecies_curve_group = NULL

Definition at line 15 of file ecies.c.

Referenced by ecies_key_alloc(), ecies_start(), and ecies_stop().

const EVP_MD* ecies_envelope_evp = NULL

Definition at line 17 of file ecies.c.

Referenced by ecies_envelope_derivation(), and ecies_start().

const EVP_MD* ecies_hmac_evp = NULL

Definition at line 16 of file ecies.c.

Referenced by ecies_start().

magma/providers/cryptography/openssl.c File Reference

The interface to OpenSSL routines.

```
#include "magma.h"
```

Functions

- `const char * lib_version_openssl (void)`
- *Return the version string of the openssl library.* `bool_t lib_load_openssl (void)`
- *Initialize the openssl library and bind dynamically to the exported functions that are required.* `char * ssl_error_string (chr_t *buffer, int_t length)`
- *Get a textual representation of the last openssl error message.* `int ssl_read (SSL *ssl, void *buffer, int length, bool_t block)`
- *Read data from an SSL connection.* `int ssl_write (SSL *ssl, const void *buffer, int length)`
- *Write data to an open SSL connection.* `int ssl_print (SSL *ssl, const char *format, va_list args)`
- *Write formatted data to an SSL connection.* `void ssl_thread_stop (void)`
- *Free the calling thread's SSL error queue.* `void ssl_free (SSL *ssl)`
- *Shutdown and free an SSL connection.* `int ssl_shutdown_get (SSL *ssl)`
- *Checks whether an SSL tunnel has been shut down or not.* `SSL * ssl_alloc (void *server, int sockd, int flags)`
- *Create an SSL session for a file descriptor, and accept the client TLS/SSL handshake.* `void ssl_server_destroy (void *server)`
- *Destroy an SSL context associated with a server.* `void * ssl_client_create (int_t sockd)`
- *Establish an SSL client wrapper around a socket descriptor.* `bool_t ssl_server_create (void *server)`
- `void ssl_stop (void)`
- *Stop the openssl library and cleanup all associated data structures.* `void ssl_locking_callback (int mode, int n, const char *file, int line)`
- *The locking function callback necessary for all multi-threaded applications using openssl.* `unsigned long ssl_thread_id_callback (void)`
- *The thread id callback necessary for all multi-threaded applications using openssl.* `bool_t ssl_start (void)`
- *Initialize the openssl facility.* `bool_t ssl_verify_privkey (const char *keyfile)`
- *Verify that the filename contains a valid private key in PEM format.* `DH * ssl_dh_exchange_callback (SSL *ssl, int is_export, int keylength)`
- *Callback handler for the Diffie-Hellman parameter generation process necessary for PFS.* `EC_KEY * ssl_ecdh_exchange_callback (SSL *ssl, int is_export, int keylength)`
- *Callback handler for the EC Diffie-Hellman parameter generation process necessary for PFS.* `int ssl_dh_generate_callback (int p, int n, BN_GENCB *cb)`

The DH param generation callback. Variables

- `char ssl_version [16]`
- `pthread_mutex_t ** ssl_locks = NULL`

Detailed Description

The interface to OpenSSL routines.

Definition in file `openssl.c`.

Function Documentation

bool_t lib_load_openssl (void)

Initialize the openssl library and bind dynamically to the exported functions that are required.

openssl.c

Returns:

true on success or false on failure.

Definition at line 31 of file openssl.c.

References lib_symbols(), M_BIND, ns_length_get(), pl_char_get(), pl_length_int(), placer_t, ssl_version, and tok_get_ns().

Referenced by lib_load().

const char* lib_version_openssl (void)

Return the version string of the openssl library.

Returns:

a pointer to a character string containing the libopenssl version information.

Definition at line 22 of file openssl.c.

References ssl_version.

Referenced by lib_load().

SSL* ssl_alloc (void * server, int sockd, int flags)

Create an SSL session for a file descriptor, and accept the client TLS/SSL handshake.

See also:

SSL_accept()

BIO_new_socket()

Parameters:

server a server object which contains the underlying SSL context.

sockd the file descriptor of the TCP connection to be made SSL-ready.

flags passed to BIO_new_socket(), determines whether the socket is shut down when the BIO is freed.

Definition at line 282 of file openssl.c.

References BIO_new_socket_d, buflen, bufptr, server_t::context, log_pedantic, server_t::ssl, SSL_accept_d, ssl_error_string(), SSL_free_d, SSL_new_d, and SSL_set_bio_d.

Referenced by imap_starttls(), pop_starttls(), protocol_secure(), and smtp_starttls().

void* ssl_client_create (int_t sockd)

Establish an SSL client wrapper around a socket descriptor.

Parameters:

sockd the file descriptor of the socket to have its transport security level upgraded.

Returns:

NULL on failure or a pointer to the SSL handle of the file descriptor if SSL negotiation was successful.

LOW: Add requisite config options and sandbox resources to verify server SSL certificates.

Definition at line 348 of file openssl.c.

References BIO_new_socket_d, log_pedantic, MEMORYBUF, SSL_connect_d, SSL_CTX_ctrl_d, SSL_CTX_free_d, SSL_CTX_new_d, ssl_error_string(), SSL_free_d, SSL_new_d, SSL_set_bio_d, and SSLv23_client_method_d.

Referenced by client_secure().

DH* ssl_dh_exchange_callback (SSL * ssl, int is_export, int keylength)

Callback handler for the Diffie-Hellman parameter generation process necessary for PFS.

Parameters:

ssl the SSL session for which the callback was triggered.

is_export LOL. We're definitely ignoring this parameter.

keylength the length, in bits, of the DH key to be generated.

Returns:

a pointer to a Diffie-Hellman key of proper size with parameters generated, or NULL on failure.

Definition at line 613 of file openssl.c.

References DH_free_d, DH_generate_parameters_ex_d, DH_new_d, log_error, log_pedantic, and ssl_dh_generate_callback().

Referenced by ssl_server_create().

int ssl_dh_generate_callback (int p, int n, BN_GENCB * cb)

The DH param generation callback.

Definition at line 699 of file openssl.c.

Referenced by ssl_dh_exchange_callback().

EC_KEY* ssl_ecdh_exchange_callback (SSL * ssl, int is_export, int keylength)

Callback handler for the EC Diffie-Hellman parameter generation process necessary for PFS.

Parameters:

ssl the SSL session for which the callback was triggered.

is_export LOL. We're definitely ignoring this parameter.

keylength the length, in bits, of the ECCH key to be generated.

Returns:

a pointer to a ECDH key of proper size with parameters generated, or NULL on failure.

Definition at line 663 of file openssl.c.

References EC_GROUP_set_point_conversion_form_d, EC_KEY_get0_group_d,
EC_KEY_new_by_curve_name_d, and log_error.
Referenced by ssl_server_create().

char* ssl_error_string (chr_t * *buffer*, int_t *length*)

Get a textual representation of the last openssl error message.

Parameters:

buffer a buffer that will receive the last openssl error message.

length the size, in bytes, of the buffer that will contain the last openssl error message.

Returns:

NULL on failure, or a pointer to the buffer where the last openssl error message has been stored.

Definition at line 87 of file openssl.c.

References ERR_error_string_n_d, ERR_get_error_d, and log_pedantic.

Referenced by rand_get_int16(), rand_get_int32(), rand_get_int64(), rand_get_int8(), rand_get_uint16(),
rand_get_uint32(), rand_get_uint64(), rand_get_uint8(), ssl_alloc(), ssl_client_create(), ssl_verify_privkey(),
and ssl_write().

void ssl_free (SSL * *ssl*)

Shutdown and free an SSL connection.

Parameters:

ssl the SSL connection to be shut down.

Returns:

This function returns no value.

Definition at line 235 of file openssl.c.

References ERR_remove_state_d, log_pedantic, SSL_free_d, and SSL_shutdown_d.

Referenced by client_close(), con_destroy(), and protocol_secure().

void ssl_locking_callback (int *mode*, int *n*, const char * *file*, int *line*)

The locking function callback necessary for all multi-threaded applications using openssl.

See also:

CRYPTO_set_locking_callback()

Parameters:

mode a bitmask specifying the requested openssl locking operation (CRYPTO_LOCK, CRYPTO_WRITE,
CRYPTO_READ, or CRYPTO_UNLOCK).

n the zero-based index of the lock that is the target of the requested operation.

file a null-terminated string containing the filename of the function setting the lock, for debugging
purposes.

line the line number of the function setting the lock, for debugging purposes.

Returns:

This function returns no value.

Definition at line 491 of file openssl.c.

References mutex_lock(), mutex_unlock(), and ssl_locks.

Referenced by ssl_start().

int ssl_print (SSL * *ssl*, const char * *format*, va_list *args*)

Write formatted data to an SSL connection.

Parameters:

ssl the SSL connection to which the data will be written.

format a format string specifying the data to be written to the SSL connection.

va_list a variable argument list containing the data parameters associated with the format string.

Returns:

-1 on error, or the number of bytes written to the SSL connection.

Definition at line 175 of file openssl.c.

References buflen, bufptr, length, mm_alloc(), mm_free(), and ssl_write().

int ssl_read (SSL * *ssl*, void * *buffer*, int *length*, bool_t *block*)

Read data from an SSL connection.

Parameters:

ssl the SSL connection from which the data will be read.

buffer a pointer to the buffer where the read data will be stored.

length the length, in bytes, of the amount of data to be read.

block a boolean variable specifying whether the read operation should block.

Returns:

-1 on failure, 0 if the connection has been closed, or the number of bytes read from the connection on success.

Definition at line 110 of file openssl.c.

References buflen, bufptr, ERR_error_string_n_d, log_pedantic, SSL_get_error_d, SSL_peek_d, and SSL_read_d.

Referenced by client_read(), client_read_line(), con_read(), and con_read_line().

bool_t ssl_server_create (void * *server*)

Definition at line 398 of file openssl.c.

References server_t::certificate, server_t::context, log_critical, MAGMA_CIPHER_LIST, server_t::ssl, SSL_CTX_check_private_key_d, SSL_CTX_ctrl_d, SSL_CTX_new_d, SSL_CTX_set_cipher_list_d, SSL_CTX_set_tmp_dh_callback_d, SSL_CTX_set_tmp_ecdh_callback_d, SSL_CTX_use_certificate_chain_file_d, SSL_CTX_use_PrivateKey_file_d, ssl_dh_exchange_callback(), ssl_ecdh_exchange_callback(), and SSLv23_server_method_d.

Referenced by servers_encryption_start().

void ssl_server_destroy (void * server)

Destroy an SSL context associated with a server.

Parameters:

server the server to be deactivated.

Returns:

This function returns no value.

Definition at line 328 of file openssl.c.

References `server_t::context`, `log_pedantic`, `server_t::ssl`, and `SSL_CTX_free_d`.

Referenced by `servers_encryption_stop()`.

int ssl_shutdown_get (SSL * ssl)

Checks whether an SSL tunnel has been shut down or not.

See also:

`SSL_get_shutdown()`

Parameters:

ssl the SSL connection to be shut down.

Returns:

0 if the connection is alive and well, or `SSL_SENT_SHUTDOWN/SSL_RECEIVED_SHUTDOWN`

Definition at line 265 of file openssl.c.

References `SSL_get_shutdown_d`.

Referenced by `con_read()`.

bool_t ssl_start (void)

Initialize the openssl facility.

Note:

First, this function initializes the mutexes necessary for the locking function callback that openssl uses for shared data structures in multi-threaded applications. Next, the DKIM key is retrieved if the **magma.dkim.enabled** configuration variable is set.

Returns:

true if openssl was initialized successfully, or false on failure.

Definition at line 525 of file openssl.c.

References `CRYPTO_num_locks_d`, `CRYPTO_set_id_callback_d`, `CRYPTO_set_locking_callback_d`, `magma_t::dkim`, `magma_t::enabled`, `file_load()`, `file_world_accessible()`, `log_critical`, `magma`, `mm_alloc()`, `mutex_init()`, `OPENSSL_add_all_algorithms_noconf_d`, `magma_t::privkey`, `SSL_library_init_d`, `SSL_load_error_strings_d`, `ssl_locking_callback()`, `ssl_locks`, `ssl_thread_id_callback()`, `ssl_verify_privkey()`, `st_char_get()`, and `st_free()`.

Referenced by `process_start()`.

void ssl_stop (void)

Stop the openssl library and cleanup all associated data structures.

Returns:

This function returns no values.

Definition at line 448 of file openssl.c.

References ASN1_STRING_TABLE_cleanup_d, BIO_sock_cleanup_d, COMP_zlib_cleanup_d, CONF_modules_unload_d, CRYPTO_cleanup_all_ex_data_d, CRYPTO_num_locks_d, CRYPTO_set_id_callback_d, CRYPTO_set_locking_callback_d, ENGINE_cleanup_d, ERR_free_strings_d, ERR_remove_state_d, EVP_cleanup_d, mm_free(), mutex_destroy(), OBJ_cleanup_d, OBJ_NAME_cleanup_d, sk_pop_free_d, and ssl_locks.

Referenced by process_stop().

unsigned long ssl_thread_id_callback (void)

The thread id callback necessary for all multi-threaded applications using openssl.

See also:

CRYPTO_set_id_callback()

Returns:

the id of the calling thread.

Definition at line 513 of file openssl.c.

References thread_get_thread_id().

Referenced by ssl_start().

void ssl_thread_stop (void)

Free the calling thread's SSL error queue.

See also:

ssl_thread_stop()

Returns:

This function returns no value.

Definition at line 223 of file openssl.c.

References ERR_remove_state_d.

Referenced by thread_stop().

bool_t ssl_verify_privkey (const char * keyfile)

Verify that the filename contains a valid private key in PEM format.

Parameters:

keyfile the pathname of the private key.

Returns:

true if the the key is valid, or false if it is not.

Definition at line 586 of file openssl.c.

References `log_pedantic`, `MEMORYBUF`, `SSL_CTX_free_d`, `SSL_CTX_new_d`, `SSL_CTX_use_PrivateKey_file_d`, `ssl_error_string()`, and `SSLv23_client_method_d`.

Referenced by `ssl_start()`.

int `ssl_write` (`SSL *ssl`, `const void *buffer`, `int length`)

Write data to an open SSL connection.

Parameters:

ssl the SSL connection to which the data will be written.

buffer a pointer to the buffer containing the data to be written.

length the length, in bytes, of the data to be written.

Returns:

-1 on error, or the number of bytes written to the SSL connection.

Definition at line 145 of file openssl.c.

References `buflen`, `bufptr`, `log_pedantic`, `ssl_error_string()`, `SSL_get_error_d`, and `SSL_write_d`.

Referenced by `client_write()`, `con_write_bl()`, and `ssl_print()`.

Variable Documentation

`pthread_mutex_t ssl_locks = NULL`**

Definition at line 16 of file openssl.c.

Referenced by `ssl_locking_callback()`, `ssl_start()`, and `ssl_stop()`.

`char ssl_version[16]`

Definition at line 15 of file openssl.c.

Referenced by `lib_load_openssl()`, and `lib_version_openssl()`.

magma/providers/cryptography/random.c File Reference

A collection of functions for generating random data.

```
#include "magma.h"
```

Functions

- **stringer_t * rand_choices** (**chr_t** *choices, **size_t** len)
- *Get a random string of data of a specified size, populated with characters from a chosen set. **size_t rand_write** (**stringer_t** *s)*
- *Fill a managed string with random data. **uint64_t rand_get_uint64** (void)*
- *Generate a random unsigned 64 bit number. **uint32_t rand_get_uint32** (void)*
- *Generate a random unsigned 32 bit number. **uint16_t rand_get_uint16** (void)*
- *Generate a random unsigned 16 bit number. **uint8_t rand_get_uint8** (void)*
- *Generate a random unsigned 8 bit number. **int64_t rand_get_int64** (void)*
- *Generate a random signed 64 bit number. **int32_t rand_get_int32** (void)*
- **int16_t rand_get_int16** (void)
- **int8_t rand_get_int8** (void)
- **bool_t rand_thread_start** (void)
- **bool_t rand_start** (void)
- *Initialize random number generation services and seed the generator. **void rand_stop** (void)*

Variables

- **__thread uint_t rand_ctx** = 0

Detailed Description

A collection of functions for generating random data.

Definition in file **random.c**.

Function Documentation

stringer_t* rand_choices (**chr_t** * choices, **size_t** len)

Get a random string of data of a specified size, populated with characters from a chosen set.

Parameters:

choices a pointer to a null-terminated string containing a pool of characters from which the contents of the random data will be selected.

len the length, in bytes, of the random string that will be returned to the caller.

Returns:

NULL on failure or a pointer to a managed string containing len bytes of random data on success.

Definition at line 28 of file random.c.

References `log_pedantic`, `ns_length_get()`, `RAND_bytes_d`, `st_alloc`, `st_data_get()`, `st_free()`, and `st_length_set()`.

Referenced by mail_create_message(), register_print_captcha(), register_session_generate(), smtp_bounce(), and smtp_reply().

int16_t rand_get_int16 (void)

Definition at line 221 of file random.c.

References buflen, bufptr, log_pedantic, RAND_bytes_d, rand_ctx, and ssl_error_string().

int32_t rand_get_int32 (void)

Definition at line 203 of file random.c.

References buflen, bufptr, log_pedantic, RAND_bytes_d, rand_ctx, and ssl_error_string().

int64_t rand_get_int64 (void)

Generate a random signed 64 bit number.

Note:

This function attempts to generate random data securely, but falls back on the pseudo-random number generator.

Returns:

the newly generated signed 64 bit integer.

See also:

RAND_bytes()

Definition at line 183 of file random.c.

References buflen, bufptr, log_pedantic, RAND_bytes_d, rand_ctx, and ssl_error_string().

Referenced by portal_message_source().

int8_t rand_get_int8 (void)

Definition at line 240 of file random.c.

References buflen, bufptr, log_pedantic, RAND_bytes_d, rand_ctx, and ssl_error_string().

uint16_t rand_get_uint16 (void)

Generate a random unsigned 16 bit number.

Note:

This function attempts to generate random data securely, but falls back on the pseudo-random number generator.

Returns:

the newly generated unsigned 16 bit integer.

See also:

RAND_bytes()

Definition at line 144 of file random.c.

References buflen, bufptr, log_pedantic, RAND_bytes_d, rand_ctx, and ssl_error_string().

uint32_t rand_get_uint32 (void)

Generate a random unsigned 32 bit number.

Note:

This function attempts to generate random data securely, but falls back on the pseudo-random number generator.

Returns:

the newly generated unsigned 32 bit integer.

See also:

RAND_bytes()

Definition at line 126 of file random.c.

References buflen, bufptr, log_pedantic, RAND_bytes_d, rand_ctx, and ssl_error_string().

Referenced by process_maint(), register_captcha_generate(), register_captcha_random_font(), register_captcha_write_noise(), and smtp_client_connect().

uint64_t rand_get_uint64 (void)

Generate a random unsigned 64 bit number.

Note:

This function attempts to generate random data securely, but falls back on the pseudo-random number generator.

Returns:

the newly generated unsigned 64 bit integer.

See also:

RAND_bytes()

Definition at line 105 of file random.c.

References buflen, bufptr, log_pedantic, RAND_bytes_d, rand_ctx, and ssl_error_string().

Referenced by mail_mime_generate_boundary(), sess_key(), smtp_store_spamsig(), and spool_mktemp().

uint8_t rand_get_uint8 (void)

Generate a random unsigned 8 bit number.

Note:

This function attempts to generate random data securely, but falls back on the pseudo-random number generator.

Returns:

the newly generated unsigned 8 bit integer.

See also:

RAND_bytes()

Definition at line 163 of file random.c.

References buflen, bufptr, log_pedantic, RAND_bytes_d, rand_ctx, and ssl_error_string().

bool_t rand_start (void)

Initialize random number generation services and seed the generator.

random.c**Note:**

The default seed source for cryptographically secure generation routines is the system device /dev/random.

Returns:

false on failure, true on success.

Definition at line 276 of file random.c.

References magma_t::cryptography, magma_t::iface, magma, rand_ctx, RAND_load_file_d, RAND_status_d, and thread_get_thread_id().

Referenced by process_start().

void rand_stop (void)

Definition at line 295 of file random.c.

References RAND_cleanup_d.

Referenced by process_stop().

bool_t rand_thread_start (void)

Definition at line 260 of file random.c.

References magma_t::cryptography, magma_t::iface, magma, rand_ctx, and thread_get_thread_id().

size_t rand_write (stringer_t * s)

Fill a managed string with random data.

Note:

This function generates random data using the cryptographically strong Openssl function RAND_bytes().

Parameters:

s the input managed string.

Returns:

0 on failure, or the total number of bytes written to the managed string.

See also:

RAND_bytes()

Definition at line 68 of file random.c.

References log_pedantic, RAND_bytes_d, st_avail_get(), st_data_get(), st_length_get(), st_length_set(), st_valid_avail(), st_valid_destination(), and st_valid_tracked().

Referenced by `symmetric_vector()`.

Variable Documentation

`__thread uint_t rand_ctx = 0`

Definition at line 16 of file `random.c`.

Referenced by `rand_get_int16()`, `rand_get_int32()`, `rand_get_int64()`, `rand_get_int8()`, `rand_get_uint16()`, `rand_get_uint32()`, `rand_get_uint64()`, `rand_get_uint8()`, `rand_start()`, and `rand_thread_start()`.

magma/providers/cryptography/scramble.c File Reference

Functions used to handle symmetric encryption.

```
#include "magma.h"
```

Functions

- `uint64_t scramble_total_length (scramble_t *buffer)`
- *TODO: Create envelope functions to translate the scrambled buffer into a condensed/encoded stringer.* `uint64_t scramble_body_hash (scramble_t *buffer)`
- *Get a hash of a scrambled object's (encrypted) body data.* `uint64_t scramble_orig_length (scramble_t *buffer)`
- *Get the length of the original (unencrypted) data underlying a scrambled object.* `uint64_t scramble_body_length (scramble_t *buffer)`
- *Get the length of a scrambled object's (encrypted) body data.* `uint64_t scramble_vector_length (scramble_t *buffer)`
- *Get the length of a scrambled object's IV.* `void * scramble_body_data (scramble_t *buffer)`
- *Get a pointer to the start of the encrypted data from a scrambled data header.* `void * scramble_vector_data (scramble_t *buffer)`
- *Get a pointer to the start of the IV from a scrambled data header.* `scramble_t * scramble_import (stringer_t *s)`
- *Return a managed string as a scrambled buffer, after validation.* `scramble_t * scramble_alloc (size_t length)`
- *Allocate a new scrambled data block.* `void scramble_free (scramble_t *buffer)`
- *Free a scrambled data block.* `scramble_t * scramble_encrypt (stringer_t *key, stringer_t *input)`
- *Scramble a block of data using an encryption key.* `stringer_t * scramble_decrypt (stringer_t *key, scramble_t *input)`

Un-scramble a block of data using an decryption key.

Detailed Description

Functions used to handle symmetric encryption.

Definition in file `scramble.c`.

Function Documentation

`scramble_t* scramble_alloc (size_t length)`

Allocate a new scrambled data block.

scramble.c

Parameters:

length the length, in bytes, of the scrambled data buffer (should include the IV and encrypted body length).

Returns:

a pointer to a newly allocated scrambled data header.

Definition at line 144 of file `scramble.c`.

References `mm_alloc()`.

Referenced by `scramble_encrypt()`.

void* scramble_body_data (scramble_t * *buffer*)

Get a pointer to the start of the encrypted data from a scrambled data header.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

a pointer to the start of the scrambled data header's associated encrypted data body.

Definition at line 84 of file scramble.c.

References `scramble_vector_length()`.

Referenced by `scramble_decrypt()`, `scramble_encrypt()`, and `scramble_import()`.

uint64_t scramble_body_hash (scramble_t * *buffer*)

Get a hash of a scrambled object's (encrypted) body data.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

the 64-bit hash of the specified scrambled object's body data.

Definition at line 36 of file scramble.c.

uint64_t scramble_body_length (scramble_t * *buffer*)

Get the length of a scrambled object's (encrypted) body data.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

the length, in bytes, of the scrambled object's body data.

Definition at line 60 of file scramble.c.

Referenced by `scramble_decrypt()`, and `scramble_total_length()`.

stringer_t* scramble_decrypt (stringer_t * *key*, scramble_t * *input*)

Un-scramble a block of data using an decryption key.

Parameters:

key a managed string containing the symmetric decryption key.

input a pointer to the scrambled data header to be decrypted.

Returns:

NULL on failure, or a managed string containing the verified decrypted data.

Definition at line 225 of file scramble.c.

References cipher_id(), hash_adler32(), log_info, log_pedantic, MANAGEDBUF, PLACER, scramble_body_data(), scramble_body_length(), scramble_vector_data(), scramble_vector_length(), st_free(), st_length_get(), symmetric_decrypt(), and symmetric_key().

Referenced by meta_data_user_build_storage_keys(), and sess_get().

scramble_t* scramble_encrypt (stringer_t * key, stringer_t * input)

Scramble a block of data using an encryption key.

Note:

This function is configured to use AES 256 in CBC mode.

Parameters:

key a managed string containing the symmetric encryption key.

input a managed string containing the data to be encrypted.

Returns:

NULL on failure, or a pointer to a scrambled data header containing the encrypted data and its metadata.

Definition at line 168 of file scramble.c.

References cipher_id(), cipher_numeric_id(), hash_adler32(), log_pedantic, MANAGEDBUF, mm_copy(), scramble_alloc(), scramble_body_data(), scramble_vector_data(), st_data_get(), st_free(), st_length_get(), symmetric_encrypt(), symmetric_key(), and symmetric_vector().

Referenced by meta_data_user_save_storage_keys(), and sess_token().

void scramble_free (scramble_t * buffer)

Free a scrambled data block.

Parameters:

buffer a pointer to the scrambled data header to be freed.

Returns:

This function returns no value.

Definition at line 154 of file scramble.c.

References mm_free().

Referenced by meta_data_user_save_storage_keys(), and sess_token().

scramble_t* scramble_import (stringer_t * s)

Return a managed string as a scrambled buffer, after validation.

Parameters:

s a managed string containing the serialized scrambled data.

Returns:

NULL on failure, or a pointer to the scrambled data header on success.

Definition at line 104 of file scramble.c.

References hash_adler32(), log_pedantic, scramble_body_data(), st_data_get(), st_empty(), and st_length_get().

Referenced by sess_get().

uint64_t scramble_orig_length (scramble_t * *buffer*)

Get the length of the original (unencrypted) data underlying a scrambled object.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

the length, in bytes, of the scrambled object's original data.

Definition at line 48 of file scramble.c.

uint64_t scramble_total_length (scramble_t * *buffer*)

TODO: Create envelope functions to translate the scrambled buffer into a condensed/encoded stringer.

Get the total length of a scrambled object.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

the total length, in bytes, of the scrambled object.

Definition at line 22 of file scramble.c.

References scramble_body_length(), and scramble_vector_length().

Referenced by meta_data_user_save_storage_keys(), and sess_token().

void* scramble_vector_data (scramble_t * *buffer*)

Get a pointer to the start of the IV from a scrambled data header.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

a pointer to the start of the scrambled data header's associated IV block.

Definition at line 94 of file scramble.c.

Referenced by scramble_decrypt(), and scramble_encrypt().

uint64_t scramble_vector_length (scramble_t * *buffer*)

Get the length of a scrambled object's IV.

Parameters:

buffer a pointer to the header of the scrambled data.

Returns:

the length, in bytes, of the scrambled object's IV.

Definition at line 72 of file scramble.c.

Referenced by `scramble_body_data()`, `scramble_decrypt()`, and `scramble_total_length()`.

magma/providers/cryptography/symmetric.c File Reference

Functions used to encrypt/decrypt data using symmetric ciphers.

```
#include "magma.h"
```

Functions

- **stringer_t * symmetric_encrypt** (**cipher_t** *cipher, **stringer_t** *vector, **stringer_t** *key, **stringer_t** *input)
- *Encrypt a block of data using a symmetric cipher.* **stringer_t * symmetric_decrypt** (**cipher_t** *cipher, **stringer_t** *vector, **stringer_t** *key, **stringer_t** *input)
- **symmetric.c stringer_t * symmetric_vector** (**cipher_t** *cipher, **stringer_t** *output)
- *Generate an IV data suitable for the specified cipher.* **stringer_t * symmetric_key** (**cipher_t** *cipher, **stringer_t** *key, **stringer_t** *output)

Derive a symmetric key from a user's password.

Detailed Description

Functions used to encrypt/decrypt data using symmetric ciphers.

Definition in file **symmetric.c**.

Function Documentation

stringer_t* symmetric_decrypt (**cipher_t** * *cipher*, **stringer_t** * *vector*, **stringer_t** * *key*,
stringer_t * *input*)

symmetric.c

Definition at line 108 of file symmetric.c.

References ERR_error_string_d, ERR_get_error_d, EVP_CIPHER_CTX_block_size_d,
EVP_CIPHER_CTX_cleanup_d, EVP_CIPHER_CTX_init_d, EVP_CIPHER_CTX_iv_length_d,
EVP_CIPHER_CTX_key_length_d, EVP_DecryptFinal_ex_d, EVP_DecryptInit_ex_d,
EVP_DecryptUpdate_d, log_pedantic, st_alloc, st_data_get(), st_empty_out(), st_free(), and st_length_set().

Referenced by scramble_decrypt().

stringer_t* symmetric_encrypt (**cipher_t** * *cipher*, **stringer_t** * *vector*, **stringer_t** * *key*,
stringer_t * *input*)

Encrypt a block of data using a symmetric cipher.

Parameters:

cipher the cipher type being used to encrypt the data.
vector the IV used for the encryption operation.
key the symmetric encryption key used to encrypt the data.
input a managed string containing the data to be encrypted.

Returns:

NULL on failure, or a pointer to a managed string containing the encrypted data.

Definition at line 24 of file symmetric.c.

References `EVP_CIPHER_CTX_block_size_d`, `EVP_CIPHER_CTX_cleanup_d`, `EVP_CIPHER_CTX_init_d`, `EVP_CIPHER_CTX_iv_length_d`, `EVP_CIPHER_CTX_key_length_d`, `EVP_EncryptFinal_ex_d`, `EVP_EncryptInit_ex_d`, `EVP_EncryptUpdate_d`, `log_pedantic`, `st_alloc`, `st_data_get()`, `st_empty_out()`, `st_free()`, and `st_length_set()`.

Referenced by `scramble_encrypt()`.

`stringer_t* symmetric_key (cipher_t * cipher, stringer_t * key, stringer_t * output)`

Derive a symmetric key from a user's password.

Note:

Depending on the length of the password, either SHA1, SHA224, or SHA256 may be used as the hashing algorithm.

If output is passed as NULL, a new managed string will be allocated to hold the output of the operation.

Parameters:

cipher the specified cipher type.

key the user's password, as a managed string.

output the output managed string to receive the digested password. Can be NULL.

Returns:

NULL on failure, or a pointer to the managed string containing the digested password.

Definition at line 234 of file symmetric.c.

References `cipher_key_length()`, `digest_sha1()`, `digest_sha224()`, `digest_sha256()`, `st_length_get()`, `st_length_set()`, `st_output()`, and `st_valid_tracked()`.

Referenced by `scramble_decrypt()`, and `scramble_encrypt()`.

`stringer_t* symmetric_vector (cipher_t * cipher, stringer_t * output)`

Generate an IV data suitable for the specified cipher.

Note:

If output is NULL, a new managed string will be allocated and returned by the function.

Parameters:

cipher the cipher type to be used.

output the managed string that will store the IV data.

Returns:

NULL on failure, or a pointer to the managed string that contains the IV data.

Definition at line 199 of file symmetric.c.

References `cipher_vector_length()`, `log_pedantic`, `PLACER`, `rand_write()`, `st_data_get()`, `st_length_set()`, `st_output()`, and `st_valid_tracked()`.

Referenced by `scramble_encrypt()`.

magma/providers/database/database.h File Reference

Functions used to interface with the database.

Defines

- `#define ISNULL(b) (my_bool *)&((my_bool){ b })`

Typedefs

- `typedef char table_t`
- `typedef char row_t`

Functions

- `bool_t lib_load_mysql (void)`
- `mysql.c` `const char * lib_version_mysql ()`
- *Return the version string of libmysql.* `const char * serv_charset_mysql (void)`
- `const char * serv_schema_mysql (void)`
- `const char * serv_type_mysql (void)`
- *Determine whether the mysql library in use is embedded or not.* `const char * serv_version_mysql (void)`
- *Return the server version string of the mysql database.* `uint_t sql_errno (MYSQL *mysql)`
- *Get the last error number for a mysql connection.* `const chr_t * sql_error (MYSQL *mysql)`
- *Get a human-readable error message for a mysql connection.* `MYSQL * sql_open (bool_t silent)`
- *Open up a new mysql connection to the magma-configured database server.* `int_t sql_ping (uint32_t connection)`
- `bool_t sql_start (void)`
- *Load up the mysql subsystem.* `void sql_stop (void)`
- *Shutdown and free the pool of mysql connections.* `bool_t sql_thread_start (void)`
- *Initialize mysql thread specific variables for the calling thread.* `void sql_thread_stop (void)`
- *Prepare the thread for exiting to destroy mysql thread specific variables.* `int64_t sql_query (stringer_t *query)`
- `query.c` `int64_t sql_query_conn (stringer_t *query, uint32_t connection)`
- *Execute a mysql statement.* `bool_t res_field_bool (row_t *row, uint64_t field)`
- `results.c` `bool_t res_row_store (uint64_t num, table_t *table, MYSQL_BIND *binding)`
- `double_t res_field_double (row_t *row, uint64_t field)`
- *Get the value of a specified field in a database result row as a double.* `float_t res_field_float (row_t *row, uint64_t field)`
- *Get the value of a specified field in a database result row as a float.* `int16_t res_field_int16 (row_t *row, uint64_t field)`
- *Get the value of a specified field in a database result row as a signed 16-bit integer.* `int32_t res_field_int32 (row_t *row, uint64_t field)`
- *Get the value of a specified field in a database result row as a signed 32-bit integer.* `int64_t res_field_int64 (row_t *row, uint64_t field)`
- *Get the value of a specified field in a database result row as a signed 64-bit integer.* `int8_t res_field_int8 (row_t *row, uint64_t field)`
- *Get the value of a specified field in a database result row as a signed 8-bit integer.* `row_t * res_row_get (table_t *table, uint64_t row)`
- *Retrieve a row from a database results table by index.* `row_t * res_row_next (table_t *table)`
- *Return the next row in the database results table and advance the cursor.* `size_t res_field_length (row_t *row, uint64_t field)`
- *Get the length of a specified field in a database result row.* `stringer_t * res_field_string (row_t *row, uint64_t field)`

- *Get the value of a specified field in a database result row as managed string.* **table_t * res_stmt_store** (MYSQL_STMT *stmt)
- **table_t * res_table_alloc** (uint64_t rows, uint64_t fields)
- **row_t * res_field_generic** (row_t *row, uint64_t field, size_t typesize)
- **uint16_t res_field_uint16** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as an unsigned 16-bit integer.* **uint32_t res_field_uint32** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as an unsigned 32-bit integer.* **uint64_t res_bind_create** (MYSQL_STMT *stmt, MYSQL_BIND **result)
- **uint64_t res_field_count** (table_t *table)
- *Return the number of fields stored in the database results table.* **uint64_t res_field_uint64** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as an unsigned 64-bit integer.* **uint64_t res_row_count** (table_t *table)
- *Return the number of rows in the mysql results table.* **uint8_t res_field_uint8** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as an unsigned 8-bit integer.* **void * res_field_block** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as generic pointer.* **void res_bind_free** (MYSQL_STMT *stmt, MYSQL_BIND *binding, uint64_t number)
- *Free a prepared mysql statement result set binding.* **void res_row_set** (row_t *row, chr_t *buffer)
- *Set the buffer location for a database result row.* **void res_table_free** (table_t *table)
- *Free a mysql results table.* **bool_t stmt_bind_param** (MYSQL_STMT *group, MYSQL_BIND *bind)
- *stmts.c* **void stmt_close** (MYSQL_STMT *local)
- **uint_t stmt_errno** (MYSQL_STMT *local)
- *Get the error number associated with a mysql statement.* **const chr_t * stmt_error** (MYSQL_STMT *local)
- **bool_t stmt_exec** (MYSQL_STMT **group, MYSQL_BIND *parameters)
- *Execute a prepared mysql statement.* **uint64_t stmt_exec_affected** (MYSQL_STMT **group, MYSQL_BIND *parameters)
- *Execute a statement and return the number of affected rows.* **uint64_t stmt_exec_affected_conn** (MYSQL_STMT **group, MYSQL_BIND *parameters, uint32_t connection)
- *Execute a statement on a specified mysql connection and return the number of affected rows.* **bool_t stmt_exec_conn** (MYSQL_STMT **group, MYSQL_BIND *parameters, uint32_t connection)
- *Execute a prepared mysql statement over a specified connection.* **table_t * stmt_get_result** (MYSQL_STMT **group, MYSQL_BIND *parameters)
- *Execute a prepared mysql statement and return the result.* **table_t * stmt_get_result_conn** (MYSQL_STMT **group, MYSQL_BIND *parameters, uint32_t connection)
- *Execute a prepared mysql statement on a specified connection and return the result.* **uint64_t stmt_insert** (MYSQL_STMT **group, MYSQL_BIND *parameters)
- *Execute a mysql prepared INSERT or UPDATE statement.* **uint64_t stmt_insert_conn** (MYSQL_STMT **group, MYSQL_BIND *parameters, uint32_t connection)
- *Execute a mysql prepared INSERT or UPDATE statement on a specified connection.* **MYSQL_STMT * stmt_open** (MYSQL *mysql)
- *Initialize a mysql prepared statement.* **bool_t stmt_prepare** (MYSQL_STMT *group, const char *query, unsigned long length)
- **bool_t stmt_rebuild** (uint32_t connection)
- **MYSQL_STMT * stmt_reset** (MYSQL_STMT **group, uint32_t connection)
- **bool_t stmt_start** (void)
- *Initialize the global array of mysql prepared statements.* **void stmt_stop** (void)
- **int64_t tran_commit** (int64_t transaction)
- *transaction.c* **int64_t tran_rollback** (int64_t transaction)
- *Rollback a pending transaction in the mysql database.* **int64_t tran_start** (void)

Pull a connection from the sql pool and start a transaction. Variables

- `pool_t * sql_pool`
-

Detailed Description

Functions used to interface with the database.

MySQL Data Types

MYSQL_TYPE_TINY for 8-bit integer variables. Normally it's 'signed char' and 'unsigned char'; MYSQL_TYPE_SHORT for 16-bit signed and unsigned variables. This is usually 'short' and 'unsigned short'; MYSQL_TYPE_LONG for 32-bit signed and unsigned variables. It corresponds to 'int' and 'unsigned int' on vast majority of platforms. On IA-32 and some other 32-bit systems you can also use 'long' here; MYSQL_TYPE_LONGLONG 64-bit signed or unsigned integer. Stands for '[unsigned] long long' on most platforms; MYSQL_TYPE_FLOAT 32-bit floating point type, 'float' on most systems; MYSQL_TYPE_DOUBLE 64-bit floating point type, 'double' on most systems; MYSQL_TYPE_TIME broken-down time stored in MYSQL_TIME structure MYSQL_TYPE_DATE date stored in MYSQL_TIME structure MYSQL_TYPE_DATETIME datetime stored in MYSQL_TIME structure See more on how to use these types for sending dates and times below; MYSQL_TYPE_STRING character string, assumed to be in character-set-client. If character set of client is not equal to character set of column, value for this placeholder will be converted to destination character set before insert. MYSQL_TYPE_BLOB sequence of bytes. This sequence is assumed to be in binary character set (which is the same as no particular character set), and is never converted to any other character set. See also notes about supplying string/blob length below. MYSQL_TYPE_NULL special typecode for binding nulls.

Definition in file **database.h**.

Define Documentation

#define ISNULL(b) (my_bool *)&((my_bool){ b })

Definition at line 66 of file database.h.

Referenced by `contact_update()`, `mail_db_insert_duplicate_message()`, and `mail_db_insert_message()`.

Typedef Documentation

typedef char row_t

Definition at line 63 of file database.h.

typedef char table_t

Definition at line 62 of file database.h.

Function Documentation

bool_t lib_load_mysql (void)

mysql.c

mysql.c

Returns:

true on success or false on failure.

Definition at line 487 of file mysql.c.

References lib_symbols(), M_BIND, and mysql_character_set_name().

Referenced by lib_load().

const char* lib_version_mysql (void)

Return the version string of libmysql.

Returns:

a pointer to a character string containing the libmysql version information.

Definition at line 464 of file mysql.c.

References mysql_get_client_version_d, and sql.

Referenced by lib_load().

uint64_t res_bind_create (MYSQL_STMT * *stmt*, MYSQL_BIND ** *result*)

Definition at line 356 of file results.c.

References length, log_info, mm_alloc(), mysql_fetch_field_d, mysql_free_result_d, mysql_num_fields_d, mysql_stmt_error_d, mysql_stmt_result_metadata_d, and res_bind_free().

Referenced by res_stmt_store().

void res_bind_free (MYSQL_STMT * *stmt*, MYSQL_BIND * *binding*, uint64_t *number*)

Free a prepared mysql statement result set binding.

Parameters:

stmt the input prepared mysql statement.

binding the binding for the result set.

number the number of fields in the result set.

Returns:

This function does not return a value.

Definition at line 298 of file results.c.

References length, mm_cleanup(), mm_free(), and mysql_stmt_bind_result_d.

Referenced by res_bind_create(), and res_stmt_store().

void* res_field_block (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as generic pointer.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a generic pointer, or NULL on failure.

Definition at line 51 of file results.c.

References res_field_generic().

Referenced by config_load_database_settings(), contact_details_fetch(), contacts_fetch(), magma_folder_fetch(), messages_fetch(), meta_data_fetch_alerts(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_messages(), meta_data_user_build_storage_keys(), smtp_check_receive_quota(), smtp_fetch_inbound(), user_config_fetch(), and warehouse_fetch_domains().

bool_t res_field_bool (row_t * row, uint64_t field)

results.c

results.c

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a boolean, or false on failure.

Definition at line 224 of file results.c.

References res_field_generic().

uint64_t res_field_count (table_t * table)

Return the number of fields stored in the database results table.

Parameters:

table the input database results table.

Returns:

0 on failure, or the number of table fields on success.

Definition at line 423 of file results.c.

References log_info.

Referenced by res_row_store().

double_t res_field_double (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as a double.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a double, or 0 on failure.
Definition at line 94 of file results.c.
References res_field_generic().

float_t res_field_float (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as a float.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a float, or 0 on failure.
Definition at line 107 of file results.c.
References res_field_generic().

row_t* res_field_generic (row_t * row, uint64_t field, size_t typesize)

Definition at line 17 of file results.c.

References debug_hook(), and log_info.

Referenced by res_field_block(), res_field_bool(), res_field_double(), res_field_float(), res_field_int16(), res_field_int32(), res_field_int64(), res_field_int8(), res_field_uint16(), res_field_uint32(), res_field_uint64(), and res_field_uint8().

int16_t res_field_int16 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as a signed 16-bit integer.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a signed 16-bit integer, or 0 on failure.
Definition at line 198 of file results.c.
References res_field_generic().

int32_t res_field_int32 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as a signed 32-bit integer.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a signed 32-bit integer, or 0 on failure.
Definition at line 185 of file results.c.
References res_field_generic().

int64_t res_field_int64 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as a signed 64-bit integer.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a signed 64-bit integer, or 0 on failure.
Definition at line 172 of file results.c.
References res_field_generic().

int8_t res_field_int8 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as a signed 8-bit integer.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a signed 8-bit integer, or 0 on failure.
Definition at line 211 of file results.c.
References res_field_generic().

Referenced by meta_data_fetch_user(), meta_data_user_build(), smtp_fetch_authorization(),
smtp_fetch_inbound(), teacher_data_fetch(), and warehouse_fetch_domains().

size_t res_field_length (row_t * row, uint64_t field)

Get the length of a specified field in a database result row.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the length of the specified result row, or 0 on failure.
Definition at line 237 of file results.c.

References log_info.

Referenced by config_load_database_settings(), contact_details_fetch(), contacts_fetch(), magma_folder_fetch(), messages_fetch(), meta_data_fetch_alerts(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_messages(), meta_data_user_build_storage_keys(), smtp_check_receive_quota(), smtp_fetch_inbound(), user_config_fetch(), and warehouse_fetch_domains().

stringer_t* res_field_string (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as managed string.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a managed string, or NULL on failure.
Definition at line 64 of file results.c.

References length, log_info, and st_import().

Referenced by meta_data_fetch_all_tags(), meta_data_fetch_message_tags(), meta_data_fetch_user(), register_data_fetch_blocklist(), smtp_fetch_authorization(), smtp_fetch_autoreply(), smtp_fetch_inbound(), smtp_rollout(), teacher_data_fetch(), and warehouse_fetch_patterns().

uint16_t res_field_uint16 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as an unsigned 16-bit integer.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as an unsigned 16-bit integer, or 0 on failure.
Definition at line 146 of file results.c.
References res_field_generic().

uint32_t res_field_uint32 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as an unsigned 32-bit integer.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as an unsigned 32-bit integer, or 0 on failure.
Definition at line 133 of file results.c.
References res_field_generic().

Referenced by magma_folder_fetch(), messages_fetch(), meta_data_fetch_folders(), meta_data_fetch_messages(), smtp_fetch_authorization(), smtp_fetch_inbound(), and smtp_rollout().

uint64_t res_field_uint64 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as an unsigned 64-bit integer.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as an unsigned 64-bit integer, or 0 on failure.

Definition at line 120 of file results.c.

References res_field_generic().

Referenced by config_fetch_host_number(), contact_details_fetch(), contacts_fetch(), magma_folder_fetch(), messages_fetch(), meta_data_acknowledge_alert(), meta_data_fetch_alerts(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_messages(), meta_data_user_build(), smtp_check_receive_quota(), smtp_check_transmit_quota(), smtp_fetch_authorization(), smtp_fetch_inbound(), smtp_rollout(), statistics_refresh(), teacher_data_fetch(), and user_config_fetch().

uint8_t res_field_uint8 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as an unsigned 8-bit integer.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as an unsigned 8-bit integer, or 0 on failure.

Definition at line 159 of file results.c.

References res_field_generic().

Referenced by meta_data_fetch_mailbox_aliases().

uint64_t res_row_count (table_t * table)

Return the number of rows in the mysql results table.

Note:

The row count is provided for a one-indexed table.

Parameters:

table the input mysql results table.

Returns:

0 on error, or the row count on success.

Definition at line 440 of file results.c.

References log_info.

Referenced by config_load_database_settings(), meta_data_fetch_message_tags(), res_row_store(), smtp_check_authorized_from(), and smtp_fetch_inbound().

row_t* res_row_get (table_t * table, uint64_t row)

Retrieve a row from a database results table by index.

Note:

This function operates on a 0-indexed table.

Parameters:

table the input database results table. *row* the row # to be fetched.

Returns:

NULL on failure, or a pointer to a result row on success.

Definition at line 477 of file results.c.

References log_info.

Referenced by config_load_database_settings(), res_row_next(), and res_row_store().

row_t* res_row_next (table_t * table)

Return the next row in the database results table and advance the cursor.

Parameters:

table the input mysql results table.

Returns:

a pointer to the next result row in the table.

Definition at line 500 of file results.c.

References count, and res_row_get().

Referenced by config_fetch_host_number(), contact_details_fetch(), contacts_fetch(), magma_folder_fetch(), messages_fetch(), meta_data_acknowledge_alert(), meta_data_check_mailbox(), meta_data_fetch_alerts(), meta_data_fetch_all_tags(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_message_tags(), meta_data_fetch_messages(), meta_data_fetch_user(), meta_data_user_build(), meta_data_user_build_storage_keys(), register_data_check_username(), register_data_fetch_blocklist(), smtp_check_receive_quota(), smtp_check_transmit_quota(), smtp_fetch_authorization(), smtp_fetch_autoreply(), smtp_fetch_inbound(), smtp_rollout(), statistics_refresh(), teacher_data_fetch(), user_config_fetch(), warehouse_fetch_domains(), and warehouse_fetch_patterns().

void res_row_set (row_t * row, chr_t * buffer)

Set the buffer location for a database result row.

Parameters:

row the input database result row.

buffer the buffer to be assigned to the target row.

Returns:

This function returns no value.

Definition at line 457 of file results.c.

References log_info.

Referenced by res_row_store().

bool_t res_row_store (uint64_t num, table_t * table, MYSQL_BIND * binding)

Definition at line 520 of file results.c.

References length, log_info, mm_alloc(), mm_copy(), res_field_count(), res_row_count(), res_row_get(), and res_row_set().

Referenced by res_stmt_store().

table_t* res_stmt_store (MYSQL_STMT * stmt)

Definition at line 589 of file results.c.

References log_info, mysql_stmt_bind_result_d, mysql_stmt_error_d, mysql_stmt_fetch_d, mysql_stmt_free_result_d, mysql_stmt_num_rows_d, mysql_stmt_store_result_d, res_bind_create(), res_bind_free(), res_row_store(), res_table_alloc(), and res_table_free().

Referenced by stmt_get_result_conn().

table_t* res_table_alloc (uint64_t rows, uint64_t fields)

Definition at line 324 of file results.c.

References log_info, and mm_alloc().

Referenced by res_stmt_store().

void res_table_free (table_t * table)

Free a mysql results table.

Parameters:

table the mysql results table to be freed.

Returns:

This function does not return a value.

Definition at line 258 of file results.c.

References log_info, and mm_free().

Referenced by config_fetch_host_number(), config_load_database_settings(), contact_details_fetch(), contacts_fetch(), magma_folder_fetch(), messages_fetch(), meta_data_acknowledge_alert(), meta_data_check_mailbox(), meta_data_fetch_alerts(), meta_data_fetch_all_tags(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_message_tags(), meta_data_fetch_messages(), meta_data_fetch_user(), meta_data_user_build(), meta_data_user_build_storage_keys(), register_data_check_username(), register_data_fetch_blocklist(), res_stmt_store(), smtp_check_authorized_from(), smtp_check_receive_quota(), and smtp_check_transmit_quota().

smtp_fetch_authorization(), smtp_fetch_autoreply(), smtp_fetch_inbound(), smtp_rollout(), statistics_refresh(), teacher_data_fetch(), user_config_fetch(), warehouse_fetch_domains(), and warehouse_fetch_patterns().

const char* serv_charset_mysql (void)

Definition at line 408 of file mysql.c.

References mm_wipe(), mysql_character_set_name_d, mysql_close_d, sql, and sql_open().

Referenced by lib_load().

const char* serv_schema_mysql (void)

Definition at line 424 of file mysql.c.

References mm_wipe(), mysql_close_d, sql, and sql_open().

Referenced by lib_load().

const char* serv_type_mysql (void)

Determine whether the mysql library in use is embedded or not.

Returns:

sql.type_embed ("Embedded") or **sql.type_serv** ("MySQL");

Definition at line 394 of file mysql.c.

References mysql_embedded_d, and sql.

Referenced by lib_load().

const char* serv_version_mysql (void)

Return the server version string of the mysql database.

Returns:

a pointer to a character string containing the mysql server version information.

Definition at line 444 of file mysql.c.

References mm_wipe(), mysql_close_d, mysql_get_server_info_d, sql, and sql_open().

Referenced by lib_load().

uint_t sql_errno (MYSQL * *mysql*)

Get the last error number for a mysql connection.

Parameters:

mysql a pointer to the MYSQL object of the connection to be queried.

Returns:

0 on failure, or the last mysql error message for the connection on success.

Definition at line 169 of file mysql.c.

References mysql_errno_d.

Referenced by stmt_close().

const chr_t* sql_error (MYSQL * *mysql*)

Get a human-readable error message for a mysql connection.

Parameters:

mysql a pointer to the MYSQL object of the connection to be queried.

Returns:

NULL on failure, or a pointer to a null-terminated string with the error message on success.

Definition at line 183 of file mysql.c.

References mysql_errno_d, and mysql_error_d.

Referenced by sql_open(), sql_ping(), sql_query_conn(), stmt_close(), stmt_open(), tran_commit(), tran_rollback(), and tran_start().

MYSQL* sql_open (bool_t *silent*)

Open up a new mysql connection to the magma-configured database server.

Note:

The reconnect option will automatically be set on all new mysql connections.

Parameters:

silent if true, suppress logging of failure messages for this function.

Returns:

NULL on failure, or a pointer to a MYSQL objection for the newly established connection on success.

Definition at line 222 of file mysql.c.

References magma_t::database, magma_t::iface, log_critical, magma, mysql_close_d, mysql_init_d, mysql_options_d, mysql_real_connect_d, and sql_error().

Referenced by serv_charset_mysql(), serv_schema_mysql(), serv_version_mysql(), and sql_start().

int_t sql_ping (uint32_t *connection*)

Definition at line 289 of file mysql.c.

References log_error, mysql_ping_d, mysql_thread_id_d, pool_get_obj(), and sql_error().

Referenced by dspam_check(), dspam_train(), and stmt_reset().

int64_t sql_query (stringer_t * *query*)

query.c

query.c

See also:

`sql_query_conn()`

Parameters:

query the mysql statement to be executed.

Returns:

0 on success, or non-zero on failure.

Definition at line 40 of file `query.c`.

References `log_info`, `PL_RESERVED`, `pool_pull()`, `pool_release()`, `sql_pool`, and `sql_query_conn()`.

int64_t sql_query_conn (stringer_t * *query*, uint32_t *connection*)

Execute a mysql statement.

See also:

`mysql_real_query()`

Parameters:

query the mysql statement to be executed.

connection a connection id for the underlying mysql session.

Returns:

0 on success, or a non-zero number on error.

Definition at line 22 of file `query.c`.

References `log_pedantic`, `mysql_real_query_d`, `pool_get_obj()`, `sql_error()`, `sql_pool`, `st_char_get()`, `st_data_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by `sql_query()`, `tran_commit()`, `tran_rollback()`, and `tran_start()`.

bool_t sql_start (void)

Load up the mysql subsystem.

Note:

This function will check that all necessary database parameters have been supplied, and that the supplied mysql library version is thread safe and initialized. It will also initialize the pool of database connections, and

Returns:

true on success or false on failure.

Definition at line 317 of file `mysql.c`.

References `magma_t::database`, `magma_t::iface`, `log_critical`, `magma`, `mysql_server_init_d`, `mysql_thread_safe_d`, `ns_empty()`, `pool_alloc()`, `pool_set_obj()`, `sql_open()`, `sql_stop()`, and `stmt_start()`.

Referenced by `process_start()`.

void sql_stop (void)

Shutdown and free the pool of mysql connections.

Returns:

This function returns no value.

Definition at line 195 of file mysql.c.

References magma_t::database, magma_t::iface, magma, my_once_free_d, mysql_close_d, mysql_server_end_d, pool_free(), pool_get_obj(), and stmt_stop().

Referenced by process_stop(), and sql_start().

bool_t sql_thread_start (void)

Initialize mysql thread specific variables for the calling thread.

Note:

mysql_thread_init()

Returns:

0 on success or non-zero on error.

Definition at line 381 of file mysql.c.

References log_error, and mysql_thread_init_d.

Referenced by thread_start().

void sql_thread_stop (void)

Prepare the thread for exiting to destroy mysql thread specific variables.

Note:

mysql_thread_end()

Returns:

This function returns no value.

Definition at line 371 of file mysql.c.

References mysql_thread_end_d.

Referenced by thread_stop().

bool_t stmt_bind_param (MYSQL_STMT * *group*, MYSQL_BIND * *bind*)**stmts.c**

Definition at line 256 of file stmts.c.

References log_critical, log_pedantic, mysql_stmt_bind_param_d, and stmt_error().

Referenced by stmt_exec_affected_conn(), stmt_exec_conn(), stmt_get_result_conn(), and stmt_insert_conn().

void stmt_close (MYSQL_STMT * *local*)

Definition at line 73 of file stmts.c.

References log_critical, mysql_stmt_close_d, sql_errno(), and sql_error().

Referenced by stmt_rebuild(), and stmt_stop().

uint_t stmt_errno (MYSQL_STMT * *local*)

Get the error number associated with a mysql statement.

Parameters:

local a pointer to the input mysql statement object.

Returns:

the errno associated with the specified mysql statement.

Definition at line 22 of file stmts.c.

References mysql_errno_d, and mysql_stmt_errno_d.

const chr_t* stmt_error (MYSQL_STMT * *local*)

Return the error string for a mysql statement.

Parameters:

local A single MYSQL_STMT* parameter.

Returns:

This function returns the mysql error string, or NULL if unable to retrieve it.

Definition at line 39 of file stmts.c.

References mysql_errno_d, mysql_error_d, mysql_stmt_errno_d, and mysql_stmt_error_d.

Referenced by stmt_bind_param(), stmt_exec_affected_conn(), stmt_exec_conn(), stmt_get_result_conn(), stmt_insert_conn(), and stmt_prepare().

bool_t stmt_exec (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*)

Execute a prepared mysql statement.

See also:

stmt_exec_conn()

Parameters:

group the mysql prepared statement to be executed.

parameters the parameters to be bound to the statement.

Returns:

false on failure, or true on success.

Definition at line 312 of file stmts.c.

References log_info, PL_RESERVED, pool_pull(), pool_release(), sql_pool, and stmt_exec_conn().

Referenced by mail_db_hide_message(), meta_data_flags_add(), meta_data_flags_remove(), meta_data_flags_replace(), smtp_update_receive_stats(), smtp_update_transmission_stats(), and teacher_data_delete().

uint64_t stmt_exec_affected (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*)

Execute a statement and return the number of affected rows.

Note:

From the MySQL documentation (section 20.9.3.1): "Because `mysql_affected_rows()` returns an unsigned value, you can check for -1 by comparing the return value to `(my_ulonglong)-1` (or to `(my_ulonglong)~0`, which is equivalent)."

See also:

`mysql_stmt_affected_rows()`

Parameters:

group the prepared mysql statement to be executed.

parameters the parameters to be bound to the prepared statement.

Returns:

-1 on failure, 0 on failure or no rows affected, and a positive number on success.

Definition at line 473 of file `stmts.c`.

References `log_info`, `PL_RESERVED`, `pool_pull()`, `pool_release()`, `sql_pool`, and `stmt_exec_affected_conn()`.

Referenced by `contact_delete()`, `contact_detail_delete()`, `contact_detail_upsert()`, `contact_update()`, `contact_update_stamp()`, `magma_folder_delete()`, `magma_folder_rename()`, `meta_data_delete_folder()`, `meta_data_delete_tag()`, `meta_data_insert_tag()`, `meta_data_truncate_tags()`, `meta_data_update_folder_name()`, `meta_data_update_lock()`, `meta_data_update_log()`, `meta_data_user_save_storage_keys()`, `user_config_delete()`, and `user_config_upsert()`.

**uint64_t stmt_exec_affected_conn (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*,
uint32_t *connection*)**

Execute a statement on a specified mysql connection and return the number of affected rows.

Note:

From the MySQL documentation (section 20.9.3.1): "Because `mysql_affected_rows()` returns an unsigned value, you can check for -1 by comparing the return value to `(my_ulonglong)-1` (or to `(my_ulonglong)~0`, which is equivalent)."

See also:

`mysql_stmt_affected_rows()`

Parameters:

group the prepared mysql statement to be executed.

parameters the parameters to be bound to the prepared statement.

connection the mysql connection id over which the specified statement will be executed.

Returns:

-1 on failure, 0 on failure or no rows affected, and a positive number on success.

Definition at line 441 of file `stmts.c`.

References `log_info`, `mysql_stmt_affected_rows_d`, `mysql_stmt_execute_d`, `stmt_bind_param()`, `stmt_error()`, and `stmt_reset()`.

Referenced by `mail_db_delete_message()`, `mail_db_update_message_folder()`, `meta_data_acknowledge_alert()`, `meta_data_user_save_storage_keys()`, `stmt_exec_affected()`, and `tank_delete_object()`.

**bool_t stmt_exec_conn (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*, uint32_t
connection)**

Execute a prepared mysql statement over a specified connection.

Note:

This function will also reset the prepared statement and bind its parameters.

Parameters:

group the mysql prepared statement to be executed.
parameters the parameters to be bound to the statement.
connection the mysql connection id.

Returns:

false on failure, or true on success.

Definition at line 283 of file stmts.c.

References log_info, mysql_stmt_execute_d, stmt_bind_param(), stmt_error(), and stmt_reset().

Referenced by mail_db_insert_duplicate_message(), mail_db_insert_message(), register_data_insert_user(), and stmt_exec().

table_t* stmt_get_result (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*)

Execute a prepared mysql statement and return the result.

Parameters:

group the prepared mysql statement to be executed.
parameters the parameters to be passed with the query.

Returns:

the result of the query, or NULL on failure.

Definition at line 363 of file stmts.c.

References log_info, PL_RESERVED, pool_pull(), pool_release(), sql_pool, and stmt_get_result_conn().

Referenced by config_fetch_host_number(), config_fetch_settings(), contact_details_fetch(), contacts_fetch(), magma_folder_fetch(), messages_fetch(), meta_data_check_mailbox(), meta_data_fetch_alerts(), meta_data_fetch_all_tags(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_message_tags(), meta_data_fetch_messages(), meta_data_fetch_user(), meta_data_user_build(), meta_data_user_build_storage_keys(), register_data_check_username(), register_data_fetch_blocklist(), smtp_check_authorized_from(), smtp_check_receive_quota(), smtp_check_transmit_quota(), smtp_fetch_authorization(), smtp_fetch_autoreply(), smtp_fetch_inbound(), smtp_fetch_rollmessages(), statistics_refresh(), teacher_data_fetch(), user_config_fetch(), warehouse_fetch_domains(), and warehouse_fetch_patterns().

table_t* stmt_get_result_conn (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*, uint32_t *connection*)

Execute a prepared mysql statement on a specified connection and return the result.

Parameters:

group the prepared mysql statement to be executed.
parameters the parameters to be passed with the query.
connection the mysql connection identifier.

Returns:

the result of the query, or NULL on failure.

Definition at line 335 of file stmts.c.

References `log_info`, `mysql_stmt_execute_d`, `res_stmt_store()`, `stmt_bind_param()`, `stmt_error()`, and `stmt_reset()`.

Referenced by `meta_data_acknowledge_alert()`, `meta_data_user_build_storage_keys()`, and `stmt_get_result()`.

uint64_t stmt_insert (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*)

Execute a mysql prepared INSERT or UPDATE statement.

See also:

`mysql_stmt_insert_id()`

Parameters:

group the mysql prepared statement to be executed.

parameters the parameters to be bound to the prepared statement.

Returns:

0 on failure, or the last `LAST_INSERT_ID()` value returned for the mysql auto-increment column.

Definition at line 415 of file `stmts.c`.

References `log_info`, `PL_RESERVED`, `pool_pull()`, `pool_release()`, `sql_pool`, and `stmt_insert_conn()`.

Referenced by `contact_insert()`, `magma_folder_insert()`, `meta_data_insert_folder()`, and `smtp_insert_spamsig()`.

uint64_t stmt_insert_conn (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*, uint32_t *connection*)

Execute a mysql prepared INSERT or UPDATE statement on a specified connection.

See also:

`mysql_stmt_insert_id()`

Parameters:

group the mysql prepared statement to be executed.

parameters the parameters to be bound to the prepared statement.

connection the underlying connection on which the statement will be executed.

Returns:

0 on failure, or the last `LAST_INSERT_ID()` value returned for the mysql auto-increment column.

Definition at line 386 of file `stmts.c`.

References `log_info`, `mysql_stmt_execute_d`, `mysql_stmt_insert_id_d`, `stmt_bind_param()`, `stmt_error()`, and `stmt_reset()`.

Referenced by `mail_db_insert_duplicate_message()`, `mail_db_insert_message()`, `register_data_insert_user()`, `stmt_insert()`, and `tank_insert_object()`.

MYSQL_STMT* stmt_open (MYSQL * *mysql*)

Initialize a mysql prepared statement.

See also:

`mysql_stmt_init()`

Parameters:

mysql the underlying mysql connection.

Returns:

This function is a thin wrapper around the mysql library function **mysql_stmt_init_d()**.

Definition at line 57 of file stmts.c.

References log_critical, mysql_stmt_init_d, and sql_error().

Referenced by stmt_rebuild(), and stmt_start().

bool_t stmt_prepare (MYSQL_STMT * *group*, const char * *query*, unsigned long *length*)

Definition at line 121 of file stmts.c.

References log_critical, mysql_stmt_attr_set_d, mysql_stmt_prepare_d, and stmt_error().

Referenced by stmt_rebuild(), and stmt_start().

bool_t stmt_rebuild (uint32_t *connection*)

Definition at line 195 of file stmts.c.

References log_critical, ns_length_get(), pool_get_obj(), queries, sql_pool, stmt_close(), stmt_open(), and stmt_prepare().

Referenced by dspam_check(), dspam_train(), and stmt_reset().

MYSQL_STMT* stmt_reset (MYSQL_STMT ** *group*, uint32_t *connection*)

Definition at line 227 of file stmts.c.

References log_critical, log_pedantic, mysql_stmt_reset_d, sql_ping(), and stmt_rebuild().

Referenced by stmt_exec_affected_conn(), stmt_exec_conn(), stmt_get_result_conn(), and stmt_insert_conn().

bool_t stmt_start (void)

Initialize the global array of mysql prepared statements.

Note:

This function readies a copy of each prepared statement for every member of the global mysql connection pool.

Returns:

true on success or false on failure.

Definition at line 148 of file stmts.c.

References magma_t::database, magma_t::iface, log_critical, magma, mm_alloc(), mm_wipe(), ns_length_get(), pool_get_obj(), queries, sql_pool, stmt_open(), stmt_prepare(), and stmt_stop().

Referenced by sql_start().

void stmt_stop (void)

Definition at line 95 of file stmts.c.

References magma_t::database, magma_t::iface, magma, mm_free(), queries, and stmt_close().

Referenced by sql_stop(), and stmt_start().

int64_t tran_commit (int64_t *transaction*)

transaction.c

transaction.c

Parameters:

transaction the mysql connection identifier.

Returns:

0 on success, or a non-zero number on error.

Definition at line 77 of file transaction.c.

References command, length, log_info, PLACER, pool_get_obj(), pool_release(), sql_error(), sql_pool, sql_query_conn(), and tran_commands.

Referenced by adjust_message_encryption(), mail_copy_message(), mail_move_message(), mail_remove_message(), mail_store_message(), portal_endpoint_alert_acknowledge(), register_business_step2(), tank_delete(), and tank_store().

int64_t tran_rollback (int64_t *transaction*)

Rollback a pending transaction in the mysql database.

Parameters:

transaction the mysql connection identifier.

Returns:

0 on success, or a non-zero number on error.

Definition at line 57 of file transaction.c.

References command, length, log_info, PLACER, pool_get_obj(), pool_release(), sql_error(), sql_pool, sql_query_conn(), and tran_commands.

Referenced by adjust_message_encryption(), mail_copy_message(), mail_move_message(), mail_remove_message(), mail_store_message(), portal_endpoint_alert_acknowledge(), register_business_step2(), tank_delete(), and tank_store().

int64_t tran_start (void)

Pull a connection from the sql pool and start a transaction.

Returns:

-1 on failure, or a mysql connection id from the sql pool on success.

Definition at line 31 of file transaction.c.

References command, length, log_info, PL_RESERVED, PLACER, pool_get_obj(), pool_pull(), pool_release(), sql_error(), sql_pool, sql_query_conn(), and tran_commands.

Referenced by adjust_message_encryption(), mail_copy_message(), mail_move_message(), mail_remove_message(), mail_store_message(), portal_endpoint_alert_acknowledge(), register_business_step2(), tank_delete(), and tank_store().

Variable Documentation

pool_t* sql_pool

Definition at line 149 of file mysql.c.

magma/providers/database/mysql.c File Reference

MYSQL Symbols.

```
#include "magma.h"
```

Functions

- **uint_t sql_errno** (MYSQL *mysql)
- *Get the last error number for a mysql connection.* **const chr_t * sql_error** (MYSQL *mysql)
- *Get a human-readable error message for a mysql connection.* **void sql_stop** (void)
- *Shutdown and free the pool of mysql connections.* **MYSQL * sql_open** (**bool_t** silent)
- *Open up a new mysql connection to the magma-configured database server.* **int_t sql_ping** (uint32_t connection)
- **bool_t sql_start** (void)
- *Load up the mysql subsystem.* **void sql_thread_stop** (void)
- *Prepare the thread for exiting to destroy mysql thread specific variables.* **bool_t sql_thread_start** (void)
- *Initialize mysql thread specific variables for the calling thread.* **const char * serv_type_mysql** (void)
- *Determine whether the mysql library in use is embedded or not.* **const char * serv_charset_mysql** (void)
- **const char * serv_schema_mysql** (void)
- **const char * serv_version_mysql** (void)
- *Return the server version string of the mysql database.* **const char * lib_version_mysql** (void)
- *Return the version string of libmysql.* **bool_t lib_load_mysql** (void)

Initialize the libmysql and bind dynamically to the exported functions that are required. Variables

- **pool_t * sql_pool** = NULL
- **struct {**
 - **char lib_version** [16]
 - **char serv_version** [16]
 - **char serv_charset** [16]
 - **char serv_schema** [32]
 - **const chr_t * type_serv**
 - **const chr_t * type_embed**
 - **const chr_t * dash**
- **} sql**

Detailed Description

MYSQL Symbols.

Definition in file **mysql.c**.

Function Documentation

bool_t lib_load_mysql (void)

Initialize the libmysql and bind dynamically to the exported functions that are required.

mysql.c

Returns:

true on success or false on failure.

Definition at line 487 of file mysql.c.

References lib_symbols(), M_BIND, and mysql_character_set_name().

Referenced by lib_load().

const char* lib_version_mysql (void)

Return the version string of libmysql.

Returns:

a pointer to a character string containing the libmysql version information.

Definition at line 464 of file mysql.c.

References mysql_get_client_version_d, and sql.

Referenced by lib_load().

const char* serv_charset_mysql (void)

Definition at line 408 of file mysql.c.

References mm_wipe(), mysql_character_set_name_d, mysql_close_d, sql, and sql_open().

Referenced by lib_load().

const char* serv_schema_mysql (void)

Definition at line 424 of file mysql.c.

References mm_wipe(), mysql_close_d, sql, and sql_open().

Referenced by lib_load().

const char* serv_type_mysql (void)

Determine whether the mysql library in use is embedded or not.

Returns:

sql.type_embed ("Embedded") or **sql.type_serv** ("MySQL");

Definition at line 394 of file mysql.c.

References mysql_embedded_d, and sql.

Referenced by lib_load().

const char* serv_version_mysql (void)

Return the server version string of the mysql database.

Returns:

a pointer to a character string containing the mysql server version information.
Definition at line 444 of file mysql.c.
References mm_wipe(), mysql_close_d, mysql_get_server_info_d, sql, and sql_open().
Referenced by lib_load().

uint_t sql_errno (MYSQL * *mysql*)

Get the last error number for a mysql connection.

Parameters:

mysql a pointer to the MYSQL object of the connection to be queried.

Returns:

0 on failure, or the last mysql error message for the connection on success.
Definition at line 169 of file mysql.c.
References mysql_errno_d.
Referenced by stmt_close().

const chr_t* sql_error (MYSQL * *mysql*)

Get a human-readable error message for a mysql connection.

Parameters:

mysql a pointer to the MYSQL object of the connection to be queried.

Returns:

NULL on failure, or a pointer to a null-terminated string with the error message on success.
Definition at line 183 of file mysql.c.
References mysql_errno_d, and mysql_error_d.
Referenced by sql_open(), sql_ping(), sql_query_conn(), stmt_close(), stmt_open(), tran_commit(), tran_rollback(), and tran_start().

MYSQL* sql_open (bool_t *silent*)

Open up a new mysql connection to the magma-configured database server.

Note:

The reconnect option will automatically be set on all new mysql connections.

Parameters:

silent if true, suppress logging of failure messages for this function.

Returns:

NULL on failure, or a pointer to a MYSQL objection for the newly established connection on success.
Definition at line 222 of file mysql.c.

References magma_t::database, magma_t::iface, log_critical, magma, mysql_close_d, mysql_init_d, mysql_options_d, mysql_real_connect_d, and sql_error().

Referenced by serv_charset_mysql(), serv_schema_mysql(), serv_version_mysql(), and sql_start().

int_t sql_ping (uint32_t *connection*)

Definition at line 289 of file mysql.c.

References log_error, mysql_ping_d, mysql_thread_id_d, pool_get_obj(), and sql_error().

Referenced by dspam_check(), dspam_train(), and stmt_reset().

bool_t sql_start (void)

Load up the mysql subsystem.

Note:

This function will check that all necessary database parameters have been supplied, and that the supplied mysql library version is thread safe and initialized. It will also initialize the pool of database connections, and

Returns:

true on success or false on failure.

Definition at line 317 of file mysql.c.

References magma_t::database, magma_t::iface, log_critical, magma, mysql_server_init_d, mysql_thread_safe_d, ns_empty(), pool_alloc(), pool_set_obj(), sql_open(), sql_stop(), and stmt_start().

Referenced by process_start().

void sql_stop (void)

Shutdown and free the pool of mysql connections.

Returns:

This function returns no value.

Definition at line 195 of file mysql.c.

References magma_t::database, magma_t::iface, magma, my_once_free_d, mysql_close_d, mysql_server_end_d, pool_free(), pool_get_obj(), and stmt_stop().

Referenced by process_stop(), and sql_start().

bool_t sql_thread_start (void)

Initialize mysql thread specific variables for the calling thread.

Note:

mysql_thread_init()

Returns:

0 on success or non-zero on error.

Definition at line 381 of file mysql.c.

References log_error, and mysql_thread_init_d.

Referenced by thread_start().

void sql_thread_stop (void)

Prepare the thread for exiting to destroy mysql thread specific variables.

Note:

mysql_thread_end()

Returns:

This function returns no value.

Definition at line 371 of file mysql.c.

References mysql_thread_end_d.

Referenced by thread_stop().

Variable Documentation

const chr_t * dash

Definition at line 157 of file mysql.c.

char lib_version[16]

Definition at line 152 of file mysql.c.

char serv_charset[16]

Definition at line 154 of file mysql.c.

char serv_schema[32]

Definition at line 155 of file mysql.c.

char serv_version[16]

Definition at line 153 of file mysql.c.

struct { ... } sql

Referenced by lib_version_mysql(), serv_charset_mysql(), serv_schema_mysql(), serv_type_mysql(), and serv_version_mysql().

pool_t* sql_pool = NULL

Definition at line 149 of file mysql.c.

Referenced by mail_db_update_message_folder(), sql_query(), sql_query_conn(), stmt_exec(), stmt_exec_affected(), stmt_get_result(), stmt_insert(), stmt_rebuild(), stmt_start(), tran_commit(), tran_rollback(), and tran_start().

const chr_t * type_embed

Definition at line 157 of file mysql.c.

const chr_t* type_serv

Definition at line 157 of file mysql.c.

magma/providers/database/query.c File Reference

Traditional SQL queries in string form.

```
#include "magma.h"
```

Functions

- `int64_t sql_query_conn (stringer_t *query, uint32_t connection)`
- *Execute a mysql statement.* `int64_t sql_query (stringer_t *query)`

Pull a connection from the sql pool and execute a mysql statement.

Detailed Description

Traditional SQL queries in string form.

Definition in file `query.c`.

Function Documentation

`int64_t sql_query (stringer_t * query)`

Pull a connection from the sql pool and execute a mysql statement.

query.c

See also:

`sql_query_conn()`

Parameters:

query the mysql statement to be executed.

Returns:

0 on success, or non-zero on failure.

Definition at line 40 of file `query.c`.

References `log_info`, `PL_RESERVED`, `pool_pull()`, `pool_release()`, `sql_pool`, and `sql_query_conn()`.

`int64_t sql_query_conn (stringer_t * query, uint32_t connection)`

Execute a mysql statement.

See also:

`mysql_real_query()`

Parameters:

query the mysql statement to be executed.

connection a connection id for the underlying mysql session.

Returns:

0 on success, or a non-zero number on error.

Definition at line 22 of file `query.c`.

References `log_pedantic`, `mysql_real_query_d`, `pool_get_obj()`, `sql_error()`, `sql_pool`, `st_char_get()`, `st_data_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by `sql_query()`, `tran_commit()`, `tran_rollback()`, and `tran_start()`.

magma/providers/database/results.c File Reference

A collection of functions for handling MySQL result sets.

```
#include "magma.h"
```

Functions

- **row_t * res_field_generic** (row_t *row, uint64_t field, size_t typesize)
- **void * res_field_block** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as generic pointer.* **stringer_t * res_field_string** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as managed string.* **double_t res_field_double** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as a double.* **float_t res_field_float** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as a float.* **uint64_t res_field_uint64** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as an unsigned 64-bit integer.* **uint32_t res_field_uint32** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as an unsigned 32-bit integer.* **uint16_t res_field_uint16** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as an unsigned 16-bit integer.* **uint8_t res_field_uint8** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as an unsigned 8-bit integer.* **int64_t res_field_int64** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as a signed 64-bit integer.* **int32_t res_field_int32** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as a signed 32-bit integer.* **int16_t res_field_int16** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as a signed 16-bit integer.* **int8_t res_field_int8** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as a signed 8-bit integer.* **bool_t res_field_bool** (row_t *row, uint64_t field)
- *Get the value of a specified field in a database result row as a boolean.* **size_t res_field_length** (row_t *row, uint64_t field)
- *Get the length of a specified field in a database result row.* **void res_table_free** (table_t *table)
- *Free a mysql results table.* **void res_bind_free** (MYSQL_STMT *stmt, MYSQL_BIND *binding, uint64_t number)
- *Free a prepared mysql statement result set binding.* **table_t * res_table_alloc** (uint64_t rows, uint64_t fields)
- **uint64_t res_bind_create** (MYSQL_STMT *stmt, MYSQL_BIND **result)
- **uint64_t res_field_count** (table_t *table)
- *Return the number of fields stored in the database results table.* **uint64_t res_row_count** (table_t *table)
- *Return the number of rows in the mysql results table.* **void res_row_set** (row_t *row, chr_t *buffer)
- *Set the buffer location for a database result row.* **row_t * res_row_get** (table_t *table, uint64_t row)
- *Retrieve a row from a database results table by index.* **row_t * res_row_next** (table_t *table)
- *Return the next row in the database results table and advance the cursor.* **bool_t res_row_store** (uint64_t num, table_t *table, MYSQL_BIND *binding)
- **table_t * res_stmt_store** (MYSQL_STMT *stmt)

Detailed Description

A collection of functions for handling MySQL result sets.

Definition in file **results.c**.

Function Documentation

uint64_t res_bind_create (MYSQL_STMT * *stmt*, MYSQL_BIND ** *result*)

Definition at line 356 of file results.c.

References `length`, `log_info`, `mm_alloc()`, `mysql_fetch_field_d`, `mysql_free_result_d`, `mysql_num_fields_d`, `mysql_stmt_error_d`, `mysql_stmt_result_metadata_d`, and `res_bind_free()`.

Referenced by `res_stmt_store()`.

void res_bind_free (MYSQL_STMT * *stmt*, MYSQL_BIND * *binding*, uint64_t *number*)

Free a prepared mysql statement result set binding.

Parameters:

stmt the input prepared mysql statement.

binding the binding for the result set.

number the number of fields in the result set.

Returns:

This function does not return a value.

Definition at line 298 of file results.c.

References `length`, `mm_cleanup()`, `mm_free()`, and `mysql_stmt_bind_result_d`.

Referenced by `res_bind_create()`, and `res_stmt_store()`.

void* res_field_block (row_t * *row*, uint64_t *field*)

Get the value of a specified field in a database result row as generic pointer.

Parameters:

row a pointer to the database result row to be examined.

field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a generic pointer, or NULL on failure.

Definition at line 51 of file results.c.

References `res_field_generic()`.

Referenced by `config_load_database_settings()`, `contact_details_fetch()`, `contacts_fetch()`, `magma_folder_fetch()`, `messages_fetch()`, `meta_data_fetch_alerts()`, `meta_data_fetch_folders()`, `meta_data_fetch_mailbox_aliases()`, `meta_data_fetch_messages()`, `meta_data_user_build_storage_keys()`, `smtp_check_receive_quota()`, `smtp_fetch_inbound()`, `user_config_fetch()`, and `warehouse_fetch_domains()`.

bool_t res_field_bool (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as a boolean.

results.c

Parameters:

row a pointer to the database result row to be examined.

field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a boolean, or false on failure.

Definition at line 224 of file results.c.

References res_field_generic().

uint64_t res_field_count (table_t * table)

Return the number of fields stored in the database results table.

Parameters:

table the input database results table.

Returns:

0 on failure, or the number of table fields on success.

Definition at line 423 of file results.c.

References log_info.

Referenced by res_row_store().

double_t res_field_double (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as a double.

Parameters:

row a pointer to the database result row to be examined.

field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a double, or 0 on failure.

Definition at line 94 of file results.c.

References res_field_generic().

float_t res_field_float (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as a float.

Parameters:

row a pointer to the database result row to be examined.

field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a float, or 0 on failure.

Definition at line 107 of file results.c.

References res_field_generic().

row_t* res_field_generic (row_t * row, uint64_t field, size_t typesize)

Definition at line 17 of file results.c.

References debug_hook(), and log_info.

Referenced by res_field_block(), res_field_bool(), res_field_double(), res_field_float(), res_field_int16(), res_field_int32(), res_field_int64(), res_field_int8(), res_field_uint16(), res_field_uint32(), res_field_uint64(), and res_field_uint8().

int16_t res_field_int16 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as a signed 16-bit integer.

Parameters:

row a pointer to the database result row to be examined.

field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a signed 16-bit integer, or 0 on failure.

Definition at line 198 of file results.c.

References res_field_generic().

int32_t res_field_int32 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as a signed 32-bit integer.

Parameters:

row a pointer to the database result row to be examined.

field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a signed 32-bit integer, or 0 on failure.

Definition at line 185 of file results.c.

References res_field_generic().

int64_t res_field_int64 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as a signed 64-bit integer.

Parameters:

row a pointer to the database result row to be examined.

field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a signed 64-bit integer, or 0 on failure.

Definition at line 172 of file results.c.

References res_field_generic().

int8_t res_field_int8 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as a signed 8-bit integer.

Parameters:

row a pointer to the database result row to be examined.

field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a signed 8-bit integer, or 0 on failure.

Definition at line 211 of file results.c.

References res_field_generic().

Referenced by meta_data_fetch_user(), meta_data_user_build(), smtp_fetch_authorization(), smtp_fetch_inbound(), teacher_data_fetch(), and warehouse_fetch_domains().

size_t res_field_length (row_t * row, uint64_t field)

Get the length of a specified field in a database result row.

Parameters:

row a pointer to the database result row to be examined.

field the zero-based index of the field in the row to be queried.

Returns:

the length of the specified result row, or 0 on failure.

Definition at line 237 of file results.c.

References log_info.

Referenced by config_load_database_settings(), contact_details_fetch(), contacts_fetch(), magma_folder_fetch(), messages_fetch(), meta_data_fetch_alerts(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_messages(), meta_data_user_build_storage_keys(), smtp_check_receive_quota(), smtp_fetch_inbound(), user_config_fetch(), and warehouse_fetch_domains().

stringer_t* res_field_string (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as managed string.

Parameters:

row a pointer to the database result row to be examined.

field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as a managed string, or NULL on failure.

Definition at line 64 of file results.c.

References `length`, `log_info`, and `st_import()`.

Referenced by `meta_data_fetch_all_tags()`, `meta_data_fetch_message_tags()`, `meta_data_fetch_user()`, `register_data_fetch_blocklist()`, `smtp_fetch_authorization()`, `smtp_fetch_autoreply()`, `smtp_fetch_inbound()`, `smtp_rollout()`, `teacher_data_fetch()`, and `warehouse_fetch_patterns()`.

uint16_t res_field_uint16 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as an unsigned 16-bit integer.

Parameters:

row a pointer to the database result row to be examined.

field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as an unsigned 16-bit integer, or 0 on failure.

Definition at line 146 of file `results.c`.

References `res_field_generic()`.

uint32_t res_field_uint32 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as an unsigned 32-bit integer.

Parameters:

row a pointer to the database result row to be examined.

field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as an unsigned 32-bit integer, or 0 on failure.

Definition at line 133 of file `results.c`.

References `res_field_generic()`.

Referenced by `magma_folder_fetch()`, `messages_fetch()`, `meta_data_fetch_folders()`, `meta_data_fetch_messages()`, `smtp_fetch_authorization()`, `smtp_fetch_inbound()`, and `smtp_rollout()`.

uint64_t res_field_uint64 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as an unsigned 64-bit integer.

Parameters:

row a pointer to the database result row to be examined.

field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as an unsigned 64-bit integer, or 0 on failure.

Definition at line 120 of file `results.c`.

References `res_field_generic()`.

Referenced by `config_fetch_host_number()`, `contact_details_fetch()`, `contacts_fetch()`, `magma_folder_fetch()`, `messages_fetch()`, `meta_data_acknowledge_alert()`, `meta_data_fetch_alerts()`, `meta_data_fetch_folders()`,

meta_data_fetch_mailbox_aliases(), meta_data_fetch_messages(), meta_data_user_build(),
smtp_check_receive_quota(), smtp_check_transmit_quota(), smtp_fetch_authorization(),
smtp_fetch_inbound(), smtp_rollout(), statistics_refresh(), teacher_data_fetch(), and user_config_fetch().

uint8_t res_field_uint8 (row_t * row, uint64_t field)

Get the value of a specified field in a database result row as an unsigned 8-bit integer.

Parameters:

row a pointer to the database result row to be examined.
field the zero-based index of the field in the row to be queried.

Returns:

the value of the specified result row as an unsigned 8-bit integer, or 0 on failure.
Definition at line 159 of file results.c.
References res_field_generic().
Referenced by meta_data_fetch_mailbox_aliases().

uint64_t res_row_count (table_t * table)

Return the number of rows in the mysql results table.

Note:

The row count is provided for a one-indexed table.

Parameters:

table the input mysql results table.

Returns:

0 on error, or the row count on success.
Definition at line 440 of file results.c.
References log_info.
Referenced by config_load_database_settings(), meta_data_fetch_message_tags(), res_row_store(),
smtp_check_authorized_from(), and smtp_fetch_inbound().

row_t* res_row_get (table_t * table, uint64_t row)

Retrieve a row from a database results table by index.

Note:

This function operates on a 0-indexed table.

Parameters:

table the input database results table. *row* the row # to be fetched.

Returns:

NULL on failure, or a pointer to a result row on success.
Definition at line 477 of file results.c.
References log_info.

Referenced by config_load_database_settings(), res_row_next(), and res_row_store().

row_t* res_row_next (table_t * table)

Return the next row in the database results table and advance the cursor.

Parameters:

table the input mysql results table.

Returns:

a pointer to the next result row in the table.

Definition at line 500 of file results.c.

References count, and res_row_get().

Referenced by config_fetch_host_number(), contact_details_fetch(), contacts_fetch(), magma_folder_fetch(), messages_fetch(), meta_data_acknowledge_alert(), meta_data_check_mailbox(), meta_data_fetch_alerts(), meta_data_fetch_all_tags(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_message_tags(), meta_data_fetch_messages(), meta_data_fetch_user(), meta_data_user_build(), meta_data_user_build_storage_keys(), register_data_check_username(), register_data_fetch_blocklist(), smtp_check_receive_quota(), smtp_check_transmit_quota(), smtp_fetch_authorization(), smtp_fetch_autoreply(), smtp_fetch_inbound(), smtp_rollout(), statistics_refresh(), teacher_data_fetch(), user_config_fetch(), warehouse_fetch_domains(), and warehouse_fetch_patterns().

void res_row_set (row_t * row, chr_t * buffer)

Set the buffer location for a database result row.

Parameters:

row the input database result row.

buffer the buffer to be assigned to the target row.

Returns:

This function returns no value.

Definition at line 457 of file results.c.

References log_info.

Referenced by res_row_store().

bool_t res_row_store (uint64_t num, table_t * table, MYSQL_BIND * binding)

Definition at line 520 of file results.c.

References length, log_info, mm_alloc(), mm_copy(), res_field_count(), res_row_count(), res_row_get(), and res_row_set().

Referenced by res_stmt_store().

table_t* res_stmt_store (MYSQL_STMT * stmt)

Definition at line 589 of file results.c.

References `log_info`, `mysql_stmt_bind_result_d`, `mysql_stmt_error_d`, `mysql_stmt_fetch_d`, `mysql_stmt_free_result_d`, `mysql_stmt_num_rows_d`, `mysql_stmt_store_result_d`, `res_bind_create()`, `res_bind_free()`, `res_row_store()`, `res_table_alloc()`, and `res_table_free()`.

Referenced by `stmt_get_result_conn()`.

table_t* res_table_alloc (uint64_t rows, uint64_t fields)

Definition at line 324 of file `results.c`.

References `log_info`, and `mm_alloc()`.

Referenced by `res_stmt_store()`.

void res_table_free (table_t * table)

Free a mysql results table.

Parameters:

table the mysql results table to be freed.

Returns:

This function does not return a value.

Definition at line 258 of file `results.c`.

References `log_info`, and `mm_free()`.

Referenced by `config_fetch_host_number()`, `config_load_database_settings()`, `contact_details_fetch()`, `contacts_fetch()`, `magma_folder_fetch()`, `messages_fetch()`, `meta_data_acknowledge_alert()`, `meta_data_check_mailbox()`, `meta_data_fetch_alerts()`, `meta_data_fetch_all_tags()`, `meta_data_fetch_folders()`, `meta_data_fetch_mailbox_aliases()`, `meta_data_fetch_message_tags()`, `meta_data_fetch_messages()`, `meta_data_fetch_user()`, `meta_data_user_build()`, `meta_data_user_build_storage_keys()`, `register_data_check_username()`, `register_data_fetch_blocklist()`, `res_stmt_store()`, `smtp_check_authorized_from()`, `smtp_check_receive_quota()`, `smtp_check_transmit_quota()`, `smtp_fetch_authorization()`, `smtp_fetch_autoreply()`, `smtp_fetch_inbound()`, `smtp_rollout()`, `statistics_refresh()`, `teacher_data_fetch()`, `user_config_fetch()`, `warehouse_fetch_domains()`, and `warehouse_fetch_patterns()`.

magma/providers/database/stmts.c File Reference

A collection of functions for working with the MySQL prepared statement interface.

```
#include "magma.h"
```

Functions

- **uint_t stmt_errno** (MYSQL_STMT *local)
- *Get the error number associated with a mysql statement.* const **chr_t * stmt_error** (MYSQL_STMT *local)
- **MYSQL_STMT * stmt_open** (MYSQL *mysql)
- *Initialize a mysql prepared statement.* void **stmt_close** (MYSQL_STMT *local)
- void **stmt_stop** (void)
- **bool_t stmt_prepare** (MYSQL_STMT *group, const char *query, unsigned long **length**)
- **bool_t stmt_start** (void)
- *Initialize the global array of mysql prepared statements.* **bool_t stmt_rebuild** (uint32_t connection)
- **MYSQL_STMT * stmt_reset** (MYSQL_STMT **group, uint32_t connection)
- **bool_t stmt_bind_param** (MYSQL_STMT *group, MYSQL_BIND *bind)
- *stmts.c* **bool_t stmt_exec_conn** (MYSQL_STMT **group, MYSQL_BIND *parameters, uint32_t connection)
- *Execute a prepared mysql statement over a specified connection.* **bool_t stmt_exec** (MYSQL_STMT **group, MYSQL_BIND *parameters)
- *Execute a prepared mysql statement.* **table_t * stmt_get_result_conn** (MYSQL_STMT **group, MYSQL_BIND *parameters, uint32_t connection)
- *Execute a prepared mysql statement on a specified connection and return the result.* **table_t * stmt_get_result** (MYSQL_STMT **group, MYSQL_BIND *parameters)
- *Execute a prepared mysql statement and return the result.* uint64_t **stmt_insert_conn** (MYSQL_STMT **group, MYSQL_BIND *parameters, uint32_t connection)
- *Execute a mysql prepared INSERT or UPDATE statement on a specified connection.* uint64_t **stmt_insert** (MYSQL_STMT **group, MYSQL_BIND *parameters)
- *Execute a mysql prepared INSERT or UPDATE statement.* uint64_t **stmt_exec_affected_conn** (MYSQL_STMT **group, MYSQL_BIND *parameters, uint32_t connection)
- *Execute a statement on a specified mysql connection and return the number of affected rows.* uint64_t **stmt_exec_affected** (MYSQL_STMT **group, MYSQL_BIND *parameters)

Execute a statement and return the number of affected rows. Variables

- **chr_t * queries []** = { QUERIES_INIT }

Detailed Description

A collection of functions for working with the MySQL prepared statement interface.

Definition in file **stmts.c**.

Function Documentation

bool_t stmt_bind_param (MYSQL_STMT * *group*, MYSQL_BIND * *bind*)

stmts.c

Definition at line 256 of file stmts.c.

References log_critical, log_pedantic, mysql_stmt_bind_param_d, and stmt_error().

Referenced by stmt_exec_affected_conn(), stmt_exec_conn(), stmt_get_result_conn(), and stmt_insert_conn().

void stmt_close (MYSQL_STMT * *local*)

Definition at line 73 of file stmts.c.

References log_critical, mysql_stmt_close_d, sql_errno(), and sql_error().

Referenced by stmt_rebuild(), and stmt_stop().

uint_t stmt_errno (MYSQL_STMT * *local*)

Get the error number associated with a mysql statement.

Parameters:

local a pointer to the input mysql statement object.

Returns:

the errno associated with the specified mysql statement.

Definition at line 22 of file stmts.c.

References mysql_errno_d, and mysql_stmt_errno_d.

const chr_t* stmt_error (MYSQL_STMT * *local*)

Return the error string for a mysql statement.

Parameters:

local A single MYSQL_STMT* parameter.

Returns:

This function returns the mysql error string, or NULL if unable to retrieve it.

Definition at line 39 of file stmts.c.

References mysql_errno_d, mysql_error_d, mysql_stmt_errno_d, and mysql_stmt_error_d.

Referenced by stmt_bind_param(), stmt_exec_affected_conn(), stmt_exec_conn(), stmt_get_result_conn(), stmt_insert_conn(), and stmt_prepare().

bool_t stmt_exec (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*)

Execute a prepared mysql statement.

See also:

stmt_exec_conn()

Parameters:

group the mysql prepared statement to be executed.

parameters the parameters to be bound to the statement.

Returns:

false on failure, or true on success.

Definition at line 312 of file stmts.c.

References log_info, PL_RESERVED, pool_pull(), pool_release(), sql_pool, and stmt_exec_conn().

Referenced by mail_db_hide_message(), meta_data_flags_add(), meta_data_flags_remove(), meta_data_flags_replace(), smtp_update_receive_stats(), smtp_update_transmission_stats(), and teacher_data_delete().

uint64_t stmt_exec_affected (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*)

Execute a statement and return the number of affected rows.

Note:

From the MySQL documentation (section 20.9.3.1): "Because mysql_affected_rows() returns an unsigned value, you can check for -1 by comparing the return value to (my_ulonglong)-1 (or to (my_ulonglong)~0, which is equivalent)."

See also:

mysql_stmt_affected_rows()

Parameters:

group the prepared mysql statement to be executed.

parameters the parameters to be bound to the prepared statement.

Returns:

-1 on failure, 0 on failure or no rows affected, and a positive number on success.

Definition at line 473 of file stmts.c.

References log_info, PL_RESERVED, pool_pull(), pool_release(), sql_pool, and stmt_exec_affected_conn().

Referenced by contact_delete(), contact_detail_delete(), contact_detail_upsert(), contact_update(), contact_update_stamp(), magma_folder_delete(), magma_folder_rename(), meta_data_delete_folder(), meta_data_delete_tag(), meta_data_insert_tag(), meta_data_truncate_tags(), meta_data_update_folder_name(), meta_data_update_lock(), meta_data_update_log(), meta_data_user_save_storage_keys(), user_config_delete(), and user_config_upsert().

uint64_t stmt_exec_affected_conn (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*, uint32_t *connection*)

Execute a statement on a specified mysql connection and return the number of affected rows.

Note:

From the MySQL documentation (section 20.9.3.1): "Because mysql_affected_rows() returns an unsigned value, you can check for -1 by comparing the return value to (my_ulonglong)-1 (or to (my_ulonglong)~0, which is equivalent)."

See also:

mysql_stmt_affected_rows()

Parameters:

group the prepared mysql statement to be executed.

parameters the parameters to be bound to the prepared statement.

connection the mysql connection id over which the specified statement will be executed.

Returns:

-1 on failure, 0 on failure or no rows affected, and a positive number on success.

Definition at line 441 of file stmts.c.

References log_info, mysql_stmt_affected_rows_d, mysql_stmt_execute_d, stmt_bind_param(), stmt_error(), and stmt_reset().

Referenced by mail_db_delete_message(), mail_db_update_message_folder(), meta_data_acknowledge_alert(), meta_data_user_save_storage_keys(), stmt_exec_affected(), and tank_delete_object().

bool_t stmt_exec_conn (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*, uint32_t *connection*)

Execute a prepared mysql statement over a specified connection.

Note:

This function will also reset the prepared statement and bind its parameters.

Parameters:

group the mysql prepared statement to be executed.

parameters the parameters to be bound to the statement.

connection the mysql connection id.

Returns:

false on failure, or true on success.

Definition at line 283 of file stmts.c.

References log_info, mysql_stmt_execute_d, stmt_bind_param(), stmt_error(), and stmt_reset().

Referenced by mail_db_insert_duplicate_message(), mail_db_insert_message(), register_data_insert_user(), and stmt_exec().

table_t* stmt_get_result (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*)

Execute a prepared mysql statement and return the result.

Parameters:

group the prepared mysql statement to be executed.

parameters the parameters to be passed with the query.

Returns:

the result of the query, or NULL on failure.

Definition at line 363 of file stmts.c.

References log_info, PL_RESERVED, pool_pull(), pool_release(), sql_pool, and stmt_get_result_conn().

Referenced by config_fetch_host_number(), config_fetch_settings(), contact_details_fetch(), contacts_fetch(), magma_folder_fetch(), messages_fetch(), meta_data_check_mailbox(), meta_data_fetch_alerts(), meta_data_fetch_all_tags(), meta_data_fetch_folders(), meta_data_fetch_mailbox_aliases(), meta_data_fetch_message_tags(), meta_data_fetch_messages(), meta_data_fetch_user(), meta_data_user_build(), meta_data_user_build_storage_keys(), register_data_check_username(), register_data_fetch_blocklist(), smtp_check_authorized_from(), smtp_check_receive_quota(), smtp_check_transmit_quota(), smtp_fetch_authorization(), smtp_fetch_autoreply(), smtp_fetch_inbound(), smtp_fetch_rollmessages(), statistics_refresh(), teacher_data_fetch(), user_config_fetch(), warehouse_fetch_domains(), and warehouse_fetch_patterns().

uint32_t stmt_get_result_conn (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*, uint32_t *connection*)

Execute a prepared mysql statement on a specified connection and return the result.

Parameters:

group the prepared mysql statement to be executed.
parameters the parameters to be passed with the query.
connection the mysql connection identifier.

Returns:

the result of the query, or NULL on failure.

Definition at line 335 of file stmts.c.

References log_info, mysql_stmt_execute_d, res_stmt_store(), stmt_bind_param(), stmt_error(), and stmt_reset().

Referenced by meta_data_acknowledge_alert(), meta_data_user_build_storage_keys(), and stmt_get_result().

uint64_t stmt_insert (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*)

Execute a mysql prepared INSERT or UPDATE statement.

See also:

mysql_stmt_insert_id()

Parameters:

group the mysql prepared statement to be executed.
parameters the parameters to be bound to the prepared statement.

Returns:

0 on failure, or the last LAST_INSERT_ID() value returned for the mysql auto-increment column.

Definition at line 415 of file stmts.c.

References log_info, PL_RESERVED, pool_pull(), pool_release(), sql_pool, and stmt_insert_conn().

Referenced by contact_insert(), magma_folder_insert(), meta_data_insert_folder(), and smtp_insert_spamsig().

uint64_t stmt_insert_conn (MYSQL_STMT ** *group*, MYSQL_BIND * *parameters*, uint32_t *connection*)

Execute a mysql prepared INSERT or UPDATE statement on a specified connection.

See also:

mysql_stmt_insert_id()

Parameters:

group the mysql prepared statement to be executed.
parameters the parameters to be bound to the prepared statement.
connection the underlying connection on which the statement will be executed.

Returns:

0 on failure, or the last LAST_INSERT_ID() value returned for the mysql auto-increment column.

Definition at line 386 of file stmts.c.

References `log_info`, `mysql_stmt_execute_d`, `mysql_stmt_insert_id_d`, `stmt_bind_param()`, `stmt_error()`, and `stmt_reset()`.

Referenced by `mail_db_insert_duplicate_message()`, `mail_db_insert_message()`, `register_data_insert_user()`, `stmt_insert()`, and `tank_insert_object()`.

MYSQL_STMT* stmt_open (MYSQL * *mysql*)

Initialize a mysql prepared statement.

See also:

`mysql_stmt_init()`

Parameters:

mysql the underlying mysql connection.

Returns:

This function is a thin wrapper around the mysql library function `mysql_stmt_init_d()`.

Definition at line 57 of file `stmts.c`.

References `log_critical`, `mysql_stmt_init_d`, and `sql_error()`.

Referenced by `stmt_rebuild()`, and `stmt_start()`.

bool_t stmt_prepare (MYSQL_STMT * *group*, const char * *query*, unsigned long *length*)

Definition at line 121 of file `stmts.c`.

References `log_critical`, `mysql_stmt_attr_set_d`, `mysql_stmt_prepare_d`, and `stmt_error()`.

Referenced by `stmt_rebuild()`, and `stmt_start()`.

bool_t stmt_rebuild (uint32_t *connection*)

Definition at line 195 of file `stmts.c`.

References `log_critical`, `ns_length_get()`, `pool_get_obj()`, `queries`, `sql_pool`, `stmt_close()`, `stmt_open()`, and `stmt_prepare()`.

Referenced by `dspam_check()`, `dspam_train()`, and `stmt_reset()`.

MYSQL_STMT* stmt_reset (MYSQL_STMT ** *group*, uint32_t *connection*)

Definition at line 227 of file `stmts.c`.

References `log_critical`, `log_pedantic`, `mysql_stmt_reset_d`, `sql_ping()`, and `stmt_rebuild()`.

Referenced by `stmt_exec_affected_conn()`, `stmt_exec_conn()`, `stmt_get_result_conn()`, and `stmt_insert_conn()`.

bool_t stmt_start (void)

Initialize the global array of mysql prepared statements.

Note:

This function readies a copy of each prepared statement for every member of the global mysql connection pool.

Returns:

true on success or false on failure.

Definition at line 148 of file stmts.c.

References magma_t::database, magma_t::iface, log_critical, magma, mm_alloc(), mm_wipe(), ns_length_get(), pool_get_obj(), queries, sql_pool, stmt_open(), stmt_prepare(), and stmt_stop().

Referenced by sql_start().

void stmt_stop (void)

Definition at line 95 of file stmts.c.

References magma_t::database, magma_t::iface, magma, mm_free(), queries, and stmt_close().

Referenced by sql_stop(), and stmt_start().

Variable Documentation**chr_t* queries[] = { QUERIES_INIT }**

Definition at line 15 of file stmts.c.

Referenced by sanity_check(), stmt_rebuild(), stmt_start(), and stmt_stop().

magma/providers/database/transaction.c File Reference

MySQL transaction interface.
#include "magma.h"

Functions

- int64_t **tran_start** (void)
- *Pull a connection from the sql pool and start a transaction.* int64_t **tran_rollback** (int64_t transaction)
- *Rollback a pending transaction in the mysql database.* int64_t **tran_commit** (int64_t transaction)

Commit a transaction to the mysql database. Variables

- struct {
- chr_t * **command**
- size_t **length**
- } **tran_commands** [3]

Detailed Description

MySQL transaction interface.

Definition in file **transaction.c**.

Function Documentation

int64_t tran_commit (int64_t *transaction*)

Commit a transaction to the mysql database.

transaction.c

Parameters:

transaction the mysql connection identifier.

Returns:

0 on success, or a non-zero number on error.

Definition at line 77 of file transaction.c.

References `command`, `length`, `log_info`, `PLACER`, `pool_get_obj()`, `pool_release()`, `sql_error()`, `sql_pool`, `sql_query_conn()`, and `tran_commands`.

Referenced by `adjust_message_encryption()`, `mail_copy_message()`, `mail_move_message()`, `mail_remove_message()`, `mail_store_message()`, `portal_endpoint_alert_acknowledge()`, `register_business_step2()`, `tank_delete()`, and `tank_store()`.

int64_t tran_rollback (int64_t *transaction*)

Rollback a pending transaction in the mysql database.

Parameters:

transaction the mysql connection identifier.

Returns:

0 on success, or a non-zero number on error.

Definition at line 57 of file transaction.c.

References `command`, `length`, `log_info`, `PLACER`, `pool_get_obj()`, `pool_release()`, `sql_error()`, `sql_pool`, `sql_query_conn()`, and `tran_commands`.

Referenced by `adjust_message_encryption()`, `mail_copy_message()`, `mail_move_message()`, `mail_remove_message()`, `mail_store_message()`, `portal_endpoint_alert_acknowledge()`, `register_business_step2()`, `tank_delete()`, and `tank_store()`.

int64_t tran_start (void)

Pull a connection from the sql pool and start a transaction.

Returns:

-1 on failure, or a mysql connection id from the sql pool on success.

Definition at line 31 of file transaction.c.

References `command`, `length`, `log_info`, `PL_RESERVED`, `PLACER`, `pool_get_obj()`, `pool_pull()`, `pool_release()`, `sql_error()`, `sql_pool`, `sql_query_conn()`, and `tran_commands`.

Referenced by `adjust_message_encryption()`, `mail_copy_message()`, `mail_move_message()`, `mail_remove_message()`, `mail_store_message()`, `portal_endpoint_alert_acknowledge()`, `register_business_step2()`, `tank_delete()`, and `tank_store()`.

Variable Documentation**chr_t* command**

Definition at line 19 of file transaction.c.

Referenced by `imap_process()`, `molten_parse()`, `pop_process()`, `portal_endpoint()`, `smtp_process()`, `tran_commit()`, `tran_rollback()`, and `tran_start()`.

size_t length

Definition at line 20 of file transaction.c.

struct { ... } tran_commands[3]

LOW: Standardize how SQL connection and transaction handles are passed around. Some functions use `int64_t`, others use `uint32_t`. Perhaps we need to use a type def? Or perhaps create a similar type mapping for pool object handles?

Referenced by `tran_commit()`, `tran_rollback()`, and `tran_start()`.

magma/providers/images/freetype.c File Reference

Functions used to handle fonts.

```
#include "magma.h"
```

Functions

- **chr_t * lib_version_freetype** (void)
 - *Return the version string of the font library.* **bool_t lib_load_freetype** (void)
Initialize the font library and bind dynamically to the exported functions that are required.
-

Detailed Description

Functions used to handle fonts.

Definition in file **freetype.c**.

Function Documentation

bool_t lib_load_freetype (void)

Initialize the font library and bind dynamically to the exported functions that are required.

freetype.c

Returns:

true on success or false on failure.

Definition at line 33 of file freetype.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

chr_t* lib_version_freetype (void)

Return the version string of the font library.

Returns:

a pointer to a character string containing the font library version information.

Definition at line 21 of file freetype.c.

References FT_Library_Version_Static_d.

Referenced by lib_load().

magma/providers/images/gd.c File Reference

The functions used to create and images using the GD library.

```
#include "magma.h"
```

Functions

- **chr_t * lib_version_gd** (void)
 - *Return the version string of the gd library.* **bool_t lib_load_gd** (void)
Initialize the gd library and bind dynamically to the exported functions that are required.
-

Detailed Description

The functions used to create and images using the GD library.

Definition in file **gd.c**.

Function Documentation

bool_t lib_load_gd (void)

Initialize the gd library and bind dynamically to the exported functions that are required.

gd.c

Returns:

true on success or false on failure.

Definition at line 27 of file gd.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

chr_t* lib_version_gd (void)

Return the version string of the gd library.

Returns:

a pointer to a character string containing the gd library version information.

Definition at line 19 of file gd.c.

References gd_version_d.

Referenced by lib_load().

magma/providers/images/images.h File Reference

The functions used to create and modify image files. For our purposes that include fonts.

Functions

- **bool_t lib_load_gd** (void)
 - **gd.c chr_t * lib_version_gd** (void)
 - *Return the version string of the gd library.* **bool_t lib_load_freetype** (void)
 - **freetype.c chr_t * lib_version_freetype** (void)
 - *Return the version string of the font library.* **bool_t lib_load_jpeg** (void)
 - **jpeg.c chr_t * lib_version_jpeg** (void)
 - *Return the version string of the jpeg library.* **bool_t lib_load_png** (void)
 - **png.c chr_t * lib_version_png** (void)
- Return the version string of png library.*
-

Detailed Description

The functions used to create and modify image files. For our purposes that include fonts.

Definition in file **images.h**.

Function Documentation

bool_t lib_load_freetype (void)

freetype.c

freetype.c

Returns:

true on success or false on failure.

Definition at line 33 of file freetype.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

bool_t lib_load_gd (void)

gd.c

gd.c

Returns:

true on success or false on failure.

Definition at line 27 of file gd.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

bool_t lib_load_jpeg (void)

jpeg.c

jpeg.c

Returns:

true on success or false on failure.

Definition at line 27 of file jpeg.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

bool_t lib_load_png (void)

png.c

png.c

Returns:

true on success or false on failure.

Definition at line 44 of file png.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

chr_t* lib_version_freetype (void)

Return the version string of the font library.

Returns:

a pointer to a character string containing the font library version information.

Definition at line 21 of file freetype.c.

References FT_Library_Version_Static_d.

Referenced by lib_load().

chr_t* lib_version_gd (void)

Return the version string of the gd library.

Returns:

a pointer to a character string containing the gd library version information.

Definition at line 19 of file gd.c.

References gd_version_d.

Referenced by lib_load().

chr_t* lib_version_jpeg (void)

Return the version string of the jpeg library.

Returns:

a pointer to a character string containing the jpeg library version information.

Definition at line 19 of file jpeg.c.

References jpeg_version_d.

Referenced by lib_load().

chr_t* lib_version_png (void)

Return the version string of png library.

Returns:

a pointer to a character string containing the png library version information.

Definition at line 21 of file png.c.

References png_access_version_number_d.

Referenced by lib_load().

magma/providers/images/jpeg.c File Reference

The functions used to create and modify JPEG image files.

```
#include "magma.h"
```

Functions

- **chr_t * lib_version_jpeg** (void)
 - *Return the version string of the jpeg library.* **bool_t lib_load_jpeg** (void)
Initialize the jpeg library and bind dynamically to the exported functions that are required.
-

Detailed Description

The functions used to create and modify JPEG image files.

Definition in file **jpeg.c**.

Function Documentation

bool_t lib_load_jpeg (void)

Initialize the jpeg library and bind dynamically to the exported functions that are required.

jpeg.c

Returns:

true on success or false on failure.

Definition at line 27 of file jpeg.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

chr_t* lib_version_jpeg (void)

Return the version string of the jpeg library.

Returns:

a pointer to a character string containing the jpeg library version information.

Definition at line 19 of file jpeg.c.

References jpeg_version_d.

Referenced by lib_load().

magma/providers/images/png.c File Reference

The functions used to create and modify PNG image files.

```
#include "magma.h"
```

Functions

- **chr_t * lib_version_png** (void)
 - *Return the version string of png library.* **bool_t lib_load_png** (void)
Initialize the png library and bind dynamically to the exported functions that are required.
-

Detailed Description

The functions used to create and modify PNG image files.

Definition in file **png.c**.

Function Documentation

bool_t lib_load_png (void)

Initialize the png library and bind dynamically to the exported functions that are required.

png.c

Returns:

true on success or false on failure.

Definition at line 44 of file png.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

chr_t* lib_version_png (void)

Return the version string of png library.

Returns:

a pointer to a character string containing the png library version information.

Definition at line 21 of file png.c.

References png_access_version_number_d.

Referenced by lib_load().

magma/providers/parsers/json.c File Reference

JSON serialization.

```
#include "magma.h"
```

Functions

- **chr_t * lib_version_jansson** (void)
- *Return the version string of libjansson.* **bool_t lib_load_jansson** (void)

Initialize the Jansson library and bind dynamically to the exported functions that are required.

Detailed Description

JSON serialization.

Definition in file **json.c**.

Function Documentation

bool_t lib_load_jansson (void)

Initialize the Jansson library and bind dynamically to the exported functions that are required.

json.c

Returns:

true on success or false on failure.

Definition at line 27 of file json.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

chr_t* lib_version_jansson (void)

Return the version string of libjansson.

Returns:

a pointer to a character string containing the libjansson version information.

Definition at line 19 of file json.c.

References jansson_version_d.

Referenced by lib_load().

magma/providers/parsers/xml.c File Reference

The interface to the xml parser.

```
#include "magma.h"
```

Functions

- `const chr_t * lib_version_xml` (void)
- *Return the version string of libxml. `bool_t lib_load_xml` (void)*
- *Initialize libxml and bind dynamically to the exported functions that are required. `void xml_node_free` (xmlNodePtr node)*
- *Free an xml node. `xmlChar * xml_encode` (xmlDocPtr doc, `stringer_t *string`)*
- *Encode an xml string, performing proper replacement of defined entities and non-ASCII values. `int_t xml_xpath_set_namespace` (xmlXPathContextPtr ctx, xmlChar *prefix, xmlChar *ns_uri)*
- *Set the namespace for an xpath context. `xmlNodePtr xml_node_add_sibling` (xmlNodePtr current, xmlNodePtr element)*
- *Add an xml node to the list of siblings of another xml node. `xmlNodePtr xml_node_new` (uchr_t *name)*
- *Create an xml node. `stringer_t * xml_node_get_content_st` (xmlNodePtr node)*
- *Get the content of an xml node as a managed string. `void xml_node_set_content` (xmlNodePtr node, `uchr_t *content`)*
- *Set the content of an xml node. `xmlAttrPtr xml_node_set_property` (xmlNodePtr node, `uchr_t *name`, `uchr_t *value`)*
- *Set an attribute of an xml node. `uint64_t xml_get_xpath_uint64` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)*
- *Get the value of a node element or attribute selected by an xpath expression as an unsigned 64-bit integer. `uint32_t xml_get_xpath_uint32` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)*
- *Get the value of a node element or attribute selected by an xpath expression as an unsigned 32-bit integer. `uint16_t xml_get_xpath_uint16` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)*
- *Get the value of a node element or attribute selected by an xpath expression as an unsigned 16-bit integer. `uint8_t xml_get_xpath_uint8` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)*
- *Get the value of a node element or attribute selected by an xpath expression as an unsigned 8-bit integer. `int64_t xml_get_xpath_int64` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)*
- *Get the value of a node element or attribute selected by an xpath expression as a signed 64-bit integer. `int32_t xml_get_xpath_int32` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)*
- *Get the value of a node element or attribute selected by an xpath expression as a signed 32-bit integer. `int16_t xml_get_xpath_int16` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)*
- *Get the value of a node element or attribute selected by an xpath expression as a signed 16-bit integer. `int8_t xml_get_xpath_int8` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)*
- *Get the value of a node element or attribute selected by an xpath expression as a signed 8-bit integer. `stringer_t * xml_get_xpath_st` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)*
- *Get the value of a node element or attribute selected by an xpath expression as a managed string. `bool_t xml_set_xpath_uint64` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query, `uint64_t val`)*
- *Set the value of a node element selected by an xpath expression, as a 64-bit unsigned integer. `bool_t xml_set_xpath_ns` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query, `uchr_t *val`)*
- *Set the value of a node element selected by an xpath expression, as a null-terminated string. `bool_t xml_set_xpath_property` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query, `uchr_t *name`, `uchr_t *val`)*
- *Set the value of a specified property of a node element selected by an xpath expression. `chr_t * xml_get_xpath_ns` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)*
- *Get the content of an evaluated xpath expression as a null-terminated string. `size_t xml_get_xpath_node_count` (xmlXPathContextPtr xpath_ctx, xmlChar *xpath_query)*
- *This function is not currently being called by any other part of the code. `void xml_error` (void *ctx, `const chr_t *format`,...)*

- A function handler for displaying xml parser error messages to the user. void **xml_free_xpath_ctx** (xmlXPathContextPtr ctx)
- Free an xml xpath context. void **xml_free_xpath_obj** (xmlXPathObjectPtr obj)
- Free an xml xpath object. xmlXPathContextPtr **xml_create_xpath_ctx** (xmlDocPtr doc)
- Create an xpath context for an xml document. xmlXPathObjectPtr **xml_xpath_eval** (const **uchar_t** *xpath, xmlXPathContextPtr ctx)
- Evaluate an xml xpath expression. xmlParserCtxtPtr **xml_create_parser_ctx** (void)
- Create and initialize a new xml parser context. void **xml_free_parser_ctx** (xmlParserCtxtPtr ctx)
- Destroy an xml parser context. void **xml_free_doc** (xmlDocPtr doc)
- Free an xml document. **stringer_t** * **xml_dump_doc** (xmlDocPtr doc)
- Get an xml document object as a string. xmlDocPtr **xml_create_doc** (xmlParserCtxtPtr ctx, const **chr_t** *buffer, **int_t** size, const **chr_t** *url, const **chr_t** *encoding, **int_t** options)
- Create a new xml document object from specified user data. void **xml_stop** (void)
- Cleanup the state and allocated memory of the xml parser in preparation to be shutdown. **bool_t** **xml_start** (void)

Initialize the xml parser library. Variables

- **chr_t** **xml_version_string** [8]
- pthread_mutex_t **log_mutex**

Detailed Description

The interface to the xml parser.

Definition in file **xml.c**.

Function Documentation

bool_t **lib_load_xml** (void)

Initialize libxml and bind dynamically to the exported functions that are required.

xml.c

Returns:

true on success or false on failure.

Definition at line 31 of file xml.c.

References lib_symbols(), M_BIND, uint64_conv_ns(), and xml_version_string.

Referenced by lib_load().

const chr_t* **lib_version_xml** (void)

Return the version string of libxml.

Returns:

a pointer to a character string containing the libxml version information.

Definition at line 22 of file xml.c.

References `xml_version_string`.

Referenced by `lib_load()`.

`xmlDocPtr xml_create_doc(xmlParserCtxtPtr ctx, const chr_t * buffer, int_t size, const chr_t * url, const chr_t * encoding, int_t options)`

Create a new xml document object from specified user data.

See also:

`xmlCtxtReadMemory()`

Parameters:

ctx a pointer to the xml context to be used for the parsing operation.

buffer a null-terminated string containing the xml data to be parsed.

size the length, in bytes, of the xml data buffer to be parsed.

url a null-terminated string containing the base url to be used for the document.

encoding a null-terminated string specifying the document encoding type, or NULL for default.

options a value containing a mask of xml parser options of type `xmlParserOption`.

Returns:

NULL on failure, or a pointer to the xml document tree of the parsed xml text on success.

Definition at line 1054 of file `xml.c`.

References `log_pedantic`, and `xmlCtxtReadMemory_d`.

Referenced by `http_page_get()`.

`xmlParserCtxtPtr xml_create_parser_ctx(void)`

Create and initialize a new xml parser context.

Returns:

NULL on failure, or a newly initialized xml parser context on success.

Definition at line 963 of file `xml.c`.

References `log_pedantic`, `xml_error()`, and `xmlNewParserCtxt_d`.

Referenced by `http_page_get()`.

`xmlXPathContextPtr xml_create_xpath_ctx(xmlDocPtr doc)`

Create an xpath context for an xml document.

Parameters:

doc a pointer to the input xml document object.

Returns:

NULL on failure, or a pointer to the new xpath context on success.

Definition at line 919 of file `xml.c`.

References `log_pedantic`, and `xmlXPathNewContext_d`.

Referenced by `http_page_get()`.

stringer_t* xml_dump_doc (xmlDocPtr doc)

Get an xml document object as a string.

Parameters:

doc a pointer to the xml document object to be serialized.

Returns:

NULL on failure or a pointer to a managed string containing the specified xml document's serialized data on success.

Definition at line 1020 of file xml.c.

References mm_free(), st_import(), and xmlDocDumpFormatMemory_d.

Referenced by contact_print_form(), contact_print_message(), portal_print_login(), statistics_process(), teacher_print_form(), and teacher_print_message().

xmlChar* xml_encode (xmlDocPtr doc, stringer_t * string)

Encode an xml string, performing proper replacement of defined entities and non-ASCII values.

Parameters:

doc a pointer to the xml document containing the DTD to be used for the encoding process.

string a managed string containing the xml data to be encoded.

NULL on failure, or a newly allocated null-terminated string containing the encoded xml data on success.

Definition at line 74 of file xml.c.

References st_data_get(), and xmlEncodeEntitiesReentrant_d.

Referenced by contact_print_form().

void xml_error (void * ctx, const chr_t * format, ...)

A function handler for displaying xml parser error messages to the user.

Parameters:

ctx a placeholder; ignored.

format a null-terminated string containing a format string for the error message to be displayed.

... a variable argument list containing the parameters to the format string.

Returns:

This function returns no value.

Definition at line 859 of file xml.c.

References log_mutex, mutex_lock(), and mutex_unlock().

Referenced by xml_create_parser_ctx().

void xml_free_doc (xmlDocPtr doc)

Free an xml document.

Parameters:

doc a pointer to the xml document to be freed.

Returns:

This function returns no value.

Definition at line 1003 of file xml.c.

References log_pedantic, and xmlFreeDoc_d.

Referenced by http_page_free().

void xml_free_parser_ctx (xmlParserCtxtPtr *ctx*)

Destroy an xml parser context.

Parameters:

ctx a pointer to the xml parser context to be freed.

Returns:

This function returns no value.

Definition at line 986 of file xml.c.

References log_pedantic, and xmlFreeParserCtxt_d.

Referenced by http_page_free().

void xml_free_xpath_ctx (xmlXPathContextPtr *ctx*)

Free an xml xpath context.

Parameters:

ctx a pointer to the xml xpath context to be freed.

Returns:

This function returns no value.

Definition at line 885 of file xml.c.

References log_pedantic, and xmlXPathFreeContext_d.

Referenced by http_page_free().

void xml_free_xpath_obj (xmlXPathObjectPtr *obj*)

Free an xml xpath object.

Parameters:

obj a pointer to the xml xpath object to be freed.

Returns:

This function returns no value.

Definition at line 902 of file xml.c.

References log_pedantic, and xmlXPathFreeObject_d.

Referenced by xml_set_xpath_ns(), xml_set_xpath_property(), and xml_set_xpath_uint64().

int16_t xml_get_xpath_int16 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as a signed 16-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as a signed 16-bit integer.

Definition at line 508 of file xml.c.

References `int16_conv_st()`, `log_pedantic`, `ns_length_get()`, `PLACER`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

int32_t xml_get_xpath_int32 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as a signed 32-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as a signed 32-bit integer.

Definition at line 453 of file xml.c.

References `int32_conv_st()`, `log_pedantic`, `ns_length_get()`, `PLACER`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

int64_t xml_get_xpath_int64 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as a signed 64-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as a signed 64-bit integer.

Definition at line 398 of file xml.c.

References `int64_conv_st()`, `log_pedantic`, `ns_length_get()`, `PLACER`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

int8_t xml_get_xpath_int8 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as a signed 8-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as a signed 8-bit integer.

Definition at line 563 of file xml.c.

References `int8_conv_st()`, `log_pedantic`, `ns_length_get()`, `PLACER`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

size_t xml_get_xpath_node_count (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

This function is not currently being called by any other part of the code.

Definition at line 822 of file xml.c.

References `log_pedantic`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

chr_t* xml_get_xpath_ns (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the content of an evaluated xpath expression as a null-terminated string.

Parameters:

ctx a pointer to the xpath context to be used for the evaluation.

xpath_query a null-terminated string containing the xpath expression to be evaluated.

Returns:

NULL on failure, or a null-terminated string containing the value of the xpath expression's specified node on success.

Definition at line 771 of file xml.c.

References `log_pedantic`, `ns_dupe()`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

stringer_t* xml_get_xpath_st (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as a managed string.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as a managed string.

Definition at line 618 of file xml.c.

References `log_pedantic`, `ns_length_get()`, `st_import()`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

uint16_t xml_get_xpath_uint16 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as an unsigned 16-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as an unsigned 16-bit integer.

Definition at line 288 of file xml.c.

References `log_pedantic`, `ns_length_get()`, `PLACER`, `uint16_conv_st()`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

uint32_t xml_get_xpath_uint32 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as an unsigned 32-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as an unsigned 32-bit integer.

Definition at line 234 of file xml.c.

References `log_pedantic`, `ns_length_get()`, `PLACER`, `uint32_conv_st()`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

uint64_t xml_get_xpath_uint64 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as an unsigned 64-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as an unsigned 64-bit integer.

Definition at line 179 of file xml.c.

References `log_pedantic`, `ns_length_get()`, `PLACER`, `uint64_conv_st()`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

uint8_t xml_get_xpath_uint8 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*)

Get the value of a node element or attribute selected by an xpath expression as an unsigned 8-bit integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

Returns:

0 on failure, or the value of the selected node as an unsigned 8-bit integer.

Definition at line 343 of file xml.c.

References `log_pedantic`, `ns_length_get()`, `PLACER`, `uint8_conv_st()`, `xmlXPathEvalExpression_d`, and `xmlXPathFreeObject_d`.

xmlNodePtr xml_node_add_sibling (xmlNodePtr *current*, xmlNodePtr *element*)

Add an xml node to the list of siblings of another xml node.

Note:

If the sibling node was already inserted into a document it will first be unlinked.

Parameters:

current a pointer to the xml node to which the sibling node will be added.

element a pointer to the sibling node to be added to the target node.

Returns:

NULL on failure, or a pointer to the sibling node on success.

Definition at line 99 of file xml.c.

References `xmlAddSibling_d`.

Referenced by `contact_business_add_error()`, and `teacher_add_error()`.

void xml_node_free (xmlNodePtr *node*)

Free an xml node.

Parameters:

node a pointer to the xml node to be freed.

Returns:

This function returns no value.

Definition at line 62 of file xml.c.

References `xmlFreeNode_d`.

Referenced by `contact_business_add_error()`, and `teacher_add_error()`.

stringer_t* xml_node_get_content_st (xmlNodePtr *node*)

Get the content of an xml node as a managed string.

Parameters:

node a pointer to the xml node to be queried.

Returns:

NULL on failure, or a pointer to a managed string

Definition at line 119 of file xml.c.

References `log_pedantic`, `st_import()`, `xmlBufferContent_d`, `xmlBufferCreate_d`, `xmlBufferFree_d`, `xmlBufferLength_d`, and `xmlNodeBufGetContent_d`.

xmlNodePtr xml_node_new (uchr_t * *name*)

Create an xml node.

Parameters:

name the name of the new xml node.

Returns:

NULL on failure, or a pointer to the newly allocated and initialized xml node on success.

Definition at line 109 of file xml.c.

References xmlNewNode_d.

Referenced by contact_business_add_error(), and teacher_add_error().

void xml_node_set_content (xmlNodePtr *node*, uchr_t * *content*)

Set the content of an xml node.

Parameters:

node a pointer to the xml node to be set.

content a null-terminated string containing the new content of the xml node.

Returns:

This function returns no value.

Definition at line 155 of file xml.c.

References xmlNodeSetContent_d.

Referenced by contact_business_add_error(), portal_print_login(), teacher_add_error(), xml_set_xpath_ns(), and xml_set_xpath_uint64().

xmlAttrPtr xml_node_set_property (xmlNodePtr *node*, uchr_t * *name*, uchr_t * *value*)

Set an attribute of an xml node.

Parameters:

node a pointer to the xml node to be set.

name a null-terminated string containing the name of the node attribute to be set.

value a null-terminated string containing the new value of the specified xml node's attribute.

Returns:

NULL on failure, or a pointer to node's attribute on success.

Definition at line 168 of file xml.c.

References xmlSetProp_d.

Referenced by contact_business_add_error(), teacher_add_error(), and xml_set_xpath_property().

bool_t xml_set_xpath_ns (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*, uchr_t * *val*)

Set the value of a node element selected by an xpath expression, as a null-terminated string.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

val the new value of the selected node, as a null-terminated string.

Returns:

true if the value was set successfully, or false on failure.

Definition at line 707 of file xml.c.

References log_pedantic, xml_free_xpath_obj(), xml_node_set_content(), and xml_xpath_eval().

Referenced by contact_print_form(), contact_print_message(), statistics_process(), teacher_print_form(), and teacher_print_message().

**bool_t xml_set_xpath_property (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*,
uchr_t * *name*, uchr_t * *val*)**

Set the value of a specified property of a node element selected by an xpath expression.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

name a null-terminated string containing the name of the node's property to be set.

val the new value of the selected node's property, as a null-terminated string.

Returns:

true if the selected node's value was set successfully, or false on failure.

Definition at line 740 of file xml.c.

References log_pedantic, xml_free_xpath_obj(), xml_node_set_property(), and xml_xpath_eval().

Referenced by contact_print_form(), contact_print_message(), and teacher_print_form().

**bool_t xml_set_xpath_uint64 (xmlXPathContextPtr *xpath_ctx*, xmlChar * *xpath_query*,
uint64_t *val*)**

Set the value of a node element selected by an xpath expression, as a 64-bit unsigned integer.

Parameters:

xpath_ctx a pointer to the specified xpath context.

xpath_query a null-terminated string containing the xpath expression to select the desired node.

val the new value of the selected node, as a 64-bit unsigned integer.

Returns:

true if the value was set successfully, or false on failure.

Definition at line 673 of file xml.c.

References log_pedantic, xml_free_xpath_obj(), xml_node_set_content(), and xml_xpath_eval().

Referenced by statistics_process().

bool_t xml_start (void)

Initialize the xml parser library.

Returns:

This function returns no value.

Definition at line 1082 of file xml.c.

References xmlInitParser_d.

Referenced by process_start().

void xml_stop (void)

Cleanup the state and allocated memory of the xml parser in preparation to be shutdown.

Returns:

This function returns no value.

Definition at line 1069 of file xml.c.

References xmlCleanupGlobals_d, xmlCleanupParser_d, and xmlMemoryDump_d.

Referenced by process_stop().

xmlXPathObjectPtr xml_xpath_eval (const uchr_t * *xpath*, xmlXPathContextPtr *ctx*)

Evaluate an xml xpath expression.

Parameters:

a null-terminated string containing the xpath expression to be evaluated.

a pointer to the xpath context in which to perform the evaluation. NULL on failure, or a pointer to the xml path object of the evaluation on success.

Definition at line 943 of file xml.c.

References log_pedantic, and xmlXPathEvalExpression_d.

Referenced by contact_business_add_error(), portal_print_login(), teacher_add_error(), xml_set_xpath_ns(), xml_set_xpath_property(), and xml_set_xpath_uint64().

int_t xml_xpath_set_namespace (xmlXPathContextPtr *ctxt*, xmlChar * *prefix*, xmlChar * *ns_uri*)

Set the namespace for an xpath context.

See also:

xmlXPathRegisterNs()

Parameters:

ctxt a pointer to the specified xpath context.

prefix a pointer to the prefix of the namespace as a string.

ns_uri a pointer to the uri of the namespace as a string.

Returns:

-1 on failure or 0 on success.

Definition at line 87 of file xml.c.
References xmlXPathRegisterNs_d.
Referenced by http_page_get().

Variable Documentation

pthread_mutex_t log_mutex

Definition at line 17 of file log.c.
Referenced by log_disable(), log_enable(), log_internal(), log_rotate(), and xml_error().

chr_t xml_version_string[8]

Definition at line 15 of file xml.c.
Referenced by lib_load_xml(), and lib_version_xml().

magma/providers/providers.h File Reference

The entry point for the provider modules.

```
#include "symbols.h"
#include "database/database.h"
#include "consumers/consumers.h"
#include "checkers/checkers.h"
#include "compress/compress.h"
#include "cryptography/cryptography.h"
#include "parsers/parsers.h"
#include "storage/storage.h"
#include "images/images.h"
```

Data Structures

- struct **symbol_t**

Functions

- **bool_t lib_load** (void)
- *Load libmagma dynamically and resolve all external dependencies from 3rd party providers.* void **lib_unload** (void)
- *Unload libmagma from memory.* **bool_t lib_symbols** (size_t count, symbol_t symbols[])

Initialize and bind an import symbol table.

Detailed Description

The entry point for the provider modules.

Definition in file **providers.h**.

Function Documentation

bool_t lib_load (void)

Load libmagma dynamically and resolve all external dependencies from 3rd party providers.

Returns:

false on failure or true on success.

Definition at line 74 of file symbols.c.

References `build_stamp()`, `build_version()`, `magma_t::config`, `CONSTANT`, `magma_t::file`, `host_platform()`, `host_version()`, `lib_load_bzip()`, `lib_load_cache()`, `lib_load_clamav()`, `lib_load_dkim()`, `lib_load_dspam()`, `lib_load_freetype()`, `lib_load_gd()`, `lib_load_jansson()`, `lib_load_jpeg()`, `lib_load_lzo()`, `lib_load_mysql()`, `lib_load_openssl()`, `lib_load_png()`, `lib_load_spf()`, `lib_load_tokyo()`, `lib_load_xml()`, `lib_load_zlib()`, `lib_magma`, `lib_version_bzip()`, `lib_version_cache()`, `lib_version_clamav()`, `lib_version_dkim()`, `lib_version_dspam()`, `lib_version_freetype()`, `lib_version_gd()`, `lib_version_jansson()`, `lib_version_jpeg()`, `lib_version_lzo()`, `lib_version_mysql()`, `lib_version_openssl()`, `lib_version_png()`, `lib_version_spf()`, `lib_version_tokyo()`, `lib_version_xml()`, `lib_version_zlib()`, `magma_t::library`, `log_critical`, `log_pedantic`,

magma, MANAGEDBUF, NULLER, serv_charset_mysql(), serv_schema_mysql(), serv_type_mysql(), serv_version_mysql(), st_char_get(), and st_cmp_ci_eq().

Referenced by process_start().

bool_t lib_symbols (size_t count, symbol_t symbols[])

Initialize and bind an import symbol table.

See also:

dlsym()

Parameters:

count the number of symbols in the table.

symbols the symbol table to be patched.

Returns:

true on success or false on failure.

Definition at line 24 of file symbols.c.

References lib_magma, log_critical, NULLER, and st_cmp_cs_eq().

Referenced by lib_load_bzip(), lib_load_cache(), lib_load_clamav(), lib_load_dkim(), lib_load_dspam(), lib_load_freetype(), lib_load_gd(), lib_load_jansson(), lib_load_jpeg(), lib_load_lzo(), lib_load_mysql(), lib_load_openssl(), lib_load_png(), lib_load_spf(), lib_load_tokyo(), lib_load_xml(), and lib_load_zlib().

void lib_unload (void)

Unload libmagma from memory.

Returns:

This function returns no value.

Definition at line 61 of file symbols.c.

References lib_magma, magma_t::library, magma, and magma_t::unload.

Referenced by process_stop().

magma/providers/storage/storage.h File Reference

The Tokyo Cabinet interface, which is primarily used for memory and disk based storage.

Data Structures

- struct `__attribute__`
- struct `__attribute__`

Defines

- #define `TANK_ENTRY_VERSION` 100
- #define `TANK_RECORD_VERSION` 100

Enumerations

- enum { `TANK_COMPRESS_LZO` = 1, `TANK_COMPRESS_ZLIB` = 2, `TANK_COMPRESS_BZIP` = 4 }

Functions

- `bool_t lib_load_tokyo` (void)
- *Initialize the Tokyo Cabinet library and bind dynamically to the exported functions that are required.* const `chr_t * lib_version_tokyo` (void)
- `uint64_t tree_count` (void *inx)
- *Binary trees.* `inx_t * tree_alloc` (uint64_t options, void *data_free)
- *Create a new on-memory tree database.* void `tank_stop` (void)
- *Startup and shutdown.* `bool_t tank_start` (void)
- *Initialize the local tank storage system.* `uint64_t tank_size` (void)
- *Info functions.* `uint64_t tank_count` (void)
- *Count the number of objects in all local storage tanks.* `uint64_t tank_cycle` (void)
- *Get the next storage tank.* void `tank_maintain` (void)
- *Maintenance.* `bool_t tank_delete` (uint64_t hnum, uint64_t tnum, uint64_t unum, uint64_t onum)
- *Object handling.* `stringer_t * tank_load` (uint64_t hnum, uint64_t tnum, uint64_t unum, uint64_t onum)
- `uint64_t tank_store` (uint64_t hnum, uint64_t tnum, uint64_t unum, `stringer_t *data`, uint64_t flags)
- *Store a binary object on the file system.* `bool_t tank_delete_object` (int64_t transaction, uint64_t hnum, uint64_t tnum, uint64_t unum, uint64_t onum)
- *Delete a tank object from the mysql database.* `uint64_t tank_insert_object` (int64_t transaction, uint64_t hnum, uint64_t tnum, uint64_t unum, uint64_t size, uint64_t flags)

Insert a tank object into the mysql database. Variables

- enum { ... } `TANK_FLAGS_E`

Detailed Description

The Tokyo Cabinet interface, which is primarily used for memory and disk based storage.

Definition in file `storage.h`.

Define Documentation

#define TANK_ENTRY_VERSION 100

Definition at line 23 of file storage.h.

Referenced by tank_store().

#define TANK_RECORD_VERSION 100

Definition at line 24 of file storage.h.

Referenced by tank_load(), and tank_store().

Enumeration Type Documentation

anonymous enum

Enumerator:

TANK_COMPRESS_LZO
TANK_COMPRESS_ZLIB
TANK_COMPRESS_BZIP

Definition at line 26 of file storage.h.

Function Documentation

bool_t lib_load_tokyo (void)

Initialize the Tokyo Cabinet library and bind dynamically to the exported functions that are required.

Returns:

true on success or false on failure.

Definition at line 24 of file tokyo.c.

References lib_symbols(), and M_BIND.

Referenced by lib_load().

const chr_t* lib_version_tokyo (void)

Definition at line 16 of file tokyo.c.

Referenced by lib_load().

uint64_t tank_count (void)

Count the number of objects in all local storage tanks.

Returns:

the total number of all objects contained in all tanks.

Definition at line 52 of file tank.c.

References count, log_pedantic, store, tanks_num, and tchdbrnum_d.

uint64_t tank_cycle (void)

Get the next storage tank.

Note:

This function cycles through all the storage tanks in order, to ensure homogeneous data distribution.

Returns:

the next storage tank in line.

Definition at line 34 of file tank.c.

References mutex_lock(), mutex_unlock(), tanks_lock, tanks_next, and tanks_num.

bool_t tank_delete (uint64_t hnum, uint64_t tnum, uint64_t unum, uint64_t onum)

Object handling.

Delete a binary object on the file system.

Parameters:

hnum The host number.

tnum The tank number.

unum The user number.

onum The object number.

Returns:

Returns true if the object was deleted without an error.

Definition at line 95 of file tank.c.

References log_critical, log_error, store, tank_delete_object(), tchdbecode_d, tchdberrmsg_d, tchdbout_d, tran_commit(), tran_rollback(), and tran_start().

bool_t tank_delete_object (int64_t transaction, uint64_t hnum, uint64_t tnum, uint64_t unum, uint64_t onum)

Delete a tank object from the mysql database.

Parameters:

transaction a mysql connection identifier for which a transaction has been started.

hnum the host number.

tnum the storage tank number.

unum the usernum of the user that owns the object.

onum the stored object id.

Returns:

false on failure, or true on success.

Definition at line 72 of file data.c.

References log_pedantic, mm_wipe(), and stmt_exec_affected_conn().

Referenced by tank_delete().

uint64_t tank_insert_object (int64_t *transaction*, uint64_t *hnum*, uint64_t *tnum*, uint64_t *unum*, uint64_t *size*, uint64_t *flags*)

Insert a tank object into the mysql database.

Parameters:

transaction a mysql connection identifier for which a transaction has been started.

hnum the host number.

tnum the storage tank number.

unum the usernum of the user that owns the object.

size the length, in bytes, of the object to be stored.

flags 0 on failure, or the objectnum field for the newly inserted object.

Definition at line 24 of file data.c.

References mm_wipe(), and stmt_insert_conn().

Referenced by tank_store().

stringer_t* tank_load (uint64_t *hnum*, uint64_t *tnum*, uint64_t *unum*, uint64_t *onum*)

Load and decompress the data for the object described by the input parameters.

Parameters:

hnum The host number.

tnum The tank number.

unum The user number.

onum The object number.

Returns:

Returns the object data inside a stringer_t or NULL to indicate an error occurred.

Definition at line 156 of file tank.c.

References decompress_bzip(), decompress_lzo(), decompress_zlib(), log_check, log_error, log_pedantic, mm_copy(), st_import(), store, TANK_COMPRESS_BZIP, TANK_COMPRESS_LZO, TANK_COMPRESS_ZLIB, TANK_RECORD_VERSION, tcfree_d, tchdbecode_d, tchdberrmsg_d, and tchdbget_d.

void tank_maintain (void)

Maintenance.

Maintenance.

Returns:

This function returns no value.

Definition at line 423 of file tank.c.

References store, tanks_num, and tchdbdefrag_d.

uint64_t tank_size (void)

Info functions.

Info functions.

Returns:

the total number of bytes occupied by all objects in all storage tanks.

Definition at line 74 of file tank.c.

References store, tanks_num, and tchdbfsiz_d.

bool_t tank_start (void)

Initialize the local tank storage system.

Returns:

Returns true if all of the storage tanks were setup successfully and the system is ready to store data.

Definition at line 503 of file tank.c.

References log_critical, magma, MAGMA_FILEPATH_MAX, mm_alloc(), ns_length_get(), magma_t::storage, store, magma_t::tank, tank_open(), and tanks_num.

void tank_stop (void)

Startup and shutdown.

Close all of the storage system file handles and release the resources they were using.

Definition at line 541 of file tank.c.

References mm_free(), store, tank_close(), and tanks_num.

uint64_t tank_store (uint64_t hnum, uint64_t tnum, uint64_t unum, stringer_t * data, uint64_t flags)

Store a binary object on the file system.

Note:

Flags include (TANK_COMPRESS_LZO | TANK_COMPRESS_ZLIB | TANK_COMPRESS_BZIP),

Parameters:

hnum the host number.

tnum the tank number.

unum the userid number.

data a managed string (placer) with the data to be stored.

flags a bitmask of flags specifying encryption, compression, and replication.

Returns:

0 on failure, If the object is stored the object number is returned, or 0 if an error occurs.

Definition at line 262 of file tank.c.

References compress_body_length(), compress_bzip(), compress_free(), compress_lzo(), compress_total_length(), compress_zlib(), log_error, mm_alloc(), mm_copy(), mm_free(), mm_wipe(),

st_data_get(), st_length_get(), store, TANK_COMPRESS_BZIP, TANK_COMPRESS_LZO, TANK_COMPRESS_ZLIB, TANK_ENTRY_VERSION, tank_insert_object(), TANK_RECORD_VERSION, tanks_num, tchdbdecode_d, tchdberrmsg_d, tchdbout_d, tchdbputasync_d, tran_commit(), tran_rollback(), and tran_start().

inx_t* tree_alloc (uint64_t *options*, void * *data_free*)

Create a new on-memory tree database.

See also:

tcndbnew2()

Parameters:

options

data_free

Returns:

Definition at line 288 of file tree.c.

References inx_t::cursor_alloc, inx_t::cursor_free, inx_t::cursor_key_active, inx_t::cursor_key_next, inx_t::cursor_reset, inx_t::cursor_value_active, inx_t::cursor_value_next, inx_t::data_free, inx_t::delete, inx_t::find, inx_t::index, inx_t::index_free, inx_t::index_truncate, inx_t::insert, log_pedantic, mm_alloc(), mm_free(), inx_t::options, tcndbnew2_d, tree_cmp(), tree_cursor_alloc(), tree_cursor_free(), tree_cursor_key_active(), tree_cursor_key_next(), tree_cursor_reset(), tree_cursor_value_active(), tree_cursor_value_next(), tree_delete(), tree_find(), tree_free(), tree_insert(), and tree_truncate().

Referenced by inx_alloc().

uint64_t tree_count (void * *inx*)

Binary trees.

Binary trees.

See also:

tcndbrnum()

Parameters:

inx

Returns:

the record count.

Definition at line 46 of file tree.c.

References inx_t::index, and tcndbrnum_d.

Variable Documentation

enum { ... } TANK_FLAGS_E

magma/providers/storage/tank.c File Reference

The storage system interface. Uses Tokyo Cabinet to store the underlying files.

```
#include "magma.h"
```

Functions

- `uint64_t tank_cycle` (void)
- *Get the next storage tank.* `uint64_t tank_count` (void)
- *Count the number of objects in all local storage tanks.* `uint64_t tank_size` (void)
- *Count the amount of space used by all local storage tanks.* `bool_t tank_delete` (`uint64_t hnum`, `uint64_t tnum`, `uint64_t unum`, `uint64_t onum`)
- *Object handling.* `stringer_t * tank_load` (`uint64_t hnum`, `uint64_t tnum`, `uint64_t unum`, `uint64_t onum`)
- `uint64_t tank_store` (`uint64_t hnum`, `uint64_t tnum`, `uint64_t unum`, `stringer_t *data`, `uint64_t flags`)
- *Store a binary object on the file system.* `void tank_maintain` (void)
- *Perform periodic maintenance on the storage tanks (defragment them in the background).* `TCHDB * tank_open` (`char *location`)
- `void tank_close` (`TCHDB *ctx`)
- `bool_t tank_start` (void)
- *Initialize the local tank storage system.* `void tank_stop` (void)

Startup and shutdown. Variables

- `uint64_t tanks_num` = 4
- `uint64_t tanks_next` = 0
- `pthread_mutex_t tanks_lock`
- struct {
- `uint8_t tuner`
- `TCHDB * system`
- `TCHDB ** tanks`
- } `store`

Detailed Description

The storage system interface. Uses Tokyo Cabinet to store the underlying files.

Definition in file `tank.c`.

Function Documentation

`void tank_close (TCHDB * ctx)`

Cleanly closes a storage file context and releases the memory.

Parameters:

ctx The storage context that needs to be closed.

Definition at line 475 of file `tank.c`.

References `log_error`, `tchdbclose_d`, `tchdbdefrag_d`, `tchdbdel_d`, `tchdbecode_d`, `tchdberrmsg_d`, `tchdboptimize_d`, `tchdbpath_d`, and `tchdbsync_d`.

Referenced by tank_stop().

uint64_t tank_count (void)

Count the number of objects in all local storage tanks.

Returns:

the total number of all objects contained in all tanks.

Definition at line 52 of file tank.c.

References count, log_pedantic, store, tanks_num, and tchdbnum_d.

uint64_t tank_cycle (void)

Get the next storage tank.

Note:

This function cycles through all the storage tanks in order, to ensure homogeneous data distribution.

Returns:

the next storage tank in line.

Definition at line 34 of file tank.c.

References mutex_lock(), mutex_unlock(), tanks_lock, tanks_next, and tanks_num.

bool_t tank_delete (uint64_t hnum, uint64_t tnum, uint64_t unum, uint64_t onum)

Object handling.

Delete a binary object on the file system.

Parameters:

hnum The host number.

tnum The tank number.

unum The user number.

onum The object number.

Returns:

Returns true if the object was deleted without an error.

Definition at line 95 of file tank.c.

References log_critical, log_error, store, tank_delete_object(), tchdbecode_d, tchdberrmsg_d, tchdbout_d, tran_commit(), tran_rollback(), and tran_start().

stringer_t* tank_load (uint64_t hnum, uint64_t tnum, uint64_t unum, uint64_t onum)

Load and decompress the data for the object described by the input parameters.

Parameters:

hnum The host number.

tnum The tank number.

unum The user number.

onum The object number.

Returns:

Returns the object data inside a `stringer_t` or `NULL` to indicate an error occurred.

Definition at line 156 of file `tank.c`.

References `decompress_bzip()`, `decompress_lzo()`, `decompress_zlib()`, `log_check`, `log_error`, `log_pedantic`, `mm_copy()`, `st_import()`, `store`, `TANK_COMPRESS_BZIP`, `TANK_COMPRESS_LZO`, `TANK_COMPRESS_ZLIB`, `TANK_RECORD_VERSION`, `tcfree_d`, `tchdbecode_d`, `tchdberrmsg_d`, and `tchdbget_d`.

void tank_maintain (void)

Perform periodic maintenance on the storage tanks (defragment them in the background).

Maintenance.

Returns:

This function returns no value.

Definition at line 423 of file `tank.c`.

References `store`, `tanks_num`, and `tchdbdefrag_d`.

TCHDB* tank_open (char * location)

Open a storage context for the given location. Use the **store.tuner** parameters to configure the context.

Parameters:

location The path to the file where the data is (or will be) stored.

Returns:

Returns a pointer to the storage context on success and `NULL` on failure.

Definition at line 437 of file `tank.c`.

References `log_critical`, `tchdbdel_d`, `tchdbecode_d`, `tchdberrmsg_d`, `tchdbnew_d`, `tchdbopen_d`, `tchdbsetmutex_d`, and `tchdbtune_d`.

Referenced by `tank_start()`.

uint64_t tank_size (void)

Count the amount of space used by all local storage tanks.

Info functions.

Returns:

the total number of bytes occupied by all objects in all storage tanks.

Definition at line 74 of file `tank.c`.

References `store`, `tanks_num`, and `tchdbfsiz_d`.

bool_t tank_start (void)

Initialize the local tank storage system.

Returns:

Returns true if all of the storage tanks were setup successfully and the system is ready to store data.

Definition at line 503 of file tank.c.

References log_critical, magma, MAGMA_FILEPATH_MAX, mm_alloc(), ns_length_get(), magma_t::storage, store, magma_t::tank, tank_open(), and tanks_num.

void tank_stop (void)

Startup and shutdown.

Close all of the storage system file handles and release the resources they were using.

Definition at line 541 of file tank.c.

References mm_free(), store, tank_close(), and tanks_num.

uint64_t tank_store (uint64_t *hnum*, uint64_t *tnum*, uint64_t *unum*, stringer_t * *data*, uint64_t *flags*)

Store a binary object on the file system.

Note:

Flags include (TANK_COMPRESS_LZO | TANK_COMPRESS_ZLIB | TANK_COMPRESS_BZIP),

Parameters:

hnum the host number.

tnum the tank number.

unum the userid number.

data a managed string (placer) with the data to be stored.

flags a bitmask of flags specifying encryption, compression, and replication.

Returns:

0 on failure, If the object is stored the object number is returned, or 0 if an error occurs.

Definition at line 262 of file tank.c.

References compress_body_length(), compress_bzip(), compress_free(), compress_lzo(), compress_total_length(), compress_zlib(), log_error, mm_alloc(), mm_copy(), mm_free(), mm_wipe(), st_data_get(), st_length_get(), store, TANK_COMPRESS_BZIP, TANK_COMPRESS_LZO, TANK_COMPRESS_ZLIB, TANK_ENTRY_VERSION, tank_insert_object(), TANK_RECORD_VERSION, tanks_num, tchdbcode_d, tchdberrmsg_d, tchdbout_d, tchdbputasync_d, tran_commit(), tran_rollback(), and tran_start().

Variable Documentation

struct { ... } store

Referenced by config_free(), tank_count(), tank_delete(), tank_load(), tank_maintain(), tank_size(), tank_start(), tank_stop(), and tank_store().

TCHDB* system

Definition at line 22 of file tank.c.

TCHDB tanks**

Definition at line 23 of file tank.c.

pthread_mutex_t tanks_lock

Initial value:

PTHREAD_MUTEX_INITIALIZER

Definition at line 17 of file tank.c.

Referenced by tank_cycle().

uint64_t tanks_next = 0

Definition at line 16 of file tank.c.

Referenced by tank_cycle().

uint64_t tanks_num = 4

Definition at line 15 of file tank.c.

Referenced by tank_count(), tank_cycle(), tank_maintain(), tank_size(), tank_start(), tank_stop(), and tank_store().

uint8_t tuner

Definition at line 21 of file tank.c.

magma/providers/storage/tokyo.c File Reference

Tokyo Cabinet symbols.

```
#include "magma.h"
```

Functions

- `const char * lib_version_tokyo` (void)
- `bool_t lib_load_tokyo` (void)

Initialize the Tokyo Cabinet library and bind dynamically to the exported functions that are required.

Detailed Description

Tokyo Cabinet symbols.

Definition in file `tokyo.c`.

Function Documentation

`bool_t lib_load_tokyo` (void)

Initialize the Tokyo Cabinet library and bind dynamically to the exported functions that are required.

Returns:

true on success or false on failure.

Definition at line 24 of file `tokyo.c`.

References `lib_symbols()`, and `M_BIND`.

Referenced by `lib_load()`.

`const char* lib_version_tokyo` (void)

Definition at line 16 of file `tokyo.c`.

Referenced by `lib_load()`.

magma/providers/storage/tree.c File Reference

Use Tokyo Cabinet to provide a tree based index implementation for the generic index interface.
`#include "magma.h"`

Data Structures

- `struct __attribute__`

Functions

- `int tree_cmp` (const char *aptr, int asiz, const char *bptr, int bsiz, void *op)
- `uint64_t tree_count` (void *inx)
- *Get the number of records in an in-memory tree.* `void * tree_find` (void *inx, **multi_t** key)
- `bool_t tree_delete` (void *inx, **multi_t** key)
- `bool_t tree_insert` (void *inx, **multi_t** key, void *data)
- `bool_t tree_cursor_next` (tree_cursor_t *cursor)
- `void * tree_cursor_value_next` (tree_cursor_t *cursor)
- `void * tree_cursor_value_active` (tree_cursor_t *cursor)
- `multi_t tree_cursor_key_next` (tree_cursor_t *cursor)
- `multi_t tree_cursor_key_active` (tree_cursor_t *cursor)
- `void tree_cursor_reset` (tree_cursor_t *cursor)
- `void tree_cursor_free` (tree_cursor_t *cursor)
- `void * tree_cursor_alloc` (inx_t *inx)
- `void tree_truncate` (void *inx)
- `void tree_free` (void *inx)
- `inx_t * tree_alloc` (uint64_t options, void *data_free)

Create a new on-memory tree database.

Detailed Description

Use Tokyo Cabinet to provide a tree based index implementation for the generic index interface.

Definition in file **tree.c**.

Function Documentation

inx_t* tree_alloc (uint64_t options, void * data_free)

Create a new on-memory tree database.

See also:

`tcndbnew2()`

Parameters:

options
data_free

Returns:

Definition at line 288 of file tree.c.

References `inx_t::cursor_alloc`, `inx_t::cursor_free`, `inx_t::cursor_key_active`, `inx_t::cursor_key_next`, `inx_t::cursor_reset`, `inx_t::cursor_value_active`, `inx_t::cursor_value_next`, `inx_t::data_free`, `inx_t::delete`, `inx_t::find`, `inx_t::index`, `inx_t::index_free`, `inx_t::index_truncate`, `inx_t::insert`, `log_pedantic`, `mm_alloc()`, `mm_free()`, `inx_t::options`, `tcndbnew2_d`, `tree_cmp()`, `tree_cursor_alloc()`, `tree_cursor_free()`, `tree_cursor_key_active()`, `tree_cursor_key_next()`, `tree_cursor_reset()`, `tree_cursor_value_active()`, `tree_cursor_value_next()`, `tree_delete()`, `tree_find()`, `tree_free()`, `tree_insert()`, and `tree_truncate()`.

Referenced by `inx_alloc()`.

int tree_cmp (const char * *aptr*, int *asiz*, const char * *bptr*, int *bsiz*, void * *op*)

The return value is positive if the former is big, negative if the latter is big, 0 if both are equivalent. If asked to search for a NULL key pointer the function will always return 0. This allows us to match every record.

Parameters:

aptr a pointer to the region of the first key.
asize the length, in bytes, of the region of the first key.
bptr a pointer to the region of the second key.
bsiz the length, in bytes, of the region of the first key.
op an optional pointer to an opaque object.

Returns:

0 if the two blocks are identical; otherwise, a signed integer indicating the difference.

Definition at line 33 of file tree.c.

References `cmp_mt_mt()`.

Referenced by `tree_alloc()`.

uint64_t tree_count (void * *inx*)

Get the number of records in an in-memory tree.

Binary trees.

See also:

`tcndbrnum()`

Parameters:

inx

Returns:

the record count.

Definition at line 46 of file tree.c.

References `inx_t::index`, and `tcndbrnum_d`.

void* tree_cursor_alloc (inx_t * *inx*)

Definition at line 190 of file tree.c.

References `inx_t::index`, `log_pedantic`, `mm_alloc()`, `tcndbdup_d`, and `tcndbiterinit_d`.

Referenced by `tree_alloc()`.

void tree_cursor_free (tree_cursor_t * *cursor*)

Definition at line 177 of file tree.c.

References mm_free(), and tcndbdel_d.

Referenced by tree_alloc().

multi_t tree_cursor_key_active (tree_cursor_t * *cursor*)

Definition at line 159 of file tree.c.

Referenced by tree_alloc().

multi_t tree_cursor_key_next (tree_cursor_t * *cursor*)

Definition at line 152 of file tree.c.

References mt_get_null(), and tree_cursor_next().

Referenced by tree_alloc().

bool_t tree_cursor_next (tree_cursor_t * *cursor*)

Definition at line 117 of file tree.c.

References mm_copy(), mt_get_null(), tcfree_d, tcndbget3_d, and tcndbiternext2_d.

Referenced by tree_cursor_key_next(), and tree_cursor_value_next().

void tree_cursor_reset (tree_cursor_t * *cursor*)

Definition at line 163 of file tree.c.

References tcndbdel_d, tcndbdup_d, and tcndbiterinit_d.

Referenced by tree_alloc().

void* tree_cursor_value_active (tree_cursor_t * *cursor*)

Definition at line 148 of file tree.c.

Referenced by tree_alloc().

void* tree_cursor_value_next (tree_cursor_t * *cursor*)

Definition at line 141 of file tree.c.

References tree_cursor_next().

Referenced by tree_alloc().

bool_t tree_delete (void * *inx*, multi_t *key*)

Definition at line 61 of file tree.c.

References `inx_t::count`, `inx_t::data_free`, `inx_t::index`, `mm_copy()`, `mt_free()`, `inx_t::serial`, `tcndbgetboth_d`, and `tcndbout_d`.

Referenced by `tree_alloc()`.

void* tree_find (void * *inx*, multi_t *key*)

Definition at line 51 of file `tree.c`.

References `tcfree_d`, and `tcndbget3_d`.

Referenced by `tree_alloc()`.

void tree_free (void * *inx*)

Frees all of the resources used by a tree based index.

Parameters:

inx A pointer to the index that should be freed.

Definition at line 261 of file `tree.c`.

References `inx_t::index`, `tcndbdel_d`, and `tree_truncate()`.

Referenced by `tree_alloc()`.

bool_t tree_insert (void * *inx*, multi_t *key*, void * *data*)

Makes a copy of the key and then attempts to add the new entry to the tree index. Note that because this is a tree index, duplicates are not allowed, so if the key already exists, false is returned.

Parameters:

inx The index were adding the entry too.

key The retrieval key expressed as a **multi_t**.

data The data buffer being stored.

Returns:

Returns true if the entry was added, or false to indicate an existing duplicate key or an error.

Definition at line 95 of file `tree.c`.

References `inx_t::count`, `inx_t::index`, `log_info`, `mt_dupe()`, `mt_free()`, `mt_is_empty()`, `inx_t::serial`, and `tcndbputkeep_d`.

Referenced by `tree_alloc()`.

void tree_truncate (void * *inx*)

Truncates a tree based index.

Parameters:

inx A pointer to the index that should be truncated.

Definition at line 216 of file `tree.c`.

References `count`, `inx_t::data_free`, `inx_t::index`, `length`, `mt_free()`, `tclistdel_d`, `tclistnum_d`, `tclistval_d`, `tcmtreekeys_d`, and `tcmtreevals_d`.

Referenced by `tree_alloc()`, and `tree_free()`.

magma/providers/symbols.c File Reference

Functions used to load the external library symbols.

```
#include "magma.h"
```

Functions

- **bool_t lib_symbols** (size_t count, symbol_t symbols[])
- *Initialize and bind an import symbol table.* void **lib_unload** (void)
- *Unload libmagma from memory.* **bool_t lib_load** (void)

Load libmagma dynamically and resolve all external dependencies from 3rd party providers. Variables

- void * **lib_magma** = NULL

Detailed Description

Functions used to load the external library symbols.

Definition in file **symbols.c**.

Function Documentation

bool_t lib_load (void)

Load libmagma dynamically and resolve all external dependencies from 3rd party providers.

Returns:

false on failure or true on success.

Definition at line 74 of file symbols.c.

References build_stamp(), build_version(), magma_t::config, CONSTANT, magma_t::file, host_platform(), host_version(), lib_load_bzip(), lib_load_cache(), lib_load_clamav(), lib_load_dkim(), lib_load_dspam(), lib_load_freetype(), lib_load_gd(), lib_load_jansson(), lib_load_jpeg(), lib_load_lzo(), lib_load_mysql(), lib_load_openssl(), lib_load_png(), lib_load_spf(), lib_load_tokyo(), lib_load_xml(), lib_load_zlib(), lib_magma, lib_version_bzip(), lib_version_cache(), lib_version_clamav(), lib_version_dkim(), lib_version_dspam(), lib_version_freetype(), lib_version_gd(), lib_version_jansson(), lib_version_jpeg(), lib_version_lzo(), lib_version_mysql(), lib_version_openssl(), lib_version_png(), lib_version_spf(), lib_version_tokyo(), lib_version_xml(), lib_version_zlib(), magma_t::library, log_critical, log_pedantic, magma, MANAGEDBUF, NULLER, serv_charset_mysql(), serv_schema_mysql(), serv_type_mysql(), serv_version_mysql(), st_char_get(), and st_cmp_ci_eq().

Referenced by process_start().

bool_t lib_symbols (size_t count, symbol_t symbols[])

Initialize and bind an import symbol table.

See also:

dlsym()

Parameters:

count the number of symbols in the table.

symbols the symbol table to be patched.

Returns:

true on success or false on failure.

Definition at line 24 of file symbols.c.

References lib_magma, log_critical, NULLER, and st_cmp_cs_eq().

Referenced by lib_load_bzip(), lib_load_cache(), lib_load_clamav(), lib_load_dkim(), lib_load_dspam(), lib_load_freetype(), lib_load_gd(), lib_load_jansson(), lib_load_jpeg(), lib_load_lzo(), lib_load_mysql(), lib_load_openssl(), lib_load_png(), lib_load_spf(), lib_load_tokyo(), lib_load_xml(), and lib_load_zlib().

void lib_unload (void)

Unload libmagma from memory.

Returns:

This function returns no value.

Definition at line 61 of file symbols.c.

References lib_magma, magma_t::library, magma, and magma_t::unload.

Referenced by process_stop().

Variable Documentation**void* lib_magma = NULL**

Definition at line 15 of file symbols.c.

Referenced by lib_load(), lib_symbols(), and lib_unload().

magma/providers/symbols.h File Reference

External function pointers/definitions.

Defines

- #define **M_BIND(x)**

Functions

- const char *STD_CALL **mysql_character_set_name** (MYSQL *mysql)
- char **SSL_version_str_d __attribute__((common)) = NULL
- **TOKYO_STACK_OF** (SSL_COMP) (*SSL_COMP_get_compression_methods_d)(void) __attribute__((common))

Variables

- memcached_return_t(* **memcached_flush_d**)(memcached_st *ptr, time_t expiration) __attribute__((common)) = NULL
- **MEMCACHED.** void(* **memcached_free_d**)(memcached_st *ptr) __attribute__((common)) = NULL
- const char *(* **memcached_lib_version_d**)(void) __attribute__((common)) = NULL
- memcached_st(* **memcached_create_d**)(memcached_st *ptr) __attribute__((common)) = NULL
- const char *(* **memcached_strerror_d**)(const memcached_st *ptr, memcached_return_t rc) __attribute__((common)) = NULL
- memcached_return_t(* **memcached_behavior_set_d**)(memcached_st *ptr, const memcached_behavior_t flag, uint64_t data) __attribute__((common)) = NULL
- memcached_return_t(* **memcached_delete_d**)(memcached_st *ptr, const char *key, size_t key_length, time_t expiration) __attribute__((common)) = NULL
- memcached_return_t(* **memcached_server_add_with_weight_d**)(memcached_st *ptr, const char *hostname, in_port_t port, uint32_t weight) __attribute__((common)) = NULL
- memcached_return_t(* **memcached_decrement_d**)(memcached_st *ptr, const char *key, size_t key_length, uint32_t offset, uint64_t *value) __attribute__((common)) = NULL
- memcached_return_t(* **memcached_increment_d**)(memcached_st *ptr, const char *key, size_t key_length, uint32_t offset, uint64_t *value) __attribute__((common)) = NULL
- char *(* **memcached_get_d**)(memcached_st *ptr, const char *key, size_t key_length, size_t *value_length, uint32_t *flags, memcached_return_t *error) __attribute__((common)) = NULL
- memcached_return_t(* **memcached_add_d**)(memcached_st *ptr, const char *key, size_t key_length, const char *value, size_t value_length, time_t expiration, uint32_t flags) __attribute__((common)) = NULL
- memcached_return_t(* **memcached_set_d**)(memcached_st *ptr, const char *key, size_t key_length, const char *value, size_t value_length, time_t expiration, uint32_t flags) __attribute__((common)) = NULL
- memcached_return_t(* **memcached_append_d**)(memcached_st *ptr, const char *key, size_t key_length, const char *value, size_t value_length, time_t expiration, uint32_t flags) __attribute__((common)) = NULL
- memcached_return_t(* **memcached_prepend_d**)(memcached_st *ptr, const char *key, size_t key_length, const char *value, size_t value_length, time_t expiration, uint32_t flags) __attribute__((common)) = NULL
- memcached_return_t(* **memcached_replace_d**)(memcached_st *ptr, const char *key, size_t key_length, const char *value, size_t value_length, time_t expiration, uint32_t flags) __attribute__((common)) = NULL
- memcached_return_t(* **memcached_cas_d**)(memcached_st *ptr, const char *key, size_t key_length, const char *value, size_t value_length, time_t expiration, uint32_t flags, uint64_t cas) __attribute__((common)) = NULL
- memcached_return_t(* **memcached_decrement_with_initial_d**)(memcached_st *ptr, const char *key, size_t key_length, uint64_t offset, uint64_t initial, time_t expiration, uint64_t *value) __attribute__((common)) = NULL

- `memcached_return_t(* memcached_increment_with_initial_d)(memcached_st *ptr, const char *key, size_t key_length, uint64_t offset, uint64_t initial, time_t expiration, uint64_t *value) __attribute__((common)) = NULL`
- `const char *(* BZ2_bzlibVersion_d)(void) __attribute__((common)) = NULL`
- `BZIP. int(* BZ2_bzBuffToBuffDecompress_d)(char *dest, unsigned int *destLen, char *source, unsigned int sourceLen, int small, int verbosity) __attribute__((common)) = NULL`
- `int(* BZ2_bzBuffToBuffCompress_d)(char *dest, unsigned int *destLen, char *source, unsigned int sourceLen, int blockSize100k, int verbosity, int workFactor) __attribute__((common)) = NULL`
- `void(* cl_shutdown_d)(void) __attribute__((common)) = NULL`
- `CLAMAV. int(* lt_dlexit_d)(void) __attribute__((common)) = NULL`
- `const char *(* cl_retver_d)(void) __attribute__((common)) = NULL`
- `int(* cl_init_d)(unsigned int initoptions) __attribute__((common)) = NULL`
- `const char *(* cl_strerror_d)(int clerror) __attribute__((common)) = NULL`
- `struct cl_engine *(* cl_engine_new_d)(void) __attribute__((common)) = NULL`
- `int(* cl_statfree_d)(struct cl_stat *dbstat) __attribute__((common)) = NULL`
- `int(* cl_engine_free_d)(struct cl_engine *engine) __attribute__((common)) = NULL`
- `int(* cl_engine_compile_d)(struct cl_engine *engine) __attribute__((common)) = NULL`
- `int(* cl_statchkdir_d)(const struct cl_stat *dbstat) __attribute__((common)) = NULL`
- `int(* cl_statinidir_d)(const char *dirname, struct cl_stat *dbstat) __attribute__((common)) = NULL`
- `int(* cl_countsigs_d)(const char *path, unsigned int countoptions, unsigned int *sigs) __attribute__((common)) = NULL`
- `int(* cl_engine_set_num_d)(struct cl_engine *engine, enum cl_engine_field field, long long num) __attribute__((common)) = NULL`
- `int(* cl_engine_set_str_d)(struct cl_engine *engine, enum cl_engine_field field, const char *str) __attribute__((common)) = NULL`
- `int(* cl_load_d)(const char *path, struct cl_engine *engine, unsigned int *signo, unsigned int dboptions) __attribute__((common)) = NULL`
- `int(* cl_scandesc_d)(int desc, const char **virname, unsigned long int *scanned, const struct cl_engine *engine, unsigned int scanoptions) __attribute__((common)) = NULL`
- `const char *(* dspam_version_d)(void) __attribute__((common)) = NULL`
- `DSPAM. int(* dspam_detach_d)(DSPAM_CTX *CTX) __attribute__((common)) = NULL`
- `void(* dspam_destroy_d)(DSPAM_CTX *CTX) __attribute__((common)) = NULL`
- `int(* dspam_init_driver_d)(DRIVER_CTX *DTX) __attribute__((common)) = NULL`
- `int(* dspam_shutdown_driver_d)(DRIVER_CTX *DTX) __attribute__((common)) = NULL`
- `int(* dspam_attach_d)(DSPAM_CTX *CTX, void *dbh) __attribute__((common)) = NULL`
- `int(* dspam_process_d)(DSPAM_CTX *CTX, const char *message) __attribute__((common)) = NULL`
- `DSPAM_CTX *(* dspam_create_d)(const char *username, const char *group, const char *home, int operating_mode, u_int32_t flags) __attribute__((common)) = NULL`
- `DKIM_STAT(* dkim_eoh_d)(DKIM *dkim) __attribute__((common)) = NULL`
- `DKIM. void(* dkim_close_d)(DKIM_LIB *lib) __attribute__((common)) = NULL`
- `DKIM_STAT(* dkim_free_d)(DKIM *dkim) __attribute__((common)) = NULL`
- `uint32_t(* dkim_libversion_d)(void) __attribute__((common)) = NULL`
- `DKIM_STAT(* dkim_eom_d)(DKIM *dkim, _Bool *testkey) __attribute__((common)) = NULL`
- `const char *(* dkim_getresultstr_d)(DKIM_STAT result) __attribute__((common)) = NULL`
- `DKIM_STAT(* dkim_body_d)(DKIM *dkim, u_char *buf, size_t len) __attribute__((common)) = NULL`
- `DKIM_STAT(* dkim_header_d)(DKIM *dkim, u_char *hdr, size_t len) __attribute__((common)) = NULL`
- `DKIM_STAT(* dkim_getsighdrx_d)(DKIM *dkim, u_char *buf, size_t len, size_t initial) __attribute__((common)) = NULL`
- `DKIM *(* dkim_verify_d)(DKIM_LIB *libhandle, const unsigned char *id, void *memclosure, DKIM_STAT *statp) __attribute__((common)) = NULL`
- `DKIM_LIB *(* dkim_init_d)(void *(*mallocf)(void *closure, size_t nbytes), void(*freef)(void *closure, void *p)) __attribute__((common)) = NULL`

- DKIM *(*(dkim_sign_d)(DKIM_LIB *libhandle, const unsigned char *id, void *memclosure, const dkim_sigkey_t secretkey, const unsigned char *selector, const unsigned char *domain, dkim_canon_t hdr_canon_alg, dkim_canon_t body_canon_alg, dkim_alg_t sign_alg, off_t length, DKIM_STAT *statp) __attribute__((common)) = NULL*
- DKIM_STAT *(*(dkim_chunk_d)(DKIM *dkim, unsigned char *chunkp, size_t len) __attribute__((common)) = NULL*
- void *(*(FT_Library_Version_Static_d)(FT_Int *amajor, FT_Int *aminor, FT_Int *apatch) __attribute__((common)) = NULL*
- *FreeType*. const char *(*(gd_version_d)(void) __attribute__((common)) = NULL*
- GD. void *(*(gdFree_d)(void *m) __attribute__((common)) = NULL*
- void *(*(gdImageGifPtr_d)(gdImagePtr im, int *size) __attribute__((common)) = NULL*
- void *(*(gdImageDestroy_d)(gdImagePtr im) __attribute__((common)) = NULL*
- void *(*(gdImageJpegPtr_d)(gdImagePtr im, int *size, int quality) __attribute__((common)) = NULL*
- void *(*(gdImageSetPixel_d)(gdImagePtr im, int x, int y, int color) __attribute__((common)) = NULL*
- gdImagePtr *(*(gdImageCreate_d)(int sx, int sy) __attribute__((common)) = NULL*
- int *(*(gdImageColorResolve_d)(gdImagePtr im, int r, int g, int b) __attribute__((common)) = NULL*
- char *(*(gdImageStringFT_d)(gdImage *im, int *breect, int fg, char *fontlist, double psize, double angle, int x, int y, char *string) __attribute__((common)) = NULL*
- const char *(*(jpeg_version_d)(void) __attribute__((common)) = NULL*
- JPEG. const char *(*(lzo_version_string_d)(void) __attribute__((common)) = NULL*
- LZO. int *(*(__lzo_init_v2_d)(unsigned, int, int, int, int, int, int, int, int) __attribute__((common)) = NULL*
- lzo_uint32 *(*(lzo_adler32_d)(lzo_uint32 _adler, const lzo_bytelp _buf, lzo_uint _len) __attribute__((common)) = NULL*
- int *(*(lzo1x_1_compress_d)(const lzo_byte *src, lzo_uint src_len, lzo_byte *dst, lzo_uintp dst_len, lzo_voidp wrkmem) __attribute__((common)) = NULL*
- int *(*(lzo1x_decompress_safe_d)(const lzo_byte *src, lzo_uint src_len, lzo_byte *dst, lzo_uintp dst_len, lzo_voidp wrkmem) __attribute__((common)) = NULL*
- void *(*(my_once_free_d)(void) __attribute__((common)) = NULL*
- MYSQL. void *(*(mysql_server_end_d)(void) __attribute__((common)) = NULL*
- void *(*(mysql_thread_end_d)(void) __attribute__((common)) = NULL*
- int *(*(mysql_ping_d)(MYSQL *mysql) __attribute__((common)) = NULL*
- void *(*(mysql_close_d)(MYSQL *mysql) __attribute__((common)) = NULL*
- my_bool *(*(mysql_thread_init_d)(void) __attribute__((common)) = NULL*
- const char *(*(mysql_get_server_info_d)(MYSQL *mysql) __attribute__((common)) = NULL*
- MYSQL *(*(mysql_init_d)(MYSQL *mysql) __attribute__((common)) = NULL*
- unsigned int *(*(mysql_thread_safe_d)(void) __attribute__((common)) = NULL*
- int *(*(mysql_stmt_fetch_d)(MYSQL_STMT *stmt) __attribute__((common)) = NULL*
- my_bool *(*(mysql_stmt_close_d)(MYSQL_STMT *) __attribute__((common)) = NULL*
- unsigned int *(*(mysql_errno_d)(MYSQL *mysql) __attribute__((common)) = NULL*
- const char *(*(mysql_error_d)(MYSQL *mysql) __attribute__((common)) = NULL*
- int *(*(mysql_stmt_execute_d)(MYSQL_STMT *stmt) __attribute__((common)) = NULL*
- my_bool *(*(mysql_embedded_d)(void) __attribute__((common)) = NULL*
- void *(*(mysql_free_result_d)(MYSQL_RES *result) __attribute__((common)) = NULL*
- my_bool *(*(mysql_stmt_reset_d)(MYSQL_STMT *stmt) __attribute__((common)) = NULL*
- my_ulonglong *(*(mysql_insert_id_d)(MYSQL *mysql) __attribute__((common)) = NULL*
- unsigned long *(*(mysql_thread_id_d)(MYSQL *mysql) __attribute__((common)) = NULL*
- MYSQL_STMT *(*(mysql_stmt_init_d)(MYSQL *mysql) __attribute__((common)) = NULL*
- MYSQL_ROW *(*(mysql_fetch_row_d)(MYSQL_RES *result) __attribute__((common)) = NULL*
- unsigned long *(*(mysql_get_client_version_d)(void) __attribute__((common)) = NULL*
- MYSQL_RES *(*(mysql_store_result_d)(MYSQL *mysql) __attribute__((common)) = NULL*
- int *(*(mysql_stmt_store_result_d)(MYSQL_STMT *stmt) __attribute__((common)) = NULL*
- my_ulonglong *(*(mysql_affected_rows_d)(MYSQL *mysql) __attribute__((common)) = NULL*
- my_ulonglong *(*(mysql_num_rows_d)(MYSQL_RES *result) __attribute__((common)) = NULL*

- unsigned int(* **mysql_stmt_errno_d**)(MYSQL_STMT *stmt) **__attribute__((common))** = NULL
- const char *(* **mysql_stmt_error_d**)(MYSQL_STMT *stmt) **__attribute__((common))** = NULL
- my_bool(* **mysql_stmt_free_result_d**)(MYSQL_STMT *stmt) **__attribute__((common))** = NULL
- unsigned int(* **mysql_num_fields_d**)(MYSQL_RES *result) **__attribute__((common))** = NULL
- my_ulonglong(* **mysql_stmt_num_rows_d**)(MYSQL_STMT *stmt) **__attribute__((common))** = NULL
- MYSQL_FIELD *(* **mysql_fetch_field_d**)(MYSQL_RES *result) **__attribute__((common))** = NULL
- const char *(* **mysql_character_set_name_d**)(MYSQL *mysql) **__attribute__((common))** = NULL
- my_ulonglong(* **mysql_stmt_insert_id_d**)(MYSQL_STMT *stmt) **__attribute__((common))** = NULL
- my_ulonglong(* **mysql_stmt_affected_rows_d**)(MYSQL_STMT *stmt) **__attribute__((common))** = NULL
- MYSQL_RES *(* **mysql_stmt_result_metadata_d**)(MYSQL_STMT *stmt) **__attribute__((common))** = NULL
- int(* **mysql_server_init_d**)(int argc, char **argv, char **groups) **__attribute__((common))** = NULL
- int(* **mysql_set_character_set_d**)(MYSQL *mysql, const char *csname) **__attribute__((common))** = NULL
- my_bool(* **mysql_stmt_bind_param_d**)(MYSQL_STMT *stmt, MYSQL_BIND *bind) **__attribute__((common))** = NULL
- int(* **mysql_options_d**)(MYSQL *mysql, enum mysql_option option, const void *arg) **__attribute__((common))** = NULL
- int(* **mysql_real_query_d**)(MYSQL *mysql, const char *query, unsigned long **length**) **__attribute__((common))** = NULL
- int(* **mysql_stmt_prepare_d**)(MYSQL_STMT *stmt, const char *query, unsigned long **length**) **__attribute__((common))** = NULL
- unsigned long(* **mysql_escape_string_d**)(char *to, const char *from, unsigned long **length**) **__attribute__((common))** = NULL
- my_bool(* **mysql_stmt_attr_set_d**)(MYSQL_STMT *stmt, enum enum_stmt_attr_type attr_type, const void *attr) **__attribute__((common))** = NULL
- my_bool(* **mysql_stmt_bind_result_d**)(MYSQL_STMT *stmt, MYSQL_BIND *bind) **__attribute__((common))** = NULL
- MYSQL *(* **mysql_real_connect_d**)(MYSQL *mysql, const char *name, const char *user, const char *passwd, const char *db, unsigned int port, const char *unix_socket, unsigned long client_flag) **__attribute__((common))** = NULL
- DH *(* **DH_new_d**)(void) **__attribute__((common))** = NULL
- OPENSSL int(* **SSL_connect_d**)(SSL *ssl) **__attribute__((common))** = NULL
- const SSL_METHOD *(* **SSLv23_client_method_d**)(void) **__attribute__((common))** = NULL
- const SSL_METHOD *(* **TLSv1_server_method_d**)(void) **__attribute__((common))** = NULL
- void(* **DH_free_d**)(DH *dh) **__attribute__((common))** = NULL
- int(* **RAND_status_d**)(void) **__attribute__((common))** = NULL
- void(* **EVP_cleanup_d**)(void) **__attribute__((common))** = NULL
- void(* **OBJ_cleanup_d**)(void) **__attribute__((common))** = NULL
- void(* **BN_free_d**)(BIGNUM *a) **__attribute__((common))** = NULL
- void(* **RAND_cleanup_d**)(void) **__attribute__((common))** = NULL
- void(* **SSL_free_d**)(SSL *ssl) **__attribute__((common))** = NULL
- int(* **SSL_accept_d**)(SSL *ssl) **__attribute__((common))** = NULL
- EC_KEY *(* **EC_KEY_new_d**)(void) **__attribute__((common))** = NULL
- void(* **CRYPTO_free_d**)(void *) **__attribute__((common))** = NULL
- void(* **ENGINE_cleanup_d**)(void) **__attribute__((common))** = NULL
- int(* **CRYPTO_num_locks_d**)(void) **__attribute__((common))** = NULL
- int(* **SSL_library_init_d**)(void) **__attribute__((common))** = NULL
- int(* **SSL_shutdown_d**)(SSL *ssl) **__attribute__((common))** = NULL
- void(* **BIO_sock_cleanup_d**)(void) **__attribute__((common))** = NULL
- void(* **ERR_free_strings_d**)(void) **__attribute__((common))** = NULL
- SSL *(* **SSL_new_d**)(SSL_CTX *ctx) **__attribute__((common))** = NULL
- const EVP_MD *(* **EVP_md4_d**)(void) **__attribute__((common))** = NULL
- const EVP_MD *(* **EVP_md5_d**)(void) **__attribute__((common))** = NULL

- `const EVP_MD *(*EVP_sha_d)(void) __attribute__((common)) = NULL`
- `void(*COMP_zlib_cleanup_d)(void) __attribute__((common)) = NULL`
- `const EVP_MD *(*EVP_sha1_d)(void) __attribute__((common)) = NULL`
- `void(*EC_KEY_free_d)(EC_KEY *key) __attribute__((common)) = NULL`
- `const char *(*OBJ_nid2sn_d)(int n) __attribute__((common)) = NULL`
- `const EVP_MD *(*EVP_sha224_d)(void) __attribute__((common)) = NULL`
- `const EVP_MD *(*EVP_sha256_d)(void) __attribute__((common)) = NULL`
- `const EVP_MD *(*EVP_sha384_d)(void) __attribute__((common)) = NULL`
- `const EVP_MD *(*EVP_sha512_d)(void) __attribute__((common)) = NULL`
- `void(*OBJ_NAME_cleanup_d)(int type) __attribute__((common)) = NULL`
- `void(*SSL_CTX_free_d)(SSL_CTX *ctx) __attribute__((common)) = NULL`
- `int(*BN_num_bits_d)(const BIGNUM *) __attribute__((common)) = NULL`
- `int(*X509_get_ext_count_d)(X509 *x) __attribute__((common)) = NULL`
- `char *(*BN_bn2hex_d)(const BIGNUM *a) __attribute__((common)) = NULL`
- `int(*EVP_MD_size_d)(const EVP_MD *md) __attribute__((common)) = NULL`
- `unsigned long(*ERR_get_error_d)(void) __attribute__((common)) = NULL`
- `void(*CONF_modules_unload_d)(int all) __attribute__((common)) = NULL`
- `void(*HMAC_CTX_init_d)(HMAC_CTX *ctx) __attribute__((common)) = NULL`
- `void(*SSL_load_error_strings_d)(void) __attribute__((common)) = NULL`
- `const EVP_MD *(*EVP_ripemd160_d)(void) __attribute__((common)) = NULL`
- `const char *(*SSLeay_version_d)(int t) __attribute__((common)) = NULL`
- `BIO *(*SSL_get_wbio_d)(const SSL *ssl) __attribute__((common)) = NULL`
- `void(*EC_GROUP_free_d)(EC_GROUP *group) __attribute__((common)) = NULL`
- `void(*EC_POINT_free_d)(EC_POINT *point) __attribute__((common)) = NULL`
- `int(*EC_KEY_generate_key_d)(EC_KEY *key) __attribute__((common)) = NULL`
- `void(*ASN1_STRING_TABLE_cleanup_d)(void) __attribute__((common)) = NULL`
- `void(*HMAC_CTX_cleanup_d)(HMAC_CTX *ctx) __attribute__((common)) = NULL`
- `int(*SSL_get_shutdown_d)(const SSL *ssl) __attribute__((common)) = NULL`
- `void(*CRYPTO_cleanup_all_ex_data_d)(void) __attribute__((common)) = NULL`
- `void(*EVP_MD_CTX_init_d)(EVP_MD_CTX *ctx) __attribute__((common)) = NULL`
- `int(*EC_KEY_check_key_d)(const EC_KEY *key) __attribute__((common)) = NULL`
- `int(*EVP_MD_CTX_cleanup_d)(EVP_MD_CTX *ctx) __attribute__((common)) = NULL`
- `void(*ERR_remove_state_d)(unsigned long pid) __attribute__((common)) = NULL`
- `int(*SSL_peek_d)(SSL *ssl, void *buf, int num) __attribute__((common)) = NULL`
- `X509_NAME *(*X509_get_subject_name_d)(X509 *a) __attribute__((common)) = NULL`
- `EC_KEY *(*EC_KEY_new_by_curve_name_d)(int nid) __attribute__((common)) = NULL`
- `int(*BN_hex2bn_d)(BIGNUM **a, const char *str) __attribute__((common)) = NULL`
- `int(*SSL_read_d)(SSL *ssl, void *buf, int num) __attribute__((common)) = NULL`
- `int(*RAND_bytes_d)(unsigned char *buf, int num) __attribute__((common)) = NULL`
- `void(*EVP_CIPHER_CTX_init_d)(EVP_CIPHER_CTX *a) __attribute__((common)) = NULL`
- `int(*EVP_CIPHER_nid_d)(const EVP_CIPHER *cipher) __attribute__((common)) = NULL`
- `void(*OPENSSL_add_all_algorithms_noconf_d)(void) __attribute__((common)) = NULL`
- `int(*SSL_get_error_d)(const SSL *s, int ret_code) __attribute__((common)) = NULL`
- `const SSL_METHOD *(*SSLv23_server_method_d)(void) __attribute__((common)) = NULL`
- `X509 *(*SSL_get_peer_certificate_d)(const SSL *s) __attribute__((common)) = NULL`
- `int(*EVP_CIPHER_CTX_cleanup_d)(EVP_CIPHER_CTX *a) __attribute__((common)) = NULL`
- `BIO *(*BIO_new_socket_d)(int sock, int close_flag) __attribute__((common)) = NULL`
- `EC_GROUP *(*EC_GROUP_new_by_curve_name_d)(int nid) __attribute__((common)) = NULL`
- `EC_POINT *(*EC_POINT_new_d)(const EC_GROUP *group) __attribute__((common)) = NULL`
- `int(*BN_bn2bin_d)(const BIGNUM *, unsigned char *) __attribute__((common)) = NULL`
- `X509_EXTENSION *(*X509_get_ext_d)(X509 *x, int loc) __attribute__((common)) = NULL`
- `SSL_CTX *(*SSL_CTX_new_d)(const SSL_METHOD *method) __attribute__((common)) = NULL`
- `void(*SSL_set_bio_d)(SSL *ssl, BIO *rbio, BIO *wbio) __attribute__((common)) = NULL`

- `int(* SSL_CTX_check_private_key_d)(const SSL_CTX *ctx) __attribute__((common)) = NULL`
- `int(* SSL_write_d)(SSL *ssl, const void *buf, int num) __attribute__((common)) = NULL`
- `void(* sk_pop_free_d)(_STACK *st, void(*func)(void *)) __attribute__((common)) = NULL`
- `int(* EVP_CIPHER_iv_length_d)(const EVP_CIPHER *cipher) __attribute__((common)) = NULL`
- `char *(* ERR_error_string_d)(unsigned long e, char *buf) __attribute__((common)) = NULL`
- `int(* EVP_CIPHER_block_size_d)(const EVP_CIPHER *cipher) __attribute__((common)) = NULL`
- `int(* EVP_CIPHER_key_length_d)(const EVP_CIPHER *cipher) __attribute__((common)) = NULL`
- `const EC_GROUP *(* EC_KEY_get0_group_d)(const EC_KEY *key) __attribute__((common)) = NULL`
- `const EVP_MD *(* EVP_get_digestbyname_d)(const char *name) __attribute__((common)) = NULL`
- `int(* SSL_CTX_set_cipher_list_d)(SSL_CTX *, const char *str) __attribute__((common)) = NULL`
- `int(* EVP_CIPHER_CTX_iv_length_d)(const EVP_CIPHER_CTX *ctx) __attribute__((common)) = NULL`
- `int(* EVP_DigestInit_d)(EVP_MD_CTX *ctx, const EVP_MD *type) __attribute__((common)) = NULL`
- `int(* EC_KEY_set_group_d)(EC_KEY *key, const EC_GROUP *group) __attribute__((common)) = NULL`
- `int(* EVP_CIPHER_CTX_block_size_d)(const EVP_CIPHER_CTX *ctx) __attribute__((common)) = NULL`
- `int(* EVP_CIPHER_CTX_key_length_d)(const EVP_CIPHER_CTX *ctx) __attribute__((common)) = NULL`
- `int(* RAND_load_file_d)(const char *filename, long max_bytes) __attribute__((common)) = NULL`
- `const BIGNUM *(* EC_KEY_get0_private_key_d)(const EC_KEY *key) __attribute__((common)) = NULL`
- `const EVP_CIPHER *(* EVP_get_cipherbyname_d)(const char *name) __attribute__((common)) = NULL`
- `const EC_POINT *(* EC_KEY_get0_public_key_d)(const EC_KEY *key) __attribute__((common)) = NULL`
- `int(* EC_GROUP_precompute_mult_d)(EC_GROUP *group, BN_CTX *ctx) __attribute__((common)) = NULL`
- `int(* EC_KEY_set_private_key_d)(EC_KEY *key, const BIGNUM *prv) __attribute__((common)) = NULL`
- `int(* EVP_CIPHER_CTX_set_padding_d)(EVP_CIPHER_CTX *c, int pad) __attribute__((common)) = NULL`
- `int(* BIO_vprintf_d)(BIO *bio, const char *format, va_list args) __attribute__((common)) = NULL`
- `int(* EC_KEY_set_public_key_d)(EC_KEY *key, const EC_POINT *pub) __attribute__((common)) = NULL`
- `void(* CRYPTO_set_id_callback_d)(unsigned long(*id_function)(void)) __attribute__((common)) = NULL`
- `long(* SSL_CTX_ctrl_d)(SSL_CTX *ctx, int cmd, long larg, void *parg) __attribute__((common)) = NULL`
- `void(* ERR_error_string_n_d)(unsigned long e, char *buf, size_t len) __attribute__((common)) = NULL`
- `BIGNUM *(* BN_bin2bn_d)(const unsigned char *s, int len, BIGNUM *ret) __attribute__((common)) = NULL`
- `int(* EVP_DigestUpdate_d)(EVP_MD_CTX *ctx, const void *d, size_t cnt) __attribute__((common)) = NULL`
- `int(* HMAC_Final_d)(HMAC_CTX *ctx, unsigned char *md, unsigned int *len) __attribute__((common)) = NULL`
- `int(* HMAC_Update_d)(HMAC_CTX *ctx, const unsigned char *data, size_t len) __attribute__((common)) = NULL`
- `int(* SSL_CTX_use_certificate_chain_file_d)(SSL_CTX *ctx, const char *file) __attribute__((common)) = NULL`
- `int(* EVP_DigestFinal_d)(EVP_MD_CTX *ctx, unsigned char *md, unsigned int *s) __attribute__((common)) = NULL`
- `int(* EVP_DigestInit_ex_d)(EVP_MD_CTX *ctx, const EVP_MD *type, ENGINE *impl) __attribute__((common)) = NULL`
- `int(* SSL_CTX_use_PrivateKey_file_d)(SSL_CTX *ctx, const char *file, int type) __attribute__((common)) = NULL`

- `int(* X509_NAME_get_text_by_NID_d)(X509_NAME *name, int nid, char *buf, int len)`
 __attribute__((common)) = NULL
- `int(* EVP_DigestFinal_ex_d)(EVP_MD_CTX *ctx, unsigned char *md, unsigned int *s)`
 __attribute__((common)) = NULL
- `int(* EVP_EncryptFinal_ex_d)(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl)`
 __attribute__((common)) = NULL
- `int(* EVP_DecryptFinal_ex_d)(EVP_CIPHER_CTX *ctx, unsigned char *outm, int *outl)`
 __attribute__((common)) = NULL
- `void(* EC_GROUP_set_point_conversion_form_d)(EC_GROUP *, point_conversion_form_t)`
 __attribute__((common)) = NULL
- `int(* DH_generate_parameters_ex_d)(DH *dh, int prime_len, int generator, BN_GENCB *cb)`
 __attribute__((common)) = NULL
- `EC_POINT *(* EC_POINT_hex2point_d)(const EC_GROUP *, const char *, EC_POINT *, BN_CTX *)`
 __attribute__((common)) = NULL
- `int(* SSL_CTX_load_verify_locations_d)(SSL_CTX *ctx, const char *CAfile, const char *CApath)`
 __attribute__((common)) = NULL
- `int(* HMAC_Init_ex_d)(HMAC_CTX *ctx, const void *key, int len, const EVP_MD *md, ENGINE *impl)`
 __attribute__((common)) = NULL
- `void(* SSL_CTX_set_tmp_dh_callback_d)(SSL_CTX *ctx, DH *(*dh)(SSL *ssl, int is_export, int keylength))`
 __attribute__((common)) = NULL
- `char *(* EC_POINT_point2hex_d)(const EC_GROUP *, const EC_POINT *, point_conversion_form_t form, BN_CTX *)`
 __attribute__((common)) = NULL
- `void(* CRYPTO_set_locking_callback_d)(void(*locking_function)(int mode, int n, const char *file, int line))`
 __attribute__((common)) = NULL
- `void(* SSL_CTX_set_tmp_ecdh_callback_d)(SSL_CTX *ctx, EC_KEY *(*ecdh)(SSL *ssl, int is_export, int keylength))`
 __attribute__((common)) = NULL
- `int(* EVP_DecryptUpdate_d)(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, const unsigned char *in, int inl)`
 __attribute__((common)) = NULL
- `int(* EVP_EncryptUpdate_d)(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, const unsigned char *in, int inl)`
 __attribute__((common)) = NULL
- `int(* EC_POINT_oct2point_d)(const EC_GROUP *group, EC_POINT *p, const unsigned char *buf, size_t len, BN_CTX *ctx)`
 __attribute__((common)) = NULL
- `int(* EVP_Digest_d)(const void *data, size_t count, unsigned char *md, unsigned int *size, const EVP_MD *type, ENGINE *impl)`
 __attribute__((common)) = NULL
- `int(* EVP_DecryptInit_ex_d)(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *cipher, ENGINE *impl, const unsigned char *key, const unsigned char *iv)`
 __attribute__((common)) = NULL
- `int(* EVP_EncryptInit_ex_d)(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *cipher, ENGINE *impl, const unsigned char *key, const unsigned char *iv)`
 __attribute__((common)) = NULL
- `size_t(* EC_POINT_point2oct_d)(const EC_GROUP *group, const EC_POINT *p, point_conversion_form_t form, unsigned char *buf, size_t len, BN_CTX *ctx)`
 __attribute__((common)) = NULL
- `int(* ECDH_compute_key_d)(void *out, size_t outlen, const EC_POINT *pub_key, EC_KEY *ecdh, void *(*KDF)(const void *in, size_t inlen, void *out, size_t *outlen))`
 __attribute__((common)) = NULL
- `png_uint_32(* png_access_version_number_d)(void)`
 __attribute__((common)) = NULL
- *PNG.* `void(* SPF_server_free_d)(SPF_server_t *sp)`
 __attribute__((common)) = NULL
- *SPF.* `void(* SPF_request_free_d)(SPF_request_t *sr)`
 __attribute__((common)) = NULL
- `void(* SPF_response_free_d)(SPF_response_t *rp)`
 __attribute__((common)) = NULL
- `const char *(* SPF_strerror_d)(SPF_reason_t reason)`
 __attribute__((common)) = NULL
- `const char *(* SPF_strerror_d)(SPF_result_t result)`
 __attribute__((common)) = NULL
- `const char *(* SPF_strerror_d)(SPF_errcode_t spf_err)`
 __attribute__((common)) = NULL
- `SPF_reason_t(* SPF_response_reason_d)(SPF_response_t *rp)`
 __attribute__((common)) = NULL
- `SPF_result_t(* SPF_response_result_d)(SPF_response_t *rp)`
 __attribute__((common)) = NULL
- `SPF_request_t *(* SPF_request_new_d)(SPF_server_t *spf_server)`
 __attribute__((common)) = NULL
- `void(* SPF_get_lib_version_d)(int *major, int *minor, int *patch)`
 __attribute__((common)) = NULL

- `int(* SPF_request_set_env_from_d)(SPF_request_t *sr, const char *from) __attribute__((common)) = NULL`
- `SPF_server_t>(* SPF_server_new_d)(SPF_server_dnstype_t dnstype, int debug) __attribute__((common)) = NULL`
- `SPF_errcode_t(* SPF_request_set_helo_dom_d)(SPF_request_t *sr, const char *dom) __attribute__((common)) = NULL`
- `SPF_errcode_t(* SPF_request_set_ipv4_d)(SPF_request_t *sr, struct in_addr addr) __attribute__((common)) = NULL`
- `SPF_errcode_t(* SPF_request_set_ipv6_d)(SPF_request_t *sr, struct in6_addr addr) __attribute__((common)) = NULL`
- `SPF_dns_server_t(* SPF_dns_zone_new_d)(SPF_dns_server_t *layer_below, const char *name, int debug) __attribute__((common)) = NULL`
- `SPF_errcode_t(* SPF_request_query_mailfrom_d)(SPF_request_t *spf_request, SPF_response_t **spf_responsep) __attribute__((common)) = NULL`
- `SPF_errcode_t(* SPF_dns_zone_add_str_d)(SPF_dns_server_t *spf_dns_server, const char *domain, ns_type rr_type, SPF_dns_stat_t herrno, const char *data) __attribute__((common)) = NULL`
- `TCHDB>(* tchdbnew_d)(void) __attribute__((common)) = NULL`
- `void(* tcfree_d)(void *ptr) __attribute__((common)) = NULL`
- `void(* tchdbdel_d)(TCHDB *hdb) __attribute__((common)) = NULL`
- `bool(* tchdbsync_d)(TCHDB *hdb) __attribute__((common)) = NULL`
- `int(* tchdbecode_d)(TCHDB *hdb) __attribute__((common)) = NULL`
- `void(* tcndbdel_d)(TCNDB *tree) __attribute__((common)) = NULL`
- `bool(* tchdbclose_d)(TCHDB *hdb) __attribute__((common)) = NULL`
- `void(* tclistdel_d)(TCLIST *list) __attribute__((common)) = NULL`
- `TCNDB(* tcndbdup_d)(TCNDB *ndb) __attribute__((common)) = NULL`
- `bool(* tchdbsetmutex_d)(TCHDB *hdb) __attribute__((common)) = NULL`
- `uint64_t(* tchdbfsiz_d)(TCHDB *hdb) __attribute__((common)) = NULL`
- `uint64_t(* tchdbbrnum_d)(TCHDB *hdb) __attribute__((common)) = NULL`
- `uint64_t(* tcndbrnum_d)(TCNDB *ndb) __attribute__((common)) = NULL`
- `void(* tcndbiterinit_d)(TCNDB *ndb) __attribute__((common)) = NULL`
- `char(* tcndbiternext2_d)(TCNDB *ndb) __attribute__((common)) = NULL`
- `int(* tclistnum_d)(const TCLIST *list) __attribute__((common)) = NULL`
- `const char(* tchdberrmsg_d)(int ecode) __attribute__((common)) = NULL`
- `const char(* tchdbpath_d)(TCHDB *hdb) __attribute__((common)) = NULL`
- `TCLIST(* tctreekeys_d)(const TCTREE *tree) __attribute__((common)) = NULL`
- `TCLIST(* tctreevals_d)(const TCTREE *tree) __attribute__((common)) = NULL`
- `TCNDB(* tcndbnew2_d)(TCCMP cmp, void *cmpop) __attribute__((common)) = NULL`
- `bool(* tchdbdefrag_d)(TCHDB *hdb, int64_t step) __attribute__((common)) = NULL`
- `bool(* tchdbsetdfunit_d)(TCHDB *hdb, int32_t dfunit) __attribute__((common)) = NULL`
- `bool(* tchdbout_d)(TCHDB *hdb, const void *kbuf, int ksiz) __attribute__((common)) = NULL`
- `bool(* tcndbout_d)(TCNDB *ndb, const void *kbuf, int ksiz) __attribute__((common)) = NULL`
- `bool(* tchdbopen_d)(TCHDB *hdb, const char *path, int omode) __attribute__((common)) = NULL`
- `const void(* tclistval_d)(const TCLIST *list, int index, int *sp) __attribute__((common)) = NULL`
- `void(* tchdbget_d)(TCHDB *hdb, const void *kbuf, int ksiz, int *sp) __attribute__((common)) = NULL`
- `void(* tcndbget3_d)(TCNDB *ndb, const void *kbuf, int ksiz, int *sp) __attribute__((common)) = NULL`
- `void(* tcndbget_d)(TCNDB *ndb, const void *kbuf, int ksiz, int *sp) __attribute__((common)) = NULL`
- `TCLIST(* tcndbfwmkeys_d)(TCNDB *ndb, const void *pbuf, int psiz, int max) __attribute__((common)) = NULL`
- `bool(* tchdbtune_d)(TCHDB *hdb, int64_t bnum, int8_t apow, int8_t fpow, uint8_t opts) __attribute__((common)) = NULL`
- `bool(* tchdboptimize_d)(TCHDB *hdb, int64_t bnum, int8_t apow, int8_t fpow, uint8_t opts) __attribute__((common)) = NULL`

- `bool(* tcnadbputkeep_d)(TCNDB *ndb, const void *kbuf, int ksiz, const void *vbuf, int vsiz) __attribute__((common)) = NULL`
- `bool(* tchdbputasync_d)(TCHDB *hdb, const void *kbuf, int ksiz, const void *vbuf, int vsiz) __attribute__((common)) = NULL`
- `bool(* tcnadbgetboth_d)(TCNDB *ndb, const void *kbuf, int ksiz, void **rkbuf, int *rksiz, void **rvbuf, int *rvsiz) __attribute__((common)) = NULL`
- `const char>(* jansson_version_d)(void) __attribute__((common)) = NULL`
- `Jansson. int(* json_array_append_d)(json_t *array, json_t *value) __attribute__((common)) = NULL`
- `int(* json_array_insert_d)(json_t *array, size_t index, json_t *value) __attribute__((common)) = NULL`
- `int(* json_array_set_d)(json_t *array, size_t index, json_t *value) __attribute__((common)) = NULL`
- `json_t(* json_array_d)(void) __attribute__((common)) = NULL`
- `int(* json_array_append_new_d)(json_t *array, json_t *value) __attribute__((common)) = NULL`
- `int(* json_array_clear_d)(json_t *array) __attribute__((common)) = NULL`
- `int(* json_array_extend_d)(json_t *array, json_t *other) __attribute__((common)) = NULL`
- `json_t(* json_array_get_d)(const json_t *array, size_t index) __attribute__((common)) = NULL`
- `int(* json_array_insert_new_d)(json_t *array, size_t index, json_t *value) __attribute__((common)) = NULL`
- `int(* json_array_remove_d)(json_t *array, size_t index) __attribute__((common)) = NULL`
- `int(* json_array_set_new_d)(json_t *array, size_t index, json_t *value) __attribute__((common)) = NULL`
- `size_t(* json_array_size_d)(const json_t *array) __attribute__((common)) = NULL`
- `json_t(* json_copy_d)(json_t *value) __attribute__((common)) = NULL`
- `void(* json_decref_d)(json_t *json) __attribute__((common)) = NULL`
- `json_t(* json_deep_copy_d)(json_t *value) __attribute__((common)) = NULL`
- `void(* json_delete_d)(json_t *json) __attribute__((common)) = NULL`
- `int(* json_dump_file_d)(const json_t *json, const char *path, size_t flags) __attribute__((common)) = NULL`
- `int(* json_dumpf_d)(const json_t *json, FILE *output, size_t flags) __attribute__((common)) = NULL`
- `char(* json_dumps_d)(const json_t *json, size_t flags) __attribute__((common)) = NULL`
- `int(* json_equal_d)(json_t *value1, json_t *value2) __attribute__((common)) = NULL`
- `json_t(* json_false_d)(void) __attribute__((common)) = NULL`
- `const char(* json_type_string_d)(json_t *json) __attribute__((common)) = NULL`
- `json_t(* json_incref_d)(json_t *json) __attribute__((common)) = NULL`
- `json_t(* json_integer_d)(json_int_t value) __attribute__((common)) = NULL`
- `int(* json_integer_set_d)(json_t *integer, json_int_t value) __attribute__((common)) = NULL`
- `json_int_t(* json_integer_value_d)(const json_t *integer) __attribute__((common)) = NULL`
- `json_t(* json_load_file_d)(const char *path, size_t flags, json_error_t *error) __attribute__((common)) = NULL`
- `json_t(* json_loadf_d)(FILE *input, size_t flags, json_error_t *error) __attribute__((common)) = NULL`
- `json_t(* json_loads_d)(const char *input, size_t flags, json_error_t *error) __attribute__((common)) = NULL`
- `json_t(* json_null_d)(void) __attribute__((common)) = NULL`
- `double(* json_number_value_d)(const json_t *json) __attribute__((common)) = NULL`
- `json_t(* json_object_d)(void) __attribute__((common)) = NULL`
- `int(* json_object_clear_d)(json_t *object) __attribute__((common)) = NULL`
- `int(* json_object_del_d)(json_t *object, const char *key) __attribute__((common)) = NULL`
- `json_t(* json_object_get_d)(const json_t *object, const char *key) __attribute__((common)) = NULL`
- `void(* json_object_iter_d)(json_t *object) __attribute__((common)) = NULL`
- `void(* json_object_iter_at_d)(json_t *object, const char *key) __attribute__((common)) = NULL`
- `const char(* json_object_iter_key_d)(void *iter) __attribute__((common)) = NULL`
- `void(* json_object_iter_next_d)(json_t *object, void *iter) __attribute__((common)) = NULL`
- `int(* json_object_iter_set_d)(json_t *object, void *iter, json_t *value) __attribute__((common)) = NULL`
- `int(* json_object_iter_set_new_d)(json_t *object, void *iter, json_t *value) __attribute__((common)) = NULL`

- `json_t *(* json_object_iter_value_d)(void *iter) __attribute__((common)) = NULL`
- `int(* json_object_set_d)(json_t *object, const char *key, json_t *value) __attribute__((common)) = NULL`
- `int(* json_object_set_new_d)(json_t *object, const char *key, json_t *value) __attribute__((common)) = NULL`
- `int(* json_object_set_new_nocheck_d)(json_t *object, const char *key, json_t *value) __attribute__((common)) = NULL`
- `int(* json_object_set_nocheck_d)(json_t *object, const char *key, json_t *value) __attribute__((common)) = NULL`
- `size_t(* json_object_size_d)(const json_t *object) __attribute__((common)) = NULL`
- `int(* json_object_update_d)(json_t *object, json_t *other) __attribute__((common)) = NULL`
- `json_t *(* json_pack_d)(const char *fmt,...) __attribute__((common)) = NULL`
- `json_t *(* json_pack_ex_d)(json_error_t *error, size_t flags, const char *fmt,...) __attribute__((common)) = NULL`
- `json_t *(* json_real_d)(double value) __attribute__((common)) = NULL`
- `int(* json_real_set_d)(json_t *real, double value) __attribute__((common)) = NULL`
- `double(* json_real_value_d)(const json_t *real) __attribute__((common)) = NULL`
- `void(* json_set_alloc_funcs_d)(json_malloc_t malloc_fn, json_free_t free_fn) __attribute__((common)) = NULL`
- `json_t *(* json_string_d)(const char *value) __attribute__((common)) = NULL`
- `json_t *(* json_string_nocheck_d)(const char *value) __attribute__((common)) = NULL`
- `int(* json_string_set_d)(json_t *string, const char *value) __attribute__((common)) = NULL`
- `int(* json_string_set_nocheck_d)(json_t *string, const char *value) __attribute__((common)) = NULL`
- `const char *(* json_string_value_d)(const json_t *string) __attribute__((common)) = NULL`
- `json_t *(* json_true_d)(void) __attribute__((common)) = NULL`
- `int(* json_unpack_d)(json_t *root, const char *fmt,...) __attribute__((common)) = NULL`
- `int(* json_unpack_ex_d)(json_t *root, json_error_t *error, size_t flags, const char *fmt,...) __attribute__((common)) = NULL`
- `json_t *(* json_vpack_ex_d)(json_error_t *error, size_t flags, const char *fmt, va_list ap) __attribute__((common)) = NULL`
- `int(* json_vunpack_ex_d)(json_t *root, json_error_t *error, size_t flags, const char *fmt, va_list ap) __attribute__((common)) = NULL`
- `void(* xmlInitParser_d)(void) __attribute__((common)) = NULL`
- `void(* xmlMemoryDump_d)(void) __attribute__((common)) = NULL`
- `void(* xmlCleanupParser_d)(void) __attribute__((common)) = NULL`
- `void(* xmlCleanupGlobals_d)(void) __attribute__((common)) = NULL`
- `void(* xmlFreeDoc_d)(xmlDocPtr doc) __attribute__((common)) = NULL`
- `void(* xmlFreeNode_d)(xmlNodePtr cur) __attribute__((common)) = NULL`
- `xmlBufferPtr(* xmlBufferCreate_d)(void) __attribute__((common)) = NULL`
- `void(* xmlBufferFree_d)(xmlBufferPtr buf) __attribute__((common)) = NULL`
- `xmlParserCtxtPtr(* xmlNewParserCtxt_d)(void) __attribute__((common)) = NULL`
- `int(* xmlBufferLength_d)(const xmlBufferPtr buf) __attribute__((common)) = NULL`
- `void(* xmlFreeParserCtxt_d)(xmlParserCtxtPtr ctx) __attribute__((common)) = NULL`
- `void(* xmlXPathFreeObject_d)(xmlXPathObjectPtr obj) __attribute__((common)) = NULL`
- `void(* xmlXPathFreeContext_d)(xmlXPathContextPtr ctx) __attribute__((common)) = NULL`
- `xmlXPathContextPtr(* xmlXPathNewContext_d)(xmlDocPtr doc) __attribute__((common)) = NULL`
- `xmlNodePtr(* xmlNewNode_d)(xmlNsPtr ns, const xmlChar *name) __attribute__((common)) = NULL`
- `const xmlChar *(* xmlBufferContent_d)(const xmlBufferPtr buf) __attribute__((common)) = NULL`
- `xmlNodePtr(* xmlAddSibling_d)(xmlNodePtr cur, xmlNodePtr elem) __attribute__((common)) = NULL`
- `int(* xmlNodeBufGetContent_d)(xmlBufferPtr buffer, xmlNodePtr cur) __attribute__((common)) = NULL`
- `void(* xmlNodeSetContent_d)(xmlNodePtr cur, const xmlChar *content) __attribute__((common)) = NULL`
- `xmlChar *(* xmlEncodeEntitiesReentrant_d)(xmlDocPtr doc, const xmlChar *input) __attribute__((common)) = NULL`

- `void(* xmlDocDumpFormatMemory_d)(xmlDocPtr cur, xmlChar **mem, int *size, int format)
__attribute__((common)) = NULL`
- `xmlAttrPtr(* xmlSetProp_d)(xmlNodePtr node, const xmlChar *name, const xmlChar *value)
__attribute__((common)) = NULL`
- `xmlXPathObjectPtr(* xmlXPathEvalExpression_d)(const xmlChar *xpath, xmlXPathContextPtr ctx)
__attribute__((common)) = NULL`
- `int(* xmlXPathRegisterNs_d)(xmlXPathContextPtr ctxt, const xmlChar *prefix, const xmlChar *ns_uri)
__attribute__((common)) = NULL`
- `xmlDocPtr(* xmlCtxtReadMemory_d)(xmlParserCtxtPtr ctxt, const char *buffer, int size, const char *url,
const char *encoding, int options) __attribute__((common)) = NULL`
- `const char *(* zlibVersion_d)(void) __attribute__((common)) = NULL`
- `ZLIB. uLong(* compressBound_d)(uLong sourceLen) __attribute__((common)) = NULL`
- `int(* uncompress_d)(Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen)
__attribute__((common)) = NULL`
- `int(* compress2_d)(Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen, int level)
__attribute__((common)) = NULL`

Detailed Description

External function pointers/definitions.

Definition in file **symbols.h**.

Define Documentation

#define M_BIND(x)

```
Value: { \
                .name = #x, \
                .pointer = (void *)&x##_d \
            }
```

Definition at line 17 of file **symbols.h**.

Referenced by `lib_load_bzip()`, `lib_load_cache()`, `lib_load_clamav()`, `lib_load_dkim()`, `lib_load_dspam()`, `lib_load_freetype()`, `lib_load_gd()`, `lib_load_jansson()`, `lib_load_jpeg()`, `lib_load_lzo()`, `lib_load_mysql()`, `lib_load_openssl()`, `lib_load_png()`, `lib_load_spf()`, `lib_load_tokyo()`, `lib_load_xml()`, and `lib_load_zlib()`.

Function Documentation

char **xmlParserVersion_d __attribute__((common)) = NULL

TOKYO.

XML.

const char* STDCALL mysql_character_set_name (MYSQL * *mysql*)

Referenced by `lib_load_mysql()`.

STACK_OF (SSL_COMP)

Variable Documentation

int(* __lzo_init_v2_d)(unsigned, int, int, int, int, int, int, int, int, int) __attribute__((common)) = NULL

Referenced by lib_load_lzo().

void(* ASN1_STRING_TABLE_cleanup_d)(void) __attribute__((common)) = NULL

Referenced by ssl_stop().

BIO*(BIO_new_socket_d)(int sock, int close_flag) __attribute__((common)) = NULL

Referenced by ssl_alloc(), and ssl_client_create().

void(* BIO_sock_cleanup_d)(void) __attribute__((common)) = NULL

Referenced by ssl_stop().

int(* BIO_vprintf_d)(BIO *bio, const char *format, va_list args) __attribute__((common)) = NULL

BIGNUM*(BN_bin2bn_d)(const unsigned char *s, int len, BIGNUM *ret) __attribute__((common)) = NULL

Referenced by ecies_key_private().

int(* BN_bn2bin_d)(const BIGNUM *, unsigned char *) __attribute__((common)) = NULL

Referenced by ecies_key_private_bin().

char*(BN_bn2hex_d)(const BIGNUM *a) __attribute__((common)) = NULL

Referenced by ecies_key_private_hex().

void(* BN_free_d)(BIGNUM *a) __attribute__((common)) = NULL

Referenced by ecies_key_private().

int(* BN_hex2bn_d)(BIGNUM **a, const char *str) __attribute__((common)) = NULL

Referenced by ecies_key_private().

int(* BN_num_bits_d)(const BIGNUM *) __attribute__((common)) = NULL

int(* BZ2_bzBuffToBuffCompress_d)(char *dest, unsigned int *destLen, char *source, unsigned int sourceLen, int blockSize100k, int verbosity, int workFactor) __attribute__((common)) = NULL

Referenced by compress_bzip().

int(* BZ2_bzBuffToBuffDecompress_d)(char *dest, unsigned int *destLen, char *source, unsigned int sourceLen, int small, int verbosity) __attribute__((common)) = NULL

Referenced by decompress_bzip().

const char*(BZ2_bzlibVersion_d)(void) __attribute__((common)) = NULL

BZIP.

Referenced by lib_load_bzip().

int(* cl_countsigs_d)(const char *path, unsigned int countoptions, unsigned int *sigs) __attribute__((common)) = NULL

Referenced by virus_sigs_total().

int(* cl_engine_compile_d)(struct cl_engine *engine) __attribute__((common)) = NULL

Referenced by virus_engine_create().

int(* cl_engine_free_d)(struct cl_engine *engine) __attribute__((common)) = NULL

Referenced by virus_engine_create(), and virus_engine_destroy().

struct cl_engine*(cl_engine_new_d)(void) __attribute__((common)) = NULL

Referenced by virus_engine_create().

int(* cl_engine_set_num_d)(struct cl_engine *engine, enum cl_engine_field field, long long num) __attribute__((common)) = NULL

Referenced by virus_engine_create().

int(* cl_engine_set_str_d)(struct cl_engine *engine, enum cl_engine_field field, const char *str) __attribute__((common)) = NULL

Referenced by virus_engine_create().

int(* cl_init_d)(unsigned int initoptions) __attribute__((common)) = NULL

Referenced by virus_start().

int(* cl_load_d)(const char *path, struct cl_engine *engine, unsigned int *signo, unsigned int dboptions) __attribute__((common)) = NULL

Referenced by virus_engine_create().

const char*(* cl_retver_d)(void) __attribute__((common)) = NULL

Referenced by lib_version_clamav().

int(* cl_scandesc_d)(int desc, const char **virname, unsigned long int *scanned, const struct cl_engine *engine, unsigned int scanoptions) __attribute__((common)) = NULL

Referenced by virus_check().

void(* cl_shutdown_d)(void) __attribute__((common)) = NULL

CLAMAV.

Referenced by virus_stop().

int(* cl_statchkdir_d)(const struct cl_stat *dbstat) __attribute__((common)) = NULL

Referenced by virus_engine_refresh().

int(* cl_statfree_d)(struct cl_stat *dbstat) __attribute__((common)) = NULL

Referenced by virus_engine_refresh(), virus_start(), and virus_stop().

int(* cl_statinidir_d)(const char *dirname, struct cl_stat *dbstat) __attribute__((common)) = NULL

Referenced by virus_engine_refresh(), and virus_start().

const char*(* cl_strerror_d)(int clerror) __attribute__((common)) = NULL

Referenced by virus_check(), virus_engine_create(), virus_sigs_total(), and virus_start().

void(* COMP_zlib_cleanup_d)(void) __attribute__((common)) = NULL

Referenced by ssl_stop().

**int(* compress2_d)(Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen, int level)
__attribute__((common)) = NULL**

Referenced by compress_zlib().

uLong(* compressBound_d)(uLong sourceLen) __attribute__((common)) = NULL

Referenced by compress_zlib().

void(* CONF_modules_unload_d)(int all) __attribute__((common)) = NULL

Referenced by ssl_stop().

void(* CRYPTO_cleanup_all_ex_data_d)(void) __attribute__((common)) = NULL

Referenced by ssl_stop().

void(* CRYPTO_free_d)(void *) __attribute__((common)) = NULL

int(* CRYPTO_num_locks_d)(void) __attribute__((common)) = NULL

Referenced by ssl_start(), and ssl_stop().

**void(* CRYPTO_set_id_callback_d)(unsigned long(*id_function)(void)) __attribute__((common)) =
NULL**

Referenced by ssl_start(), and ssl_stop().

**void(* CRYPTO_set_locking_callback_d)(void(*locking_function)(int mode, int n, const char *file,
int line)) __attribute__((common)) = NULL**

Referenced by ssl_start(), and ssl_stop().

void(* DH_free_d)(DH *dh) __attribute__((common)) = NULL

Referenced by ssl_dh_exchange_callback().

**int(* DH_generate_parameters_ex_d)(DH *dh, int prime_len, int generator, BN_GENCB *cb)
__attribute__((common)) = NULL**

Referenced by ssl_dh_exchange_callback().

DH*(* DH_new_d)(void) __attribute__((common)) = NULL

OPENSSL.

Referenced by ssl_dh_exchange_callback().

DKIM_STAT(* dkim_body_d)(DKIM *dkim, u_char *buf, size_t len) __attribute__((common)) = NULL

**DKIM_STAT(* dkim_chunk_d)(DKIM *dkim, unsigned char *chunkp, size_t len)
__attribute__((common)) = NULL**

Referenced by dkim_check(), and dkim_create().

void(* dkim_close_d)(DKIM_LIB *lib) __attribute__((common)) = NULL

Referenced by dkim_stop().

DKIM_STAT(* dkim_eoh_d)(DKIM *dkim) __attribute__((common)) = NULL

DKIM.

DKIM_STAT(* dkim_eom_d)(DKIM *dkim, _Bool *testkey) __attribute__((common)) = NULL

Referenced by dkim_check(), and dkim_create().

DKIM_STAT(* dkim_free_d)(DKIM *dkim) __attribute__((common)) = NULL

Referenced by dkim_check(), and dkim_create().

const char*(* dkim_getresultstr_d)(DKIM_STAT result) __attribute__((common)) = NULL

Referenced by dkim_check(), and dkim_create().

**DKIM_STAT(* dkim_getsighdrx_d)(DKIM *dkim, u_char *buf, size_t len, size_t initial)
__attribute__((common)) = NULL**

Referenced by dkim_create(), and lib_load_dkim().

**DKIM_STAT(* dkim_header_d)(DKIM *dkim, u_char *hdr, size_t len) __attribute__((common)) =
NULL**

**DKIM_LIB(* dkim_init_d)(void *(*mallocf)(void *closure, size_t nbytes), void(*freef)(void *closure,
void *p)) __attribute__((common)) = NULL**

Referenced by dkim_start().

uint32_t(* dkim_libversion_d)(void) __attribute__((common)) = NULL

Referenced by lib_load_dkim().

DKIM>(* dkim_sign_d)(DKIM_LIB *libhandle, const unsigned char *id, void *memclosure, const dkim_sigkey_t secretkey, const unsigned char *selector, const unsigned char *domain, dkim_canon_t hdr_canon_alg, dkim_canon_t body_canon_alg, dkim_alg_t sign_alg, off_t length, DKIM_STAT *statp) __attribute__((common)) = NULL

Referenced by dkim_create().

DKIM>(* dkim_verify_d)(DKIM_LIB *libhandle, const unsigned char *id, void *memclosure, DKIM_STAT *statp) __attribute__((common)) = NULL

Referenced by dkim_check().

int(* dspam_attach_d)(DSPAM_CTX *CTX, void *dbh) __attribute__((common)) = NULL

Referenced by dspam_check(), and dspam_train().

DSPAM_CTX(* dspam_create_d)(const char *username, const char *group, const char *home, int operating_mode, u_int32_t flags) __attribute__((common)) = NULL

Referenced by dspam_check(), and dspam_train().

void(* dspam_destroy_d)(DSPAM_CTX *CTX) __attribute__((common)) = NULL

Referenced by dspam_check(), and dspam_train().

int(* dspam_detach_d)(DSPAM_CTX *CTX) __attribute__((common)) = NULL

Referenced by dspam_check(), and dspam_train().

int(* dspam_init_driver_d)(DRIVER_CTX *DTX) __attribute__((common)) = NULL

Referenced by dspam_start().

int(* dspam_process_d)(DSPAM_CTX *CTX, const char *message) __attribute__((common)) = NULL

Referenced by dspam_check(), and dspam_train().

int(* dspam_shutdown_driver_d)(DRIVER_CTX *DTX) __attribute__((common)) = NULL

Referenced by dspam_stop().

const char*(* dspam_version_d)(void) __attribute__((common)) = NULL

DSPAM.

Referenced by lib_version_dspam().

void(* EC_GROUP_free_d)(EC_GROUP *group) __attribute__((common)) = NULL

Referenced by ecies_group(), and ecies_stop().

EC_GROUP*(* EC_GROUP_new_by_curve_name_d)(int nid) __attribute__((common)) = NULL

Referenced by ecies_group().

int(* EC_GROUP_precompute_mult_d)(EC_GROUP *group, BN_CTX *ctx) __attribute__((common)) = NULL

Referenced by ecies_group().

void(* EC_GROUP_set_point_conversion_form_d)(EC_GROUP *, point_conversion_form_t) __attribute__((common)) = NULL

Referenced by ecies_group(), ecies_key_alloc(), and ssl_ecdh_exchange_callback().

int(* EC_KEY_check_key_d)(const EC_KEY *key) __attribute__((common)) = NULL

Referenced by ecies_key_public().

void(* EC_KEY_free_d)(EC_KEY *key) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), ecies_encrypt(), ecies_key_alloc(), ecies_key_create(), ecies_key_free(), ecies_key_private(), and ecies_key_public().

int(* EC_KEY_generate_key_d)(EC_KEY *key) __attribute__((common)) = NULL

Referenced by ecies_key_create().

const EC_GROUP*(* EC_KEY_get0_group_d)(const EC_KEY *key) __attribute__((common)) = NULL

Referenced by ecies_encrypt(), ecies_key_alloc(), ecies_key_public(), ecies_key_public_bin(), ecies_key_public_hex(), and ssl_ecdh_exchange_callback().

const BIGNUM*(* EC_KEY_get0_private_key_d)(const EC_KEY *key) __attribute__((common)) = NULL

Referenced by ecies_key_private_bin(), and ecies_key_private_hex().

const EC_POINT*(* EC_KEY_get0_public_key_d)(const EC_KEY *key) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), ecies_encrypt(), ecies_key_public_bin(), and ecies_key_public_hex().

EC_KEY*(* EC_KEY_new_by_curve_name_d)(int nid) __attribute__((common)) = NULL

Referenced by ecies_key_alloc(), and ssl_ecdh_exchange_callback().

EC_KEY*(* EC_KEY_new_d)(void) __attribute__((common)) = NULL

Referenced by ecies_key_alloc().

int(* EC_KEY_set_group_d)(EC_KEY *key, const EC_GROUP *group) __attribute__((common)) = NULL

Referenced by ecies_key_alloc().

int(* EC_KEY_set_private_key_d)(EC_KEY *key, const BIGNUM *prv) __attribute__((common)) = NULL

Referenced by ecies_key_private().

int(* EC_KEY_set_public_key_d)(EC_KEY *key, const EC_POINT *pub) __attribute__((common)) = NULL

Referenced by ecies_key_public().

void(* EC_POINT_free_d)(EC_POINT *point) __attribute__((common)) = NULL

Referenced by ecies_key_public().

EC_POINT*(* EC_POINT_hex2point_d)(const EC_GROUP *, const char *, EC_POINT *, BN_CTX *) __attribute__((common)) = NULL

Referenced by ecies_key_public().

EC_POINT*(* EC_POINT_new_d)(const EC_GROUP *group) __attribute__((common)) = NULL

Referenced by ecies_key_public().

int(* EC_POINT_oct2point_d)(const EC_GROUP *group, EC_POINT *p, const unsigned char *buf, size_t len, BN_CTX *ctx) __attribute__((common)) = NULL

Referenced by ecies_key_public().

char>(* EC_POINT_point2hex_d)(const EC_GROUP *, const EC_POINT *, point_conversion_form_t form, BN_CTX *) __attribute__((common)) = NULL

Referenced by ecies_key_public_hex().

size_t(* EC_POINT_point2oct_d)(const EC_GROUP *group, const EC_POINT *p, point_conversion_form_t form, unsigned char *buf, size_t len, BN_CTX *ctx) __attribute__((common)) = NULL

Referenced by ecies_encrypt(), and ecies_key_public_bin().

int(* ECDH_compute_key_d)(void *out, size_t outlen, const EC_POINT *pub_key, EC_KEY *ecdh, void *(*KDF)(const void *in, size_t inlen, void *out, size_t *outlen)) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), and ecies_encrypt().

void(* ENGINE_cleanup_d)(void) __attribute__((common)) = NULL

Referenced by ssl_stop().

char>(* ERR_error_string_d)(unsigned long e, char *buf) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), ecies_encrypt(), ecies_group(), ecies_key_alloc(), ecies_key_create(), ecies_key_private(), ecies_key_private_bin(), ecies_key_private_hex(), ecies_key_public(), ecies_key_public_bin(), ecies_key_public_hex(), and symmetric_decrypt().

void(* ERR_error_string_n_d)(unsigned long e, char *buf, size_t len) __attribute__((common)) = NULL

Referenced by ssl_error_string(), and ssl_read().

void(* ERR_free_strings_d)(void) __attribute__((common)) = NULL

Referenced by ssl_stop().

unsigned long(* ERR_get_error_d)(void) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), ecies_encrypt(), ecies_group(), ecies_key_alloc(), ecies_key_create(), ecies_key_private(), ecies_key_private_bin(), ecies_key_private_hex(), ecies_key_public(), ecies_key_public_bin(), ecies_key_public_hex(), ssl_error_string(), and symmetric_decrypt().

void(* ERR_remove_state_d)(unsigned long pid) __attribute__((common)) = NULL

Referenced by ssl_free(), ssl_stop(), and ssl_thread_stop().

int(* EVP_CIPHER_block_size_d)(const EVP_CIPHER *cipher) __attribute__((common)) = NULL

Referenced by cipher_block_length(), and ecies_encrypt().

int(* EVP_CIPHER_CTX_block_size_d)(const EVP_CIPHER_CTX *ctx) __attribute__((common)) = NULL

Referenced by symmetric_decrypt(), and symmetric_encrypt().

int(* EVP_CIPHER_CTX_cleanup_d)(EVP_CIPHER_CTX *a) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), ecies_encrypt(), symmetric_decrypt(), and symmetric_encrypt().

void(* EVP_CIPHER_CTX_init_d)(EVP_CIPHER_CTX *a) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), ecies_encrypt(), symmetric_decrypt(), and symmetric_encrypt().

int(* EVP_CIPHER_CTX_iv_length_d)(const EVP_CIPHER_CTX *ctx) __attribute__((common)) = NULL

Referenced by symmetric_decrypt(), and symmetric_encrypt().

int(* EVP_CIPHER_CTX_key_length_d)(const EVP_CIPHER_CTX *ctx) __attribute__((common)) = NULL

Referenced by symmetric_decrypt(), and symmetric_encrypt().

int(* EVP_CIPHER_CTX_set_padding_d)(EVP_CIPHER_CTX *c, int pad) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), and ecies_encrypt().

int(* EVP_CIPHER_iv_length_d)(const EVP_CIPHER *cipher) __attribute__((common)) = NULL

Referenced by cipher_vector_length().

int(* EVP_CIPHER_key_length_d)(const EVP_CIPHER *cipher) __attribute__((common)) = NULL

Referenced by cipher_key_length(), ecies_decrypt(), and ecies_encrypt().

int(* EVP_CIPHER_nid_d)(const EVP_CIPHER *cipher) __attribute__((common)) = NULL

Referenced by cipher_block_length(), cipher_key_length(), cipher_numeric_id(), and cipher_vector_length().

void(* EVP_cleanup_d)(void) __attribute__((common)) = NULL

Referenced by ssl_stop().

int(* EVP_DecryptFinal_ex_d)(EVP_CIPHER_CTX *ctx, unsigned char *outm, int *outl) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), and symmetric_decrypt().

int(* EVP_DecryptInit_ex_d)(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *cipher, ENGINE *impl, const unsigned char *key, const unsigned char *iv) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), and symmetric_decrypt().

int(* EVP_DecryptUpdate_d)(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, const unsigned char *in, int inl) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), and symmetric_decrypt().

int(* EVP_Digest_d)(const void *data, size_t count, unsigned char *md, unsigned int *size, const EVP_MD *type, ENGINE *impl) __attribute__((common)) = NULL

Referenced by ecies_envelope_derivation().

int(* EVP_DigestFinal_d)(EVP_MD_CTX *ctx, unsigned char *md, unsigned int *s) __attribute__((common)) = NULL

Referenced by digest_hash().

int(* EVP_DigestFinal_ex_d)(EVP_MD_CTX *ctx, unsigned char *md, unsigned int *s) __attribute__((common)) = NULL

int(* EVP_DigestInit_d)(EVP_MD_CTX *ctx, const EVP_MD *type) __attribute__((common)) = NULL

int(* EVP_DigestInit_ex_d)(EVP_MD_CTX *ctx, const EVP_MD *type, ENGINE *impl) __attribute__((common)) = NULL

Referenced by digest_hash().

int(* EVP_DigestUpdate_d)(EVP_MD_CTX *ctx, const void *d, size_t cnt) __attribute__((common)) = NULL

Referenced by digest_hash().

**int(* EVP_EncryptFinal_ex_d)(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl)
__attribute__((common)) = NULL**

Referenced by ecies_encrypt(), and symmetric_encrypt().

**int(* EVP_EncryptInit_ex_d)(EVP_CIPHER_CTX *ctx, const EVP_CIPHER *cipher, ENGINE *impl,
const unsigned char *key, const unsigned char *iv) __attribute__((common)) = NULL**

Referenced by ecies_encrypt(), and symmetric_encrypt().

**int(* EVP_EncryptUpdate_d)(EVP_CIPHER_CTX *ctx, unsigned char *out, int *outl, const unsigned
char *in, int inl) __attribute__((common)) = NULL**

Referenced by ecies_encrypt(), and symmetric_encrypt().

**const EVP_CIPHER*(* EVP_get_cipherbyname_d)(const char *name) __attribute__((common)) =
NULL**

Referenced by cipher_id(), cipher_name(), ecies_decrypt(), ecies_encrypt(), and ecies_start().

const EVP_MD*(* EVP_get_digestbyname_d)(const char *name) __attribute__((common)) = NULL

Referenced by digest_id(), digest_name(), ecies_decrypt(), ecies_encrypt(), and ecies_start().

const EVP_MD*(* EVP_md4_d)(void) __attribute__((common)) = NULL

Referenced by digest_md4().

const EVP_MD*(* EVP_md5_d)(void) __attribute__((common)) = NULL

Referenced by digest_md5().

int(* EVP_MD_CTX_cleanup_d)(EVP_MD_CTX *ctx) __attribute__((common)) = NULL

Referenced by digest_hash().

void(* EVP_MD_CTX_init_d)(EVP_MD_CTX *ctx) __attribute__((common)) = NULL

Referenced by digest_hash().

int(* EVP_MD_size_d)(const EVP_MD *md) __attribute__((common)) = NULL

Referenced by digest_hash(), and ecies_encrypt().

const EVP_MD*(* EVP_ripemd160_d)(void) __attribute__((common)) = NULL

Referenced by digest_ripemd160().

const EVP_MD*(* EVP_sha1_d)(void) __attribute__((common)) = NULL

Referenced by digest_sha1().

const EVP_MD*(* EVP_sha224_d)(void) __attribute__((common)) = NULL

Referenced by digest_sha224().

const EVP_MD*(* EVP_sha256_d)(void) __attribute__((common)) = NULL

Referenced by digest_sha256().

const EVP_MD*(* EVP_sha384_d)(void) __attribute__((common)) = NULL

Referenced by digest_sha384().

const EVP_MD*(* EVP_sha512_d)(void) __attribute__((common)) = NULL

Referenced by digest_sha512().

const EVP_MD*(* EVP_sha_d)(void) __attribute__((common)) = NULL

Referenced by digest_sha().

**void(* FT_Library_Version_Static_d)(FT_Int *amajor, FT_Int *aminor, FT_Int *apatch)
__attribute__((common)) = NULL**

FreeType.

Referenced by lib_version_freetype().

const char*(* gd_version_d)(void) __attribute__((common)) = NULL

GD.

Referenced by lib_version_gd().

void(* gdFree_d)(void *m) __attribute__((common)) = NULL

Referenced by register_captcha_generate().

int(* gdlmageColorResolve_d)(gdImagePtr im, int r, int g, int b) __attribute__((common)) = NULL

Referenced by register_captcha_generate(), and register_captcha_write_noise().

gdImagePtr(* gdlmageCreate_d)(int sx, int sy) __attribute__((common)) = NULL

Referenced by register_captcha_generate().

void(* gdlmageDestroy_d)(gdImagePtr im) __attribute__((common)) = NULL

Referenced by register_captcha_generate().

void(* gdlmageGifPtr_d)(gdImagePtr im, int *size) __attribute__((common)) = NULL

Referenced by register_captcha_generate().

void(* gdlmageJpegPtr_d)(gdImagePtr im, int *size, int quality) __attribute__((common)) = NULL

void(* gdlmageSetPixel_d)(gdImagePtr im, int x, int y, int color) __attribute__((common)) = NULL

Referenced by register_captcha_write_noise().

char(* gdlmageStringFT_d)(gdImage *im, int *brect, int fg, char *fontlist, double ptsize, double angle, int x, int y, char *string) __attribute__((common)) = NULL

Referenced by register_captcha_generate().

void(* HMAC_CTX_cleanup_d)(HMAC_CTX *ctx) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), and ecies_encrypt().

void(* HMAC_CTX_init_d)(HMAC_CTX *ctx) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), and ecies_encrypt().

int(* HMAC_Final_d)(HMAC_CTX *ctx, unsigned char *md, unsigned int *len) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), and ecies_encrypt().

int(* HMAC_Init_ex_d)(HMAC_CTX *ctx, const void *key, int len, const EVP_MD *md, ENGINE *impl) __attribute__((common)) = NULL

Referenced by ecies_decrypt(), and ecies_encrypt().

**int(* HMAC_Update_d)(HMAC_CTX *ctx, const unsigned char *data, size_t len)
__attribute__((common)) = NULL**

Referenced by ecies_decrypt(), and ecies_encrypt().

const char*(* jansson_version_d)(void) __attribute__((common)) = NULL

Jansson.

Referenced by lib_version_jansson().

const char*(* jpeg_version_d)(void) __attribute__((common)) = NULL

JPEG.

Referenced by lib_version_jpeg().

int(* json_array_append_d)(json_t *array, json_t *value) __attribute__((common)) = NULL

int(* json_array_append_new_d)(json_t *array, json_t *value) __attribute__((common)) = NULL

Referenced by portal_config_entry_flags(), portal_contact_detail_flags(), portal_endpoint_alert_list(), portal_endpoint_aliases(), portal_endpoint_contacts_list(), portal_endpoint_folders_list(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(), portal_endpoint_messages_tag(), portal_endpoint_messages_tags(), portal_message_attachments(), portal_message_flags_array(), and portal_message_tags_array().

int(* json_array_clear_d)(json_t *array) __attribute__((common)) = NULL

json_t*(* json_array_d)(void) __attribute__((common)) = NULL

Referenced by portal_config_entry_flags(), portal_contact_detail_flags(), portal_contact_details(), portal_endpoint_alert_list(), portal_endpoint_aliases(), portal_endpoint_contacts_list(), portal_endpoint_folders_list(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(), portal_endpoint_messages_tag(), portal_endpoint_messages_tags(), portal_message_attachments(), portal_message_flags_array(), and portal_message_tags_array().

int(* json_array_extend_d)(json_t *array, json_t *other) __attribute__((common)) = NULL

json_t*(* json_array_get_d)(const json_t *array, size_t index) __attribute__((common)) = NULL

Referenced by portal_endpoint_alert_acknowledge(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(), portal_endpoint_messages_tag(), portal_parse_flags(), portal_parse_json_str_array(), and portal_parse_sections().

int(* json_array_insert_d)(json_t *array, size_t index, json_t *value) __attribute__((common)) = NULL

int(* json_array_insert_new_d)(json_t *array, size_t index, json_t *value) __attribute__((common)) = NULL

int(* json_array_remove_d)(json_t *array, size_t index) __attribute__((common)) = NULL

int(* json_array_set_d)(json_t *array, size_t index, json_t *value) __attribute__((common)) = NULL

int(* json_array_set_new_d)(json_t *array, size_t index, json_t *value) __attribute__((common)) = NULL

size_t(* json_array_size_d)(const json_t *array) __attribute__((common)) = NULL

Referenced by portal_endpoint_alert_acknowledge(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(), portal_endpoint_messages_tag(), portal_parse_flags(), portal_parse_json_str_array(), and portal_parse_sections().

json_t>(* json_copy_d)(json_t *value) __attribute__((common)) = NULL

void(* json_decref_d)(json_t *json) __attribute__((common)) = NULL

Referenced by http_parse_context(), http_session_reset(), portal_config_collection(), portal_config_entry(), portal_config_entry_flags(), portal_contact_details(), portal_endpoint_alert_list(), portal_endpoint_aliases(), portal_endpoint_contacts_list(), portal_endpoint_error(), portal_endpoint_folders_list(), portal_endpoint_folders_tags(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(), portal_endpoint_messages_load(), portal_endpoint_messages_tag(), portal_endpoint_messages_tags(), portal_endpoint_response(), and portal_message_attachments().

json_t(* json_deep_copy_d)(json_t *value) __attribute__((common)) = NULL

void(* json_delete_d)(json_t *json) __attribute__((common)) = NULL

int(* json_dump_file_d)(const json_t *json, const char *path, size_t flags) __attribute__((common)) = NULL

int(* json_dumpf_d)(const json_t *json, FILE *output, size_t flags) __attribute__((common)) = NULL

char>(* json_dumps_d)(const json_t *json, size_t flags) __attribute__((common)) = NULL

Referenced by portal_endpoint_error(), and portal_endpoint_response().

int(* json_equal_d)(json_t *value1, json_t *value2) __attribute__((common)) = NULL

json_t>(* json_false_d)(void) __attribute__((common)) = NULL

json_t>(* json_incref_d)(json_t *json) __attribute__((common)) = NULL

json_t>(* json_integer_d)(json_int_t value) __attribute__((common)) = NULL

Referenced by portal_endpoint_folders_tags().

int(* json_integer_set_d)(json_t *integer, json_int_t value) __attribute__((common)) = NULL

json_int_t(* json_integer_value_d)(const json_t *integer) __attribute__((common)) = NULL

Referenced by portal_endpoint(), portal_endpoint_alert_acknowledge(), portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_move(), portal_endpoint_messages_remove(), and portal_endpoint_messages_tag().

json_t>(* json_load_file_d)(const char *path, size_t flags, json_error_t *error) __attribute__((common)) = NULL

json_t>(* json_loadf_d)(FILE *input, size_t flags, json_error_t *error) __attribute__((common)) = NULL

json_t>(* json_loads_d)(const char *input, size_t flags, json_error_t *error) __attribute__((common)) = NULL

Referenced by http_parse_context(), and portal_endpoint().

json_t>(* json_null_d)(void) __attribute__((common)) = NULL

double(* json_number_value_d)(const json_t *json) __attribute__((common)) = NULL

int(* json_object_clear_d)(json_t *object) __attribute__((common)) = NULL

json_t>(* json_object_d)(void) __attribute__((common)) = NULL

Referenced by portal_config_collection(), portal_endpoint_attachments_add(), portal_endpoint_folders_tags(), portal_endpoint_messages_compose(), portal_endpoint_messages_load(), portal_meta(), portal_settings_changepass(), and portal_settings_identity().

int(* json_object_del_d)(json_t *object, const char *key) __attribute__((common)) = NULL

json_t>(* json_object_get_d)(const json_t *object, const char *key) __attribute__((common)) = NULL

Referenced by http_parse_context(), portal_endpoint(), and portal_endpoint_contacts_add().

void>(* json_object_iter_at_d)(json_t *object, const char *key) __attribute__((common)) = NULL

void(* json_object_iter_d)(json_t *object) __attribute__((common)) = NULL

Referenced by portal_endpoint_config_edit(), portal_endpoint_contacts_add(), and
portal_endpoint_contacts_edit().

const char>(* json_object_iter_key_d)(void *iter) __attribute__((common)) = NULL

Referenced by portal_endpoint_config_edit(), portal_endpoint_contacts_add(), and
portal_endpoint_contacts_edit().

void(* json_object_iter_next_d)(json_t *object, void *iter) __attribute__((common)) = NULL

Referenced by portal_endpoint_config_edit(), portal_endpoint_contacts_add(), and
portal_endpoint_contacts_edit().

int(* json_object_iter_set_d)(json_t *object, void *iter, json_t *value) __attribute__((common)) = NULL

int(* json_object_iter_set_new_d)(json_t *object, void *iter, json_t *value) __attribute__((common)) = NULL

json_t(* json_object_iter_value_d)(void *iter) __attribute__((common)) = NULL

Referenced by portal_endpoint_config_edit(), portal_endpoint_contacts_add(), and
portal_endpoint_contacts_edit().

int(* json_object_set_d)(json_t *object, const char *key, json_t *value) __attribute__((common)) = NULL

int(* json_object_set_new_d)(json_t *object, const char *key, json_t *value) __attribute__((common)) = NULL

Referenced by portal_config_collection(), portal_contact_details(), portal_endpoint_contacts_list(),
portal_endpoint_folders_tags(), and portal_endpoint_messages_load().

int(* json_object_set_new_nocheck_d)(json_t *object, const char *key, json_t *value) __attribute__((common)) = NULL

int(* json_object_set_nocheck_d)(json_t *object, const char *key, json_t *value) __attribute__((common)) = NULL

size_t(* json_object_size_d)(const json_t *object) __attribute__((common)) = NULL

Referenced by portal_endpoint_contacts_edit(), portal_endpoint_folders_add(),
portal_endpoint_messages_flag(), portal_endpoint_messages_load(), portal_endpoint_messages_send(),
portal_endpoint_messages_tag(), and portal_validate_request().

int(* json_object_update_d)(json_t *object, json_t *other) __attribute__((common)) = NULL

json_t>(* json_pack_d)(const char *fmt,...) __attribute__((common)) = NULL

**json_t>(* json_pack_ex_d)(json_error_t *error, size_t flags, const char *fmt,...)
__attribute__((common)) = NULL**

Referenced by portal_config_entry(), portal_contact_details(), portal_endpoint_alert_list(),
portal_endpoint_aliases(), portal_endpoint_attachments_add(), portal_endpoint_contacts_list(),
portal_endpoint_error(), portal_endpoint_folders_list(), portal_endpoint_messages_compose(),
portal_endpoint_messages_copy(), portal_endpoint_messages_flag(), portal_endpoint_messages_list(),
portal_endpoint_messages_tag(), portal_message_attachments(), portal_message_body(),
portal_message_header(), portal_message_info(), portal_message_meta(), portal_message_security(),
portal_message_server(), portal_message_source(), portal_meta(), portal_settings_changepass(), and
portal_settings_identity().

json_t>(* json_real_d)(double value) __attribute__((common)) = NULL

int(* json_real_set_d)(json_t *real, double value) __attribute__((common)) = NULL

double(* json_real_value_d)(const json_t *real) __attribute__((common)) = NULL

**void(* json_set_alloc_funcs_d)(json_malloc_t malloc_fn, json_free_t free_fn)
__attribute__((common)) = NULL**

json_t>(* json_string_d)(const char *value) __attribute__((common)) = NULL

Referenced by portal_config_entry_flags(), portal_contact_detail_flags(), portal_endpoint_contacts_list(),
portal_endpoint_messages_list(), portal_endpoint_messages_tag(), portal_endpoint_messages_tags(),
portal_message_flags_array(), and portal_message_tags_array().

json_t>(* json_string_nocheck_d)(const char *value) __attribute__((common)) = NULL

int(* json_string_set_d)(json_t *string, const char *value) __attribute__((common)) = NULL

**int(* json_string_set_nocheck_d)(json_t *string, const char *value) __attribute__((common)) =
NULL**

const char>(* json_string_value_d)(const json_t *string) __attribute__((common)) = NULL

Referenced by http_parse_context(), portal_endpoint(), portal_endpoint_config_edit(),
portal_endpoint_contacts_add(), portal_endpoint_contacts_edit(), portal_endpoint_messages_tag(),
portal_parse_flags(), portal_parse_json_str_array(), and portal_parse_sections().

json_t>(* json_true_d)(void) __attribute__((common)) = NULL

Referenced by portal_endpoint_aliases().

const char*(* json_type_string_d)(json_t *json) __attribute__((common)) = NULL

Referenced by portal_endpoint(), and portal_endpoint_contacts_add().

int(* json_unpack_d)(json_t *root, const char *fmt,...) __attribute__((common)) = NULL

**int(* json_unpack_ex_d)(json_t *root, json_error_t *error, size_t flags, const char *fmt,...)
__attribute__((common)) = NULL**

Referenced by portal_endpoint_alert_acknowledge(), portal_endpoint_attachments_add(),
portal_endpoint_attachments_remove(), portal_endpoint_auth(), portal_endpoint_contacts_add(),
portal_endpoint_contacts_copy(), portal_endpoint_contacts_edit(), portal_endpoint_contacts_list(),
portal_endpoint_contacts_load(), portal_endpoint_contacts_move(), portal_endpoint_contacts_remove(),
portal_endpoint_folders_add(), portal_endpoint_folders_list(), portal_endpoint_folders_remove(),
portal_endpoint_folders_rename(), portal_endpoint_folders_tags(), portal_endpoint_messages_copy(),
portal_endpoint_messages_flag(), portal_endpoint_messages_list(), portal_endpoint_messages_load(),
portal_endpoint_messages_move(), portal_endpoint_messages_remove(), portal_endpoint_messages_send(),
portal_endpoint_messages_tag(), and portal_settings_changepass().

**json_t*(* json_vpack_ex_d)(json_error_t *error, size_t flags, const char *fmt, va_list ap)
__attribute__((common)) = NULL**

Referenced by portal_endpoint_response().

**int(* json_vunpack_ex_d)(json_t *root, json_error_t *error, size_t flags, const char *fmt, va_list ap)
__attribute__((common)) = NULL**

int(* lt_dlexit_d)(void) __attribute__((common)) = NULL

**int(* lzo1x_1_compress_d)(const lzo_byte *src, lzo_uint src_len, lzo_byte *dst, lzo_uintp dst_len,
lzo_voidp wrkmem) __attribute__((common)) = NULL**

Referenced by compress_lzo().

**int(* lzo1x_decompress_safe_d)(const lzo_byte *src, lzo_uint src_len, lzo_byte *dst, lzo_uintp
dst_len, lzo_voidp wrkmem) __attribute__((common)) = NULL**

Referenced by decompress_block_lzo(), and decompress_lzo().

**lzo_uint32(* lzo_adler32_d)(lzo_uint32 _adler, const lzo_bytew _buf, lzo_uint _len)
__attribute__((common)) = NULL**

const char*(* lzo_version_string_d)(void) __attribute__((common)) = NULL

LZO.

Referenced by lib_version_lzo().

memcached_return_t(* memcached_add_d)(memcached_st *ptr, const char *key, size_t key_length, const char *value, size_t value_length, time_t expiration, uint32_t flags) __attribute__((common)) = NULL

Referenced by cache_add(), cache_append(), and cache_silent_add().

memcached_return_t(* memcached_append_d)(memcached_st *ptr, const char *key, size_t key_length, const char *value, size_t value_length, time_t expiration, uint32_t flags) __attribute__((common)) = NULL

Referenced by cache_append().

memcached_return_t(* memcached_behavior_set_d)(memcached_st *ptr, const memcached_behavior_t flag, uint64_t data) __attribute__((common)) = NULL

Referenced by cache_start().

memcached_return_t(* memcached_cas_d)(memcached_st *ptr, const char *key, size_t key_length, const char *value, size_t value_length, time_t expiration, uint32_t flags, uint64_t cas) __attribute__((common)) = NULL

memcached_st(* memcached_create_d)(memcached_st *ptr) __attribute__((common)) = NULL

Referenced by cache_start().

memcached_return_t(* memcached_decrement_d)(memcached_st *ptr, const char *key, size_t key_length, uint32_t offset, uint64_t *value) __attribute__((common)) = NULL

memcached_return_t(* memcached_decrement_with_initial_d)(memcached_st *ptr, const char *key, size_t key_length, uint64_t offset, uint64_t initial, time_t expiration, uint64_t *value) __attribute__((common)) = NULL

memcached_return_t(* memcached_delete_d)(memcached_st *ptr, const char *key, size_t key_length, time_t expiration) __attribute__((common)) = NULL

Referenced by cache_delete().

memcached_return_t(* memcached_flush_d)(memcached_st *ptr, time_t expiration) __attribute__((common)) = NULL

MEMCACHED.

Referenced by cache_flush().

void(* memcached_free_d)(memcached_st *ptr) __attribute__((common)) = NULL

Referenced by cache_start(), and cache_stop().

char*(* memcached_get_d)(memcached_st *ptr, const char *key, size_t key_length, size_t *value_length, uint32_t *flags, memcached_return_t *error) __attribute__((common)) = NULL

Referenced by cache_get(), and cache_get_u64().

memcached_return_t(* memcached_increment_d)(memcached_st *ptr, const char *key, size_t key_length, uint32_t offset, uint64_t *value) __attribute__((common)) = NULL

memcached_return_t(* memcached_increment_with_initial_d)(memcached_st *ptr, const char *key, size_t key_length, uint64_t offset, uint64_t initial, time_t expiration, uint64_t *value) __attribute__((common)) = NULL

Referenced by cache_increment().

const char*(* memcached_lib_version_d)(void) __attribute__((common)) = NULL

Referenced by lib_version_cache().

memcached_return_t(* memcached_prepend_d)(memcached_st *ptr, const char *key, size_t key_length, const char *value, size_t value_length, time_t expiration, uint32_t flags) __attribute__((common)) = NULL

memcached_return_t(* memcached_replace_d)(memcached_st *ptr, const char *key, size_t key_length, const char *value, size_t value_length, time_t expiration, uint32_t flags) __attribute__((common)) = NULL

memcached_return_t(* memcached_server_add_with_weight_d)(memcached_st *ptr, const char *hostname, in_port_t port, uint32_t weight) __attribute__((common)) = NULL

Referenced by cache_start().

memcached_return_t(* memcached_set_d)(memcached_st *ptr, const char *key, size_t key_length, const char *value, size_t value_length, time_t expiration, uint32_t flags) __attribute__((common)) = NULL

Referenced by cache_set(), and cache_set_u64().

const char*(* memcached_strerror_d)(const memcached_st *ptr, memcached_return_t rc) __attribute__((common)) = NULL

Referenced by cache_add(), cache_append(), cache_delete(), cache_flush(), cache_get(), cache_get_u64(), cache_increment(), cache_set(), cache_set_u64(), and cache_start().

void(* my_once_free_d)(void) __attribute__((common)) = NULL

MYSQL.

Referenced by sql_stop().

my_ulonglong(* mysql_affected_rows_d)(MYSQL *mysql) __attribute__((common)) = NULL

const char*(* mysql_character_set_name_d)(MYSQL *mysql) __attribute__((common)) = NULL

Referenced by serv_charset_mysql().

void(* mysql_close_d)(MYSQL *mysql) __attribute__((common)) = NULL

Referenced by serv_charset_mysql(), serv_schema_mysql(), serv_version_mysql(), sql_open(), and sql_stop().

my_bool(* mysql_embedded_d)(void) __attribute__((common)) = NULL

Referenced by serv_type_mysql().

unsigned int(* mysql_errno_d)(MYSQL *mysql) __attribute__((common)) = NULL

Referenced by sql_errno(), sql_error(), stmt_errno(), and stmt_error().

const char*(* mysql_error_d)(MYSQL *mysql) __attribute__((common)) = NULL

Referenced by sql_error(), and stmt_error().

unsigned long(* mysql_escape_string_d)(char *to, const char *from, unsigned long length) __attribute__((common)) = NULL

MYSQL_FIELD(* mysql_fetch_field_d)(MYSQL_RES *result) __attribute__((common)) = NULL

Referenced by res_bind_create().

MYSQL_ROW(* mysql_fetch_row_d)(MYSQL_RES *result) __attribute__((common)) = NULL

void(* mysql_free_result_d)(MYSQL_RES *result) __attribute__((common)) = NULL

Referenced by res_bind_create().

unsigned long(* mysql_get_client_version_d)(void) __attribute__((common)) = NULL

Referenced by lib_version_mysql().

const char*(* mysql_get_server_info_d)(MYSQL *mysql) __attribute__((common)) = NULL

Referenced by serv_version_mysql().

MYSQL(* mysql_init_d)(MYSQL *mysql) __attribute__((common)) = NULL

Referenced by sql_open().

my_ulonglong(* mysql_insert_id_d)(MYSQL *mysql) __attribute__((common)) = NULL

unsigned int(* mysql_num_fields_d)(MYSQL_RES *result) __attribute__((common)) = NULL

Referenced by res_bind_create().

my_ulonglong(* mysql_num_rows_d)(MYSQL_RES *result) __attribute__((common)) = NULL

**int(* mysql_options_d)(MYSQL *mysql, enum mysql_option option, const void *arg)
__attribute__((common)) = NULL**

Referenced by sql_open().

int(* mysql_ping_d)(MYSQL *mysql) __attribute__((common)) = NULL

Referenced by sql_ping().

**MYSQL*(* mysql_real_connect_d)(MYSQL *mysql, const char *name, const char *user, const char
*passwd, const char *db, unsigned int port, const char *unix_socket, unsigned long client_flag)
__attribute__((common)) = NULL**

Referenced by sql_open().

**int(* mysql_real_query_d)(MYSQL *mysql, const char *query, unsigned long length)
__attribute__((common)) = NULL**

Referenced by sql_query_conn().

void(* mysql_server_end_d)(void) __attribute__((common)) = NULL

Referenced by sql_stop().

int(* mysql_server_init_d)(int argc, char **argv, char **groups) __attribute__((common)) = NULL

Referenced by sql_start().

**int(* mysql_set_character_set_d)(MYSQL *mysql, const char *csname) __attribute__((common)) =
NULL**

**my_ulonglong(* mysql_stmt_affected_rows_d)(MYSQL_STMT *stmt) __attribute__((common)) =
NULL**

Referenced by stmt_exec_affected_conn().

my_bool(* mysql_stmt_attr_set_d)(MYSQL_STMT *stmt, enum enum_stmt_attr_type attr_type, const void *attr) __attribute__((common)) = NULL

Referenced by stmt_prepare().

my_bool(* mysql_stmt_bind_param_d)(MYSQL_STMT *stmt, MYSQL_BIND *bind) __attribute__((common)) = NULL

Referenced by stmt_bind_param().

my_bool(* mysql_stmt_bind_result_d)(MYSQL_STMT *stmt, MYSQL_BIND *bind) __attribute__((common)) = NULL

Referenced by res_bind_free(), and res_stmt_store().

my_bool(* mysql_stmt_close_d)(MYSQL_STMT *) __attribute__((common)) = NULL

Referenced by stmt_close().

unsigned int(* mysql_stmt_errno_d)(MYSQL_STMT *stmt) __attribute__((common)) = NULL

Referenced by mail_db_update_message_folder(), stmt_errno(), and stmt_error().

const char*(* mysql_stmt_error_d)(MYSQL_STMT *stmt) __attribute__((common)) = NULL

Referenced by mail_db_update_message_folder(), res_bind_create(), res_stmt_store(), and stmt_error().

int(* mysql_stmt_execute_d)(MYSQL_STMT *stmt) __attribute__((common)) = NULL

Referenced by stmt_exec_affected_conn(), stmt_exec_conn(), stmt_get_result_conn(), and stmt_insert_conn().

int(* mysql_stmt_fetch_d)(MYSQL_STMT *stmt) __attribute__((common)) = NULL

Referenced by res_stmt_store().

my_bool(* mysql_stmt_free_result_d)(MYSQL_STMT *stmt) __attribute__((common)) = NULL

Referenced by res_stmt_store().

MYSQL_STMT*(* mysql_stmt_init_d)(MYSQL *mysql) __attribute__((common)) = NULL

Referenced by stmt_open().

my_ulonglong(* mysql_stmt_insert_id_d)(MYSQL_STMT *stmt) __attribute__((common)) = NULL

Referenced by stmt_insert_conn().

my_ulonglong(* mysql_stmt_num_rows_d)(MYSQL_STMT *stmt) __attribute__((common)) = NULL

Referenced by res_stmt_store().

**int(* mysql_stmt_prepare_d)(MYSQL_STMT *stmt, const char *query, unsigned long length)
__attribute__((common)) = NULL**

Referenced by stmt_prepare().

my_bool(* mysql_stmt_reset_d)(MYSQL_STMT *stmt) __attribute__((common)) = NULL

Referenced by stmt_reset().

**MYSQL_RES(* mysql_stmt_result_metadata_d)(MYSQL_STMT *stmt) __attribute__((common)) =
NULL**

Referenced by res_bind_create().

int(* mysql_stmt_store_result_d)(MYSQL_STMT *stmt) __attribute__((common)) = NULL

Referenced by res_stmt_store().

MYSQL_RES(* mysql_store_result_d)(MYSQL *mysql) __attribute__((common)) = NULL

void(* mysql_thread_end_d)(void) __attribute__((common)) = NULL

Referenced by sql_thread_stop().

unsigned long(* mysql_thread_id_d)(MYSQL *mysql) __attribute__((common)) = NULL

Referenced by sql_ping().

my_bool(* mysql_thread_init_d)(void) __attribute__((common)) = NULL

Referenced by sql_thread_start().

unsigned int(* mysql_thread_safe_d)(void) __attribute__((common)) = NULL

Referenced by sql_start().

void(* OBJ_cleanup_d)(void) __attribute__((common)) = NULL

Referenced by ssl_stop().

void(* OBJ_NAME_cleanup_d)(int type) __attribute__((common)) = NULL

Referenced by ssl_stop().

const char*(OBJ_nid2sn_d)(int n) __attribute__((common)) = NULL

Referenced by cipher_block_length(), cipher_id(), cipher_key_length(), cipher_vector_length(), digest_id(), ecies_decrypt(), ecies_encrypt(), and ecies_start().

void(* OPENSSL_add_all_algorithms_noconf_d)(void) __attribute__((common)) = NULL

Referenced by ssl_start().

png_uint_32(* png_access_version_number_d)(void) __attribute__((common)) = NULL

PNG.

Referenced by lib_version_png().

int(* RAND_bytes_d)(unsigned char *buf, int num) __attribute__((common)) = NULL

Referenced by rand_choices(), rand_get_int16(), rand_get_int32(), rand_get_int64(), rand_get_int8(), rand_get_uint16(), rand_get_uint32(), rand_get_uint64(), rand_get_uint8(), and rand_write().

void(* RAND_cleanup_d)(void) __attribute__((common)) = NULL

Referenced by rand_stop().

int(* RAND_load_file_d)(const char *filename, long max_bytes) __attribute__((common)) = NULL

Referenced by rand_start().

int(* RAND_status_d)(void) __attribute__((common)) = NULL

Referenced by rand_start().

void(* sk_pop_free_d)(_STACK *st, void(*func)(void *)) __attribute__((common)) = NULL

Referenced by ssl_stop().

SPF_errcode_t(* SPF_dns_zone_add_str_d)(SPF_dns_server_t *spf_dns_server, const char *domain, ns_type rr_type, SPF_dns_stat_t herrno, const char *data) __attribute__((common)) = NULL

SPF_dns_server_t(* SPF_dns_zone_new_d)(SPF_dns_server_t *layer_below, const char *name, int debug) __attribute__((common)) = NULL

void(* SPF_get_lib_version_d)(int *major, int *minor, int *patch) __attribute__((common)) = NULL

Referenced by lib_load_spf().

void(* SPF_request_free_d)(SPF_request_t *sr) __attribute__((common)) = NULL

Referenced by spf_check().

SPF_request_t(* SPF_request_new_d)(SPF_server_t *spf_server) __attribute__((common)) = NULL

Referenced by spf_check().

SPF_errcode_t(* SPF_request_query_mailfrom_d)(SPF_request_t *spf_request, SPF_response_t **spf_responsep) __attribute__((common)) = NULL

Referenced by spf_check().

int(* SPF_request_set_env_from_d)(SPF_request_t *sr, const char *from) __attribute__((common)) = NULL

Referenced by spf_check().

SPF_errcode_t(* SPF_request_set_helo_dom_d)(SPF_request_t *sr, const char *dom) __attribute__((common)) = NULL

Referenced by spf_check().

SPF_errcode_t(* SPF_request_set_ipv4_d)(SPF_request_t *sr, struct in_addr addr) __attribute__((common)) = NULL

Referenced by spf_check().

SPF_errcode_t(* SPF_request_set_ipv6_d)(SPF_request_t *sr, struct in6_addr addr) __attribute__((common)) = NULL

Referenced by spf_check().

void(* SPF_response_free_d)(SPF_response_t *rp) __attribute__((common)) = NULL

Referenced by spf_check().

SPF_reason_t(* SPF_response_reason_d)(SPF_response_t *rp) __attribute__((common)) = NULL

Referenced by spf_check().

SPF_result_t(* SPF_response_result_d)(SPF_response_t *rp) __attribute__((common)) = NULL

Referenced by spf_check().

void(* SPF_server_free_d)(SPF_server_t *sp) __attribute__((common)) = NULL

SPF.

Referenced by spf_stop().

**SPF_server_t(* SPF_server_new_d)(SPF_server_dnstype_t dnstype, int debug)
__attribute__((common)) = NULL**

Referenced by spf_start().

const char*(* SPF_strerror_d)(SPF_errcode_t spf_err) __attribute__((common)) = NULL

Referenced by spf_check().

const char*(* SPF_strerror_d)(SPF_reason_t reason) __attribute__((common)) = NULL

Referenced by spf_check().

const char*(* SPF_strresult_d)(SPF_result_t result) __attribute__((common)) = NULL

Referenced by spf_check().

int(* SSL_accept_d)(SSL *ssl) __attribute__((common)) = NULL

Referenced by ssl_alloc().

int(* SSL_connect_d)(SSL *ssl) __attribute__((common)) = NULL

Referenced by ssl_client_create().

int(* SSL_CTX_check_private_key_d)(const SSL_CTX *ctx) __attribute__((common)) = NULL

Referenced by `ssl_server_create()`.

`long(* SSL_CTX_ctrl_d)(SSL_CTX *ctx, int cmd, long larg, void *parg) __attribute__((common)) = NULL`

Referenced by `ssl_client_create()`, and `ssl_server_create()`.

`void(* SSL_CTX_free_d)(SSL_CTX *ctx) __attribute__((common)) = NULL`

Referenced by `ssl_client_create()`, `ssl_server_destroy()`, and `ssl_verify_privkey()`.

`int(* SSL_CTX_load_verify_locations_d)(SSL_CTX *ctx, const char *CAfile, const char *CApath) __attribute__((common)) = NULL`

`SSL_CTX*(* SSL_CTX_new_d)(const SSL_METHOD *method) __attribute__((common)) = NULL`

Referenced by `ssl_client_create()`, `ssl_server_create()`, and `ssl_verify_privkey()`.

`int(* SSL_CTX_set_cipher_list_d)(SSL_CTX *, const char *str) __attribute__((common)) = NULL`

Referenced by `ssl_server_create()`.

`void(* SSL_CTX_set_tmp_dh_callback_d)(SSL_CTX *ctx, DH *(*dh)(SSL *ssl, int is_export, int keylength)) __attribute__((common)) = NULL`

Referenced by `ssl_server_create()`.

`void(* SSL_CTX_set_tmp_ecdh_callback_d)(SSL_CTX *ctx, EC_KEY *(*ecdh)(SSL *ssl, int is_export, int keylength)) __attribute__((common)) = NULL`

Referenced by `ssl_server_create()`.

`int(* SSL_CTX_use_certificate_chain_file_d)(SSL_CTX *ctx, const char *file) __attribute__((common)) = NULL`

Referenced by `ssl_server_create()`.

`int(* SSL_CTX_use_PrivateKey_file_d)(SSL_CTX *ctx, const char *file, int type) __attribute__((common)) = NULL`

Referenced by `ssl_server_create()`, and `ssl_verify_privkey()`.

`void(* SSL_free_d)(SSL *ssl) __attribute__((common)) = NULL`

Referenced by `ssl_alloc()`, `ssl_client_create()`, and `ssl_free()`.

int(* SSL_get_error_d)(const SSL *s, int ret_code) __attribute__((common)) = NULL

Referenced by client_read(), client_read_line(), client_write(), con_write_bl(), ssl_read(), and ssl_write().

X509(* SSL_get_peer_certificate_d)(const SSL *s) __attribute__((common)) = NULL

int(* SSL_get_shutdown_d)(const SSL *ssl) __attribute__((common)) = NULL

Referenced by ssl_shutdown_get().

BIO(* SSL_get_wbio_d)(const SSL *ssl) __attribute__((common)) = NULL

int(* SSL_library_init_d)(void) __attribute__((common)) = NULL

Referenced by ssl_start().

void(* SSL_load_error_strings_d)(void) __attribute__((common)) = NULL

Referenced by ssl_start().

SSL(* SSL_new_d)(SSL_CTX *ctx) __attribute__((common)) = NULL

Referenced by ssl_alloc(), and ssl_client_create().

int(* SSL_peek_d)(SSL *ssl, void *buf, int num) __attribute__((common)) = NULL

Referenced by ssl_read().

int(* SSL_read_d)(SSL *ssl, void *buf, int num) __attribute__((common)) = NULL

Referenced by ssl_read().

void(* SSL_set_bio_d)(SSL *ssl, BIO *rbio, BIO *wbio) __attribute__((common)) = NULL

Referenced by ssl_alloc(), and ssl_client_create().

int(* SSL_shutdown_d)(SSL *ssl) __attribute__((common)) = NULL

Referenced by ssl_free().

int(* SSL_write_d)(SSL *ssl, const void *buf, int num) __attribute__((common)) = NULL

Referenced by ssl_write().

const char*(* SSLeay_version_d)(int t) __attribute__((common)) = NULL

const SSL_METHOD*(* SSLv23_client_method_d)(void) __attribute__((common)) = NULL

Referenced by ssl_client_create(), and ssl_verify_privkey().

const SSL_METHOD*(* SSLv23_server_method_d)(void) __attribute__((common)) = NULL

Referenced by ssl_server_create().

void(* tcfree_d)(void *ptr) __attribute__((common)) = NULL

Referenced by tank_load(), tree_cursor_next(), and tree_find().

bool(* tchdbclose_d)(TCHDB *hdb) __attribute__((common)) = NULL

Referenced by tank_close().

bool(* tchdbdefrag_d)(TCHDB *hdb, int64_t step) __attribute__((common)) = NULL

Referenced by tank_close(), and tank_maintain().

void(* tchdbdel_d)(TCHDB *hdb) __attribute__((common)) = NULL

Referenced by tank_close(), and tank_open().

int(* tchdbecode_d)(TCHDB *hdb) __attribute__((common)) = NULL

Referenced by tank_close(), tank_delete(), tank_load(), tank_open(), and tank_store().

const char*(* tchdberrmsg_d)(int ecode) __attribute__((common)) = NULL

Referenced by tank_close(), tank_delete(), tank_load(), tank_open(), and tank_store().

uint64_t(* tchdbfsiz_d)(TCHDB *hdb) __attribute__((common)) = NULL

Referenced by tank_size().

void(* tchdbget_d)(TCHDB *hdb, const void *kbuf, int ksiz, int *sp) __attribute__((common)) = NULL

Referenced by tank_load().

TCHDB>(* tchdbnew_d)(void) __attribute__((common)) = NULL

Referenced by tank_open().

bool(* tchdbopen_d)(TCHDB *hdb, const char *path, int omode) __attribute__((common)) = NULL

Referenced by tank_open().

bool(* tchdboptimize_d)(TCHDB *hdb, int64_t bnum, int8_t apow, int8_t fpow, uint8_t opts) __attribute__((common)) = NULL

Referenced by tank_close().

bool(* tchdbout_d)(TCHDB *hdb, const void *kbuf, int ksiz) __attribute__((common)) = NULL

Referenced by tank_delete(), and tank_store().

const char*(tchdbpath_d)(TCHDB *hdb) __attribute__((common)) = NULL

Referenced by tank_close().

bool(* tchdbputasync_d)(TCHDB *hdb, const void *kbuf, int ksiz, const void *vbuf, int vsiz) __attribute__((common)) = NULL

Referenced by tank_store().

uint64_t(* tchdbbrnum_d)(TCHDB *hdb) __attribute__((common)) = NULL

Referenced by tank_count().

bool(* tchdbsetdfunit_d)(TCHDB *hdb, int32_t dfunit) __attribute__((common)) = NULL

bool(* tchdbsetmutex_d)(TCHDB *hdb) __attribute__((common)) = NULL

Referenced by tank_open().

bool(* tchdbsync_d)(TCHDB *hdb) __attribute__((common)) = NULL

Referenced by tank_close().

bool(* tchdbtune_d)(TCHDB *hdb, int64_t bnum, int8_t apow, int8_t fpow, uint8_t opts) __attribute__((common)) = NULL

Referenced by tank_open().

void(* tclistdel_d)(TCLIST *list) __attribute__((common)) = NULL

Referenced by tree_truncate().

int(* tclistnum_d)(const TCLIST *list) __attribute__((common)) = NULL

Referenced by tree_truncate().

const void(* tclistval_d)(const TCLIST *list, int index, int *sp) __attribute__((common)) = NULL

Referenced by tree_truncate().

void(* tcndbdel_d)(TCNDB *tree) __attribute__((common)) = NULL

Referenced by tree_cursor_free(), tree_cursor_reset(), and tree_free().

TCNDB(* tcndbdup_d)(TCNDB *ndb) __attribute__((common)) = NULL

Referenced by tree_cursor_alloc(), and tree_cursor_reset().

TCLIST(* tcndbfwmkeys_d)(TCNDB *ndb, const void *pbuf, int psiz, int max) __attribute__((common)) = NULL

void(* tcndbget3_d)(TCNDB *ndb, const void *kbuf, int ksiz, int *sp) __attribute__((common)) = NULL

Referenced by tree_cursor_next(), and tree_find().

void(* tcndbget_d)(TCNDB *ndb, const void *kbuf, int ksiz, int *sp) __attribute__((common)) = NULL

bool(* tcndbgetboth_d)(TCNDB *ndb, const void *kbuf, int ksiz, void **rkbuf, int *rksiz, void **rvbuf, int *rvsiz) __attribute__((common)) = NULL

Referenced by tree_delete().

void(* tcndbiterinit_d)(TCNDB *ndb) __attribute__((common)) = NULL

Referenced by tree_cursor_alloc(), and tree_cursor_reset().

char(* tcndbiternext2_d)(TCNDB *ndb) __attribute__((common)) = NULL

Referenced by tree_cursor_next().

TCNDB*(* tcndbnew2_d)(TCCMP cmp, void *cmpop) __attribute__((common)) = NULL

Referenced by tree_alloc().

bool(* tcndbout_d)(TCNDB *ndb, const void *kbuf, int ksiz) __attribute__((common)) = NULL

Referenced by tree_delete().

bool(* tcndbputkeep_d)(TCNDB *ndb, const void *kbuf, int ksiz, const void *vbuf, int vsiz) __attribute__((common)) = NULL

Referenced by tree_insert().

uint64_t(* tcndbrnum_d)(TCNDB *ndb) __attribute__((common)) = NULL

Referenced by tree_count().

TCLIST*(* tctreekeys_d)(const TCTREE *tree) __attribute__((common)) = NULL

Referenced by tree_truncate().

TCLIST*(* tctreevals_d)(const TCTREE *tree) __attribute__((common)) = NULL

Referenced by tree_truncate().

const SSL_METHOD*(* TLSv1_server_method_d)(void) __attribute__((common)) = NULL

int(* uncompress_d)(Bytef *dest, uLongf *destLen, const Bytef *source, uLong sourceLen) __attribute__((common)) = NULL

Referenced by decompress_zlib().

int(* X509_get_ext_count_d)(X509 *x) __attribute__((common)) = NULL

X509_EXTENSION*(* X509_get_ext_d)(X509 *x, int loc) __attribute__((common)) = NULL

X509_NAME*(* X509_get_subject_name_d)(X509 *a) __attribute__((common)) = NULL

int(* X509_NAME_get_text_by_NID_d)(X509_NAME *name, int nid, char *buf, int len) __attribute__((common)) = NULL

xmlNodePtr(* xmlAddSibling_d)(xmlNodePtr cur, xmlNodePtr elem) __attribute__((common)) = NULL

Referenced by xml_node_add_sibling().

const xmlChar*(* xmlBufferContent_d)(const xmlBufferPtr buf) __attribute__((common)) = NULL

Referenced by xml_node_get_content_st().

xmlBufferPtr(* xmlBufferCreate_d)(void) __attribute__((common)) = NULL

Referenced by xml_node_get_content_st().

void(* xmlBufferFree_d)(xmlBufferPtr buf) __attribute__((common)) = NULL

Referenced by xml_node_get_content_st().

int(* xmlBufferLength_d)(const xmlBufferPtr buf) __attribute__((common)) = NULL

Referenced by xml_node_get_content_st().

void(* xmlCleanupGlobals_d)(void) __attribute__((common)) = NULL

Referenced by xml_stop().

void(* xmlCleanupParser_d)(void) __attribute__((common)) = NULL

Referenced by xml_stop().

xmlDocPtr(* xmlCtxtReadMemory_d)(xmlParserCtxtPtr ctxt, const char *buffer, int size, const char *url, const char *encoding, int options) __attribute__((common)) = NULL

Referenced by xml_create_doc().

void(* xmlDocDumpFormatMemory_d)(xmlDocPtr cur, xmlChar **mem, int *size, int format) __attribute__((common)) = NULL

Referenced by xml_dump_doc().

xmlChar*(* xmlEncodeEntitiesReentrant_d)(xmlDocPtr doc, const xmlChar *input) __attribute__((common)) = NULL

Referenced by xml_encode().

void(* xmlFreeDoc_d)(xmlDocPtr doc) __attribute__((common)) = NULL

Referenced by xml_free_doc().

void(* xmlFreeNode_d)(xmlNodePtr cur) __attribute__((common)) = NULL

Referenced by xml_node_free().

void(* xmlFreeParserCtxt_d)(xmlParserCtxtPtr ctx) __attribute__((common)) = NULL

Referenced by xml_free_parser_ctx().

void(* xmlInitParser_d)(void) __attribute__((common)) = NULL

Referenced by xml_start().

void(* xmlMemoryDump_d)(void) __attribute__((common)) = NULL

Referenced by xml_stop().

xmlNodePtr(* xmlNewNode_d)(xmlNsPtr ns, const xmlChar *name) __attribute__((common)) = NULL

Referenced by xml_node_new().

xmlParserCtxtPtr(* xmlNewParserCtxt_d)(void) __attribute__((common)) = NULL

Referenced by xml_create_parser_ctx().

int(* xmlNodeBufGetContent_d)(xmlBufferPtr buffer, xmlNodePtr cur) __attribute__((common)) = NULL

Referenced by xml_node_get_content_st().

void(* xmlNodeSetContent_d)(xmlNodePtr cur, const xmlChar *content) __attribute__((common)) = NULL

Referenced by xml_node_set_content().

xmlAttrPtr(* xmlSetProp_d)(xmlNodePtr node, const xmlChar *name, const xmlChar *value) __attribute__((common)) = NULL

Referenced by xml_node_set_property().

xmlXPathObjectPtr(* xmlXPathEvalExpression_d)(const xmlChar *xpath, xmlXPathContextPtr ctx) __attribute__((common)) = NULL

Referenced by xml_get_xpath_int16(), xml_get_xpath_int32(), xml_get_xpath_int64(), xml_get_xpath_int8(), xml_get_xpath_node_count(), xml_get_xpath_ns(), xml_get_xpath_st(), xml_get_xpath_uint16(), xml_get_xpath_uint32(), xml_get_xpath_uint64(), xml_get_xpath_uint8(), and xml_xpath_eval().

void(* xmlXPathFreeContext_d)(xmlXPathContextPtr ctx) __attribute__((common)) = NULL

Referenced by xml_free_xpath_ctx().

void(* xmlXPathFreeObject_d)(xmlXPathObjectPtr obj) __attribute__((common)) = NULL

Referenced by xml_free_xpath_obj(), xml_get_xpath_int16(), xml_get_xpath_int32(), xml_get_xpath_int64(), xml_get_xpath_int8(), xml_get_xpath_node_count(), xml_get_xpath_ns(), xml_get_xpath_st(), xml_get_xpath_uint16(), xml_get_xpath_uint32(), xml_get_xpath_uint64(), and xml_get_xpath_uint8().

xmlXPathContextPtr(* xmlXPathNewContext_d)(xmlDocPtr doc) __attribute__((common)) = NULL

Referenced by xml_create_xpath_ctx().

int(* xmlXPathRegisterNs_d)(xmlXPathContextPtr ctxt, const xmlChar *prefix, const xmlChar *ns_uri) __attribute__((common)) = NULL

Referenced by xml_xpath_set_namespace().

const char*(* zlibVersion_d)(void) __attribute__((common)) = NULL

ZLIB.

Referenced by lib_version_zlib().

magma/queries.h File Reference

Assorted SQL queries used throughout Magma.

Defines

- #define **SELECT_DOMAINS** "SELECT domain, restricted, mailboxes, wildcard, dkim, spf FROM Domains"
- #define **SELECT_CONFIG** "SELECT name, value FROM Host_Config LEFT JOIN Hosts ON (Host_Config.hostnum = Hosts.hostnum) WHERE application = 'magma' AND (Host_Config.hostnum IS NULL OR Hosts.hostname = ?) ORDER BY Host_Config.hostnum ASC"
- #define **SELECT_HOST_NUMBER** "SELECT hostnum FROM Hosts WHERE hostname = ?"
- #define **DELETE_OBJECT** "DELETE FROM Objects WHERE objectnum = ? AND hostnum = ? AND tank = ? AND usernum = ?"
- #define **INSERT_OBJECT** "INSERT INTO Objects (usernum, hostnum, tank, size, serial, flags, `references`, timestamp) VALUES (?, ?, ?, ?, 0, ?, 0, NOW())"
- #define **SELECT_USER** "SELECT Dispatch.secure, locked, Users.usernum, `ssl`, overquota FROM Users INNER JOIN Dispatch ON Users.usernum = Dispatch.usernum WHERE userid = ? AND password = ? AND email = 1"
- #define **SELECT_USER_RECORD** "SELECT password, Dispatch.secure, locked, `ssl`, overquota FROM Users INNER JOIN Dispatch ON Users.usernum = Dispatch.usernum WHERE Users.usernum = ? AND email = 1"
- #define **SELECT_USER_STORAGE_KEYS** "SELECT storage_pub, storage_priv FROM `Keys` WHERE usernum = ?"
- #define **UPDATE_USER_STORAGE_KEYS** "INSERT INTO `Keys` (usernum, storage_pub, storage_priv) VALUES (?, ?, ?) ON DUPLICATE KEY UPDATE storage_pub = ?, storage_priv = ?"
- #define **UPDATE_USER_LOCK** "UPDATE Users SET locked = ? WHERE usernum = ?"
- #define **UPDATE_USER_QUOTA_ADD** "UPDATE Users SET size = size + ?, overquota = IF(size < quota, 0, 1) WHERE usernum = ?"
- #define **UPDATE_USER_QUOTA_SUBTRACT** "UPDATE Users SET size = size - ?, overquota = IF(size < quota, 0, 1) WHERE usernum = ?"
- #define **SELECT_MAILBOX_ALIASES**
- #define **SELECT_ALERTS** "SELECT alertnum, type, message, UNIX_TIMESTAMP(created) FROM Alerts WHERE usernum = ? AND acknowledged IS NULL"
- #define **UPDATE_ALERTS_ACKNOWLEDGE** "UPDATE Alerts SET acknowledged = NOW() WHERE alertnum = ? AND usernum = ? AND acknowledged IS NULL"
- #define **UPDATE_LOG_POP** "UPDATE Log SET lastpop = NOW(), popsessions = popsessions + 1 WHERE usernum = ?"
- #define **UPDATE_LOG_IMAP** "UPDATE Log SET lastmap = NOW(), mapsessions = mapsessions + 1 WHERE usernum = ?"
- #define **UPDATE_LOG_WEB** "UPDATE Log SET lastweb = NOW(), websessions = websessions + 1 WHERE usernum = ?"
- #define **SELECT_FOLDERS** "SELECT foldernum, parent, `order`, foldername FROM Folders WHERE usernum = ? AND type = ?"
- #define **INSERT_FOLDER** "INSERT INTO Folders (usernum, foldername, `order`, parent, type) VALUES (?, ?, ?, ?, ?)"
- #define **DELETE_FOLDER** "DELETE FROM Folders WHERE foldernum = ? AND usernum = ? AND type = ?"
- #define **UPDATE_FOLDER** "UPDATE Folders SET foldername = ?, parent = ?, `order` = ? WHERE foldernum = ? AND usernum = ? AND type = ?"
- #define **RENAME_FOLDER** "UPDATE Folders SET foldername = ? WHERE foldernum = ? AND usernum = ? AND type = ?"

- #define **SELECT_MESSAGES** "SELECT messagenum, foldernum, server, status, size, signum, sigkey, UNIX_TIMESTAMP(created) FROM Messages WHERE usernum = ? AND visible = 1 ORDER BY messagenum ASC"
- #define **UPDATE_MESSAGE_VISIBILITY** "UPDATE Messages SET visible = 0 WHERE messagenum = ?"
- #define **UPDATE_MESSAGE_FLAGS_ADD** "UPDATE Messages SET status = (status | ?) WHERE usernum = ? AND foldernum = ? AND messagenum = ?"
- #define **UPDATE_MESSAGE_FLAGS_REMOVE** "UPDATE Messages SET status = ((status | ?) ^ ?) WHERE usernum = ? AND foldernum = ? AND messagenum = ?"
- #define **UPDATE_MESSAGE_FLAGS_REPLACE** "UPDATE Messages SET status = (((status | ?) ^ ?) | ?) WHERE usernum = ? AND foldernum = ? AND messagenum = ?"
- #define **UPDATE_MESSAGE_FOLDER** "UPDATE Messages SET foldernum = ? WHERE messagenum = ? AND usernum = ? AND foldernum = ?"
- #define **INSERT_MESSAGE** "INSERT INTO Messages (usernum, foldernum, server, status, size, signum, sigkey, created) VALUES (?, ?, ?, ?, ?, ?, ?, NOW())"
- #define **INSERT_MESSAGE_DUPLICATE** "INSERT INTO Messages (usernum, foldernum, server, status, size, signum, sigkey, created) VALUES (?, ?, ?, ?, ?, ?, ?, FROM_UNIXTIME(?))"
- #define **DELETE_MESSAGE** "DELETE FROM Messages WHERE messagenum = ? AND usernum = ?"
- #define **SELECT_ALL_MESSAGE_TAGS** "SELECT DISTINCT tag from Message_Tags LEFT JOIN Messages ON Message_Tags.messagenum = Messages.messagenum"
- #define **DELETE_MESSAGE_TAGS** "DELETE FROM Message_Tags WHERE messagenum = ?"
- #define **SELECT_MESSAGE_TAGS** "SELECT tag FROM Message_Tags WHERE messagenum = ?"
- #define **INSERT_MESSAGE_TAG** "INSERT INTO Message_Tags (messagenum, tag) VALUES (?, ?)"
- #define **DELETE_MESSAGE_TAG** "DELETE FROM Message_Tags WHERE messagenum = ? AND tag = ?"
- #define **SELECT_AGENTS** "SELECT agentnum, agent, popularity FROM Agents"
- #define **SELECT_MAILBOX_ADDRESS** "SELECT usernum FROM Mailboxes WHERE address = ? AND usernum = ?"
- #define **SELECT_MAILBOX_ADDRESS_ANY** "SELECT * FROM Mailboxes WHERE address = ?"
- #define **SELECT_AUTOREPLY** "SELECT message FROM Autoreplies WHERE replynum = ? AND usernum = ?"
- #define **SELECT_PATTERNS** "SELECT pattern FROM Patterns"
- #define **SELECT_FILTERS** "SELECT rulenum, location, type, action, foldernum, field, label, expression FROM Filters WHERE usernum = ? ORDER BY rulenum ASC"
- #define **SELECT_MESSAGES_ROLLOUT** "SELECT messagenum, size, server FROM Messages WHERE usernum = ? ORDER BY created ASC LIMIT 20"
- #define **SELECT_TRANSMITTING** "SELECT COUNT(*) FROM Transmitting WHERE usernum = ? AND timestamp >= DATE_SUB(NOW(), INTERVAL 1 DAY)"
- #define **SELECT_RECEIVING** "SELECT COUNT(*), SUM(subnet = ?) FROM Receiving WHERE usernum = ? AND timestamp >= DATE_SUB(NOW(), INTERVAL 1 DAY)"
- #define **SELECT_USERS_AUTH** "SELECT Users.usernum, Users.locked, Users.`ssl`, Users.domain, Dispatch.send_size_limit, Dispatch.daily_send_limit, Dispatch.class FROM Users LEFT JOIN Dispatch ON Users.usernum = Dispatch.usernum WHERE userid = ? AND password = ? AND email = 1"
- #define **SELECT_PREFS_INBOUND**
- #define **INSERT_TRANSMITTING** "INSERT INTO Transmitting (usernum, timestamp) VALUES (?, NOW())"
- #define **INSERT_SIGNATURE** "INSERT INTO Signatures (usernum, cryptkey, junk, signature, created) VALUES (?, ?, ?, ?, NOW())"
- #define **INSERT_RECEIVING** "REPLACE INTO Receiving (usernum, subnet, timestamp) VALUES (?, ?, NOW())"
- #define **UPDATE_LOG_SENT** "UPDATE Log SET lastsent = NOW(), totalsent = totalsent + ? WHERE usernum = ?"
- #define **UPDATE_LOG_RECEIVED** "UPDATE Log SET lastreceived = NOW(), totalreceived = totalreceived + ?, totalbounces = totalbounces + ? WHERE usernum = ?"

- #define **SELECT_CONTACTS** "SELECT `contactnum`, `name` FROM `Contacts` WHERE `usernum` = ? AND `foldernum` = ?"
- #define **INSERT_CONTACT** "INSERT INTO `Contacts` (`contactnum`, `usernum`, `foldernum`, `name`, `updated`, `created`) VALUES (NULL, ?, ?, ?, NOW(), NOW())"
- #define **UPDATE_CONTACT** "UPDATE `Contacts` SET `foldernum` = IFNULL(?, `foldernum`), `name` = IFNULL(?, `name`), `updated` = NOW() WHERE `contactnum` = ? AND `usernum` = ? AND `foldernum` = ?"
- #define **UPDATE_CONTACT_STAMP** "UPDATE `Contacts` SET `updated` = NOW() WHERE `contactnum` = ? AND `usernum` = ? AND `foldernum` = ?"
- #define **DELETE_CONTACT** "DELETE `Contacts`, `Contact_Details` FROM `Contacts` LEFT JOIN `Contact_Details` ON `Contacts`.`contactnum` = `Contact_Details`.`contactnum` WHERE `Contacts`.`contactnum` = ? AND `Contacts`.`usernum` = ? AND `Contacts`.`foldernum` = ?"
- #define **UPSERT_CONTACT_DETAIL** "INSERT INTO `Contact_Details` (`contactnum`, `key`, `value`, `flags`) VALUES (?, ?, ?, 0) ON DUPLICATE KEY UPDATE `value` = VALUES(`value`), `flags` = VALUES(`flags`)"
- #define **SELECT_CONTACT_DETAILS** "SELECT `key`, `value`, `flags` FROM `Contact_Details` WHERE `contactnum` = ?"
- #define **DELETE_CONTACT_DETAILS** "DELETE FROM `Contact_Details` WHERE `contactnum` = ? AND `key` = ?"
- #define **SELECT_MESSAGE_FOLDER** "SELECT messagenum, UNIX_TIMESTAMP(created), signum, sigkey, status, server, size FROM Messages WHERE usernum = ? AND foldernum = ? AND visible = 1 ORDER BY messagenum ASC"
- #define **UPSERT_USER_CONFIG** "INSERT INTO `User_Config` (`usernum`, `key`, `value`, `flags`, `timestamp`) VALUES (?, ?, ?, ?, NOW()) ON DUPLICATE KEY UPDATE `value` = VALUES(`value`), `flags` = VALUES(`flags`)"
- #define **SELECT_USER_CONFIG** "SELECT `key`, `value`, `flags` FROM `User_Config` WHERE `usernum` = ?"
- #define **DELETE_USER_CONFIG** "DELETE FROM `User_Config` WHERE `usernum` = ? AND `key` = ?"
- #define **FETCH_SIGNATURE** "SELECT Users.userid, Users.password, Users.usernum, Signatures.junk, Signatures.cryptkey, Signatures.signature FROM Signatures LEFT JOIN Users ON (Signatures.usernum = Users.usernum) WHERE Signatures.signum = ?"
- #define **UPDATE_SIGNATURE_FLAGS_ADD** "UPDATE Messages SET status = (status | ?) WHERE usernum = ? AND signum = ?"
- #define **UPDATE_SIGNATURE_FLAGS_REMOVE** "UPDATE Messages SET status = ((status | ?) ^ ?) WHERE usernum = ? AND signum = ?"
- #define **DELETE_SIGNATURE** "DELETE FROM Signatures WHERE signum = ?"
- #define **REGISTER_CHECK_USERNAME** "SELECT usernum FROM Users WHERE userid = ?"
- #define **REGISTER_INSERT_USER** "INSERT INTO Users (`userid`, `password`, `plan`, `quota`, `plan_expiration`) VALUES (?, ?, ?, ?, ?)"
- #define **REGISTER_INSERT_PROFILE** "INSERT INTO Profile (`usernum`) VALUES (?)"
- #define **REGISTER_INSERT_FOLDERS** "INSERT INTO Folders (`usernum`) VALUES (?)"
- #define **REGISTER_INSERT_FOLDER_NAME** "INSERT INTO Folders (`usernum`, `foldername`) VALUES (?, ?)"
- #define **REGISTER_INSERT_LOG** "INSERT INTO Log (`usernum`, `created_ip`) VALUES (?, ?)"
- #define **REGISTER_INSERT_DISPATCH** "INSERT INTO Dispatch (`usernum`, `spamfolder`, `inbox`, `send_size_limit`, `recv_size_limit`, `daily_send_limit`, `daily_recv_limit`, `daily_recv_limit_ip`) VALUES (?, ?, ?, ?, ?, ?, ?, ?)"
- #define **REGISTER_INSERT_MAILBOXES** "INSERT INTO Mailboxes (`address`, `usernum`) VALUES (?, ?)"
- #define **REGISTER_FETCH_BLOCKLIST** "SELECT sequence FROM Banned"
- #define **STATISTICS_GET_TOTAL_USERS** "SELECT COUNT(*) FROM Users"

- `#define STATISTICS_GET_USERS_CHECKED_EMAIL_TODAY "SELECT COUNT(*) FROM Log WHERE lastpop >= DATE_SUB(NOW(), INTERVAL 1 DAY) OR lastweb >= DATE_SUB(NOW(), INTERVAL 1 DAY)"`
- `#define STATISTICS_GET_USERS_CHECKED_EMAIL_WEEK "SELECT COUNT(*) FROM Log WHERE lastpop >= DATE_SUB(NOW(), INTERVAL 7 DAY) OR lastweb >= DATE_SUB(NOW(), INTERVAL 7 DAY)"`
- `#define STATISTICS_GET_USERS_SENT_EMAIL_TODAY "SELECT COUNT(*) FROM Log WHERE lastsent >= DATE_SUB(NOW(), INTERVAL 1 DAY)"`
- `#define STATISTICS_GET_USERS_SENT_EMAIL_WEEK "SELECT COUNT(*) FROM Log WHERE lastsent >= DATE_SUB(NOW(), INTERVAL 7 DAY)"`
- `#define STATISTICS_GET_EMAILS_RECEIVED_TODAY "SELECT COUNT(*) FROM Receiving WHERE timestamp >= DATE_SUB(NOW(), INTERVAL 1 DAY)"`
- `#define STATISTICS_GET_EMAILS_RECEIVED_WEEK "SELECT COUNT(*) FROM Receiving WHERE timestamp >= DATE_SUB(NOW(), INTERVAL 7 DAY)"`
- `#define STATISTICS_GET_EMAILS_SENT_TODAY "SELECT COUNT(*) FROM Transmitting WHERE timestamp >= DATE_SUB(NOW(), INTERVAL 1 DAY)"`
- `#define STATISTICS_GET_EMAILS_SENT_WEEK "SELECT COUNT(*) FROM Transmitting WHERE timestamp >= DATE_SUB(NOW(), INTERVAL 7 DAY)"`
- `#define STATISTICS_GET_USERS_REGISTERED_TODAY "SELECT COUNT(*) FROM Creation WHERE timestamp >= DATE_SUB(NOW(), INTERVAL 1 DAY)"`
- `#define STATISTICS_GET_USERS_REGISTERED_WEEK "SELECT COUNT(*) FROM Creation WHERE timestamp >= DATE_SUB(NOW(), INTERVAL 7 DAY)"`
- `#define QUERIES_INIT`
- `#define STMTS_INIT`

Variables

- `chr_t * queries []`
- `struct {`
- `MYSQL_STMT STMTS_INIT`
- `} common`

Detailed Description

Assorted SQL queries used throughout Magma.

Definition in file `queries.h`.

Define Documentation

```
#define DELETE_CONTACT "DELETE `Contacts`, `Contact_Details` FROM `Contacts` LEFT JOIN
`Contact_Details` ON `Contacts`.`contactnum` = `Contact_Details`.`contactnum` WHERE
`Contacts`.`contactnum` = ? AND `Contacts`.`usernum` = ? AND `Contacts`.`foldernum` = ?"
```

Definition at line 107 of file `queries.h`.

```
#define DELETE_CONTACT_DETAILS "DELETE FROM `Contact_Details` WHERE `contactnum` =
? AND `key` = ?"
```

Definition at line 112 of file queries.h.

```
#define DELETE_FOLDER "DELETE FROM Folders WHERE foldernum = ? AND usernum = ? AND  
type = ?"
```

Definition at line 54 of file queries.h.

```
#define DELETE_MESSAGE "DELETE FROM Messages WHERE messagenum = ? AND usernum  
= ?"
```

Definition at line 67 of file queries.h.

```
#define DELETE_MESSAGE_TAG "DELETE FROM Message_Tags WHERE messagenum = ? AND  
tag = ?"
```

Definition at line 74 of file queries.h.

```
#define DELETE_MESSAGE_TAGS "DELETE FROM Message_Tags WHERE messagenum = ?"
```

Definition at line 71 of file queries.h.

```
#define DELETE_OBJECT "DELETE FROM Objects WHERE objectnum = ? AND hostnum = ?  
AND tank = ? AND usernum = ?"
```

Definition at line 25 of file queries.h.

```
#define DELETE_SIGNATURE "DELETE FROM Signatures WHERE signum = ?"
```

Definition at line 126 of file queries.h.

```
#define DELETE_USER_CONFIG "DELETE FROM `User_Config` WHERE `usernum` = ? AND `key`  
= ?"
```

Definition at line 120 of file queries.h.

```
#define FETCH_SIGNATURE "SELECT Users.userid, Users.password, Users.usernum,  
Signatures.junk, Signatures.cryptkey, Signatures.signature FROM Signatures LEFT JOIN Users  
ON (Signatures.usernum = Users.usernum) WHERE Signatures.signum = ?"
```

Definition at line 123 of file queries.h.

```
#define INSERT_CONTACT "INSERT INTO `Contacts` ( `contactnum`, `usernum`, `foldernum`,  
`name`, `updated`, `created`) VALUES (NULL, ?, ?, ?, NOW(), NOW())"
```

Definition at line 104 of file queries.h.

```
#define INSERT_FOLDER "INSERT INTO Folders (usernum, foldername, `order`, parent, type)  
VALUES (?, ?, ?, ?, ?)"
```

Definition at line 53 of file queries.h.

```
#define INSERT_MESSAGE "INSERT INTO Messages (usernum, foldernum, server, status, size,  
signum, sigkey, created) VALUES (?, ?, ?, ?, ?, ?, ?, NOW())"
```

Definition at line 65 of file queries.h.

```
#define INSERT_MESSAGE_DUPLICATE "INSERT INTO Messages (usernum, foldernum, server,  
status, size, signum, sigkey, created) VALUES (?, ?, ?, ?, ?, ?, ?, FROM_UNIXTIME(?))"
```

Definition at line 66 of file queries.h.

```
#define INSERT_MESSAGE_TAG "INSERT INTO Message_Tags (messagenum, tag) VALUES (?,  
?)"
```

Definition at line 73 of file queries.h.

```
#define INSERT_OBJECT "INSERT INTO Objects (usernum, hostnum, tank, size, serial, flags,  
`references`, timestamp) VALUES (?, ?, ?, ?, ?, 0, 0, NOW())"
```

Definition at line 26 of file queries.h.

```
#define INSERT_RECEIVING "REPLACE INTO Receiving (usernum, subnet, timestamp) VALUES  
(?, ?, NOW())"
```

Definition at line 98 of file queries.h.

```
#define INSERT_SIGNATURE "INSERT INTO Signatures (usernum, cryptkey, junk, signature,  
created) VALUES (?, ?, ?, ?, NOW())"
```

Definition at line 97 of file queries.h.

```
#define INSERT_TRANSMITTING "INSERT INTO Transmitting (usernum, timestamp) VALUES (?,  
NOW())"
```

Definition at line 96 of file queries.h.

```
#define QUERIES_INIT
```

Note:

Be sure to add any new queries to this list.

Definition at line 163 of file queries.h.


```
#define REGISTER_CHECK_USERNAME "SELECT usernum FROM Users WHERE userid = ?"
```

Definition at line 129 of file queries.h.

```
#define REGISTER_FETCH_BLOCKLIST "SELECT sequence FROM Banned"
```

Definition at line 137 of file queries.h.

```
#define REGISTER_INSERT_DISPATCH "INSERT INTO Dispatch (`usernum`, `spamfolder`,  
`inbox`, `send_size_limit`, `recv_size_limit`, `daily_send_limit`, `daily_recv_limit`,  
`daily_recv_limit_ip`) VALUES (?, ?, ?, ?, ?, ?, ?, ?)"
```

Definition at line 135 of file queries.h.

```
#define REGISTER_INSERT_FOLDER_NAME "INSERT INTO Folders (`usernum`, `foldername`)  
VALUES (?, ?)"
```

Definition at line 133 of file queries.h.

```
#define REGISTER_INSERT_FOLDERS "INSERT INTO Folders (`usernum`) VALUES (?)"
```

Definition at line 132 of file queries.h.

```
#define REGISTER_INSERT_LOG "INSERT INTO Log (`usernum`, `created_ip`) VALUES (?, ?)"
```

Definition at line 134 of file queries.h.

```
#define REGISTER_INSERT_MAILBOXES "INSERT INTO Mailboxes (`address`, `usernum`)  
VALUES (?, ?)"
```

Definition at line 136 of file queries.h.

```
#define REGISTER_INSERT_PROFILE "INSERT INTO Profile (`usernum`) VALUES (?)"
```

Definition at line 131 of file queries.h.

```
#define REGISTER_INSERT_USER "INSERT INTO Users (`userid`, `password`, `plan`, `quota`,  
`plan_expiration`) VALUES (?, ?, ?, ?, ?)"
```

Definition at line 130 of file queries.h.

```
#define RENAME_FOLDER "UPDATE Folders SET foldername = ? WHERE foldernum = ? AND  
usernum = ? AND type = ?"
```

Definition at line 56 of file queries.h.

#define SELECT_AGENTS "SELECT agentnum, agent, popularity FROM Agents"

Definition at line 77 of file queries.h.

#define SELECT_ALERTS "SELECT alertnum, type, message, UNIX_TIMESTAMP(created) FROM Alerts WHERE usernum = ? AND acknowledged IS NULL"

Definition at line 43 of file queries.h.

#define SELECT_ALL_MESSAGE_TAGS "SELECT DISTINCT tag from Message_Tags LEFT JOIN Messages ON Message_Tags.messageenum = Messages.messageenum"

Definition at line 70 of file queries.h.

#define SELECT_AUTOREPLY "SELECT message FROM Autoreplies WHERE replynum = ? AND usernum = ?"

Definition at line 82 of file queries.h.

#define SELECT_CONFIG "SELECT name, value FROM Host_Config LEFT JOIN Hosts ON (Host_Config.hostnum = Hosts.hostnum) WHERE application = 'magma' AND (Host_Config.hostnum IS NULL OR Hosts.hostname = ?) ORDER BY Host_Config.hostnum ASC"

Definition at line 21 of file queries.h.

#define SELECT_CONTACT_DETAILS "SELECT `key`, `value`, `flags` FROM `Contact_Details` WHERE `contactnum` = ?"

Definition at line 111 of file queries.h.

#define SELECT_CONTACTS "SELECT `contactnum`, `name` FROM `Contacts` WHERE `usernum` = ? AND `foldernum` = ?"

Definition at line 103 of file queries.h.

#define SELECT_DOMAINS "SELECT domain, restricted, mailboxes, wildcard, dkim, spf FROM Domains"

Definition at line 17 of file queries.h.

Referenced by sanity_check().

#define SELECT_FILTERS "SELECT rulenum, location, type, action, foldernum, field, label, expression FROM Filters WHERE usernum = ? ORDER BY rulenum ASC"

Definition at line 84 of file queries.h.

```
#define SELECT_FOLDERS "SELECT foldernum, parent, `order`, foldername FROM Folders  
WHERE usernum = ? AND type = ?"
```

Definition at line 52 of file queries.h.

```
#define SELECT_HOST_NUMBER "SELECT hostnum FROM Hosts WHERE hostname = ?"
```

Definition at line 22 of file queries.h.

```
#define SELECT_MAILBOX_ADDRESS "SELECT usernum FROM Mailboxes WHERE address = ?  
AND usernum = ?"
```

Definition at line 80 of file queries.h.

```
#define SELECT_MAILBOX_ADDRESS_ANY "SELECT * FROM Mailboxes WHERE address = ?"
```

Definition at line 81 of file queries.h.

```
#define SELECT_MAILBOX_ALIASES
```

```
Value: "SELECT Aliases.aliasnum, Mailboxes.address, Aliases.display, Aliases.selected,  
UNIX_TIMESTAMP(Aliases.created) from Mailboxes " \  
    "LEFT JOIN Aliases ON Mailboxes.address = Aliases.address AND Mailboxes.usernum =  
    Aliases.usernum " \  
    "WHERE Mailboxes.usernum = ? ORDER BY selected DESC, LOWER(Aliases.display) ASC,  
    LOWER(Mailboxes.address) ASC"
```

Definition at line 38 of file queries.h.

```
#define SELECT_MESSAGE_FOLDER "SELECT messagenum, UNIX_TIMESTAMP(created),  
signum, sigkey, status, server, size FROM Messages WHERE usernum = ? AND foldernum = ?  
AND visible = 1 ORDER BY messagenum ASC"
```

Definition at line 115 of file queries.h.

```
#define SELECT_MESSAGE_TAGS "SELECT tag FROM Message_Tags WHERE messagenum =  
?"
```

Definition at line 72 of file queries.h.

```
#define SELECT_MESSAGES "SELECT messagenum, foldernum, server, status, size, signum,  
sigkey, UNIX_TIMESTAMP(created) FROM Messages WHERE usernum = ? AND visible = 1 ORDER  
BY messagenum ASC"
```

Definition at line 59 of file queries.h.

```
#define SELECT_MESSAGES_ROLLOUT "SELECT messagenum, size, server FROM Messages  
WHERE usernum = ? ORDER BY created ASC LIMIT 20"
```

Definition at line 85 of file queries.h.

#define SELECT_PATTERNS "SELECT pattern FROM Patterns"

Definition at line 83 of file queries.h.

#define SELECT_PREFS_INBOUND

```
Value: "SELECT Mailboxes.username, Users.locked, Users.size, Users.quota, Users.overquota, " \
      "Users.domain, Dispatch.secure, Dispatch.bounces, Dispatch.forwarded,
Dispatch.rollout, " \
      "Dispatch.spam, Dispatch.spamaction, Dispatch.virus, Dispatch.virusaction,
Dispatch.phish, Dispatch.phishaction, " \
      "Dispatch.autoreply, Dispatch.inbox, Dispatch.recv_size_limit,
Dispatch.daily_recv_limit, Dispatch.daily_recv_limit_ip, " \
      "Dispatch.greylst, Dispatch.greytime, Dispatch.spf, Dispatch.spfaction,
Dispatch.dkim, Dispatch.dkimaction, Dispatch.rbl, " \
      "Dispatch.rblaction, Dispatch.filters FROM Mailboxes LEFT JOIN Users ON
Mailboxes.username = Users.username LEFT JOIN Dispatch ON " \
      "Mailboxes.username = Dispatch.username WHERE Mailboxes.address = ?"
```

Definition at line 89 of file queries.h.

#define SELECT_RECEIVING "SELECT COUNT(*), SUM(subnet = ?) FROM Receiving WHERE username = ? AND timestamp >= DATE_SUB(NOW(), INTERVAL 1 DAY)"

Definition at line 87 of file queries.h.

#define SELECT_TRANSMITTING "SELECT COUNT(*) FROM Transmitting WHERE username = ? AND timestamp >= DATE_SUB(NOW(), INTERVAL 1 DAY)"

Definition at line 86 of file queries.h.

#define SELECT_USER "SELECT Dispatch.secure, locked, Users.username, `ssl`, overquota FROM Users INNER JOIN Dispatch ON Users.username = Dispatch.username WHERE userid = ? AND password = ? AND email = 1"

Definition at line 29 of file queries.h.

#define SELECT_USER_CONFIG "SELECT `key`, `value`, `flags` FROM `User_Config` WHERE `username` = ?"

Definition at line 119 of file queries.h.

#define SELECT_USER_RECORD "SELECT password, Dispatch.secure, locked, `ssl`, overquota FROM Users INNER JOIN Dispatch ON Users.username = Dispatch.username WHERE Users.username = ? AND email = 1"

Definition at line 30 of file queries.h.

#define SELECT_USER_STORAGE_KEYS "SELECT storage_pub, storage_priv FROM `Keys` WHERE username = ?"

Definition at line 31 of file queries.h.

```
#define SELECT_USERS_AUTH "SELECT Users.usernum, Users.locked, Users.`ssl`,  
Users.domain, Dispatch.send_size_limit, Dispatch.daily_send_limit, Dispatch.class FROM Users  
LEFT JOIN Dispatch ON Users.usernum = Dispatch.usernum WHERE userid = ? AND password =  
? AND email = 1"
```

Definition at line 88 of file queries.h.

```
#define STATISTICS_GET_EMAILS_RECEIVED_TODAY "SELECT COUNT(*) FROM Receiving  
WHERE timestamp >= DATE_SUB(NOW(), INTERVAL 1 DAY)"
```

Definition at line 145 of file queries.h.

```
#define STATISTICS_GET_EMAILS_RECEIVED_WEEK "SELECT COUNT(*) FROM Receiving  
WHERE timestamp >= DATE_SUB(NOW(), INTERVAL 7 DAY)"
```

Definition at line 146 of file queries.h.

```
#define STATISTICS_GET_EMAILS_SENT_TODAY "SELECT COUNT(*) FROM Transmitting  
WHERE timestamp >= DATE_SUB(NOW(), INTERVAL 1 DAY)"
```

Definition at line 147 of file queries.h.

```
#define STATISTICS_GET_EMAILS_SENT_WEEK "SELECT COUNT(*) FROM Transmitting  
WHERE timestamp >= DATE_SUB(NOW(), INTERVAL 7 DAY)"
```

Definition at line 148 of file queries.h.

```
#define STATISTICS_GET_TOTAL_USERS "SELECT COUNT(*) FROM Users"
```

Definition at line 140 of file queries.h.

```
#define STATISTICS_GET_USERS_CHECKED_EMAIL_TODAY "SELECT COUNT(*) FROM Log  
WHERE lastpop >= DATE_SUB(NOW(), INTERVAL 1 DAY) OR lastweb >= DATE_SUB(NOW(),  
INTERVAL 1 DAY)"
```

Definition at line 141 of file queries.h.

```
#define STATISTICS_GET_USERS_CHECKED_EMAIL_WEEK "SELECT COUNT(*) FROM Log  
WHERE lastpop >= DATE_SUB(NOW(), INTERVAL 7 DAY) OR lastweb >= DATE_SUB(NOW(),  
INTERVAL 7 DAY)"
```

Definition at line 142 of file queries.h.

```
#define STATISTICS_GET_USERS_REGISTERED_TODAY "SELECT COUNT(*) FROM Creation  
WHERE timestamp >= DATE_SUB(NOW(), INTERVAL 1 DAY)"
```

Definition at line 149 of file queries.h.

```
#define STATISTICS_GET_USERS_REGISTERED_WEEK "SELECT COUNT(*) FROM Creation  
WHERE timestamp >= DATE_SUB(NOW(), INTERVAL 7 DAY)"
```

Definition at line 150 of file queries.h.

```
#define STATISTICS_GET_USERS_SENT_EMAIL_TODAY "SELECT COUNT(*) FROM Log WHERE  
lastsent >= DATE_SUB(NOW(), INTERVAL 1 DAY)"
```

Definition at line 143 of file queries.h.

```
#define STATISTICS_GET_USERS_SENT_EMAIL_WEEK "SELECT COUNT(*) FROM Log WHERE  
lastsent >= DATE_SUB(NOW(), INTERVAL 7 DAY)"
```

Definition at line 144 of file queries.h.

```
#define STMTS_INIT
```

Definition at line 255 of file queries.h.

```
#define UPDATE_ALERTS_ACKNOWLEDGE "UPDATE Alerts SET acknowledged = NOW()  
WHERE alertnum = ? AND usernum = ? AND acknowledged IS NULL"
```

Definition at line 44 of file queries.h.

```
#define UPDATE_CONTACT "UPDATE `Contacts` SET `foldernum` = IFNULL(?, `foldernum`),  
`name` = IFNULL(?, `name`), `updated` = NOW() WHERE `contactnum` = ? AND `usernum` = ? AND  
`foldernum` = ?"
```

Definition at line 105 of file queries.h.

```
#define UPDATE_CONTACT_STAMP "UPDATE `Contacts` SET `updated` = NOW() WHERE  
`contactnum` = ? AND `usernum` = ? AND `foldernum` = ?"
```

Definition at line 106 of file queries.h.

```
#define UPDATE_FOLDER "UPDATE Folders SET foldername = ?, parent = ?, `order` = ? WHERE  
foldernum = ? AND usernum = ? AND type = ?"
```

Definition at line 55 of file queries.h.

```
#define UPDATE_LOG_IMAP "UPDATE Log SET lastmap = NOW(), mapsessions = mapsessions  
+ 1 WHERE usernum = ?"
```

Definition at line 48 of file queries.h.

```
#define UPDATE_LOG_POP "UPDATE Log SET lastpop = NOW(), popsessions = popsessions + 1  
WHERE usernum = ?"
```

Definition at line 47 of file queries.h.

```
#define UPDATE_LOG_RECEIVED "UPDATE Log SET lastreceived = NOW(), totalreceived =  
totalreceived + ?, totalbounces = totalbounces + ? WHERE usernum = ?"
```

Definition at line 100 of file queries.h.

```
#define UPDATE_LOG_SENT "UPDATE Log SET lastsent = NOW(), totalsent = totalsent + ?  
WHERE usernum = ?"
```

Definition at line 99 of file queries.h.

```
#define UPDATE_LOG_WEB "UPDATE Log SET lastweb = NOW(), websessions = websessions +  
1 WHERE usernum = ?"
```

Definition at line 49 of file queries.h.

```
#define UPDATE_MESSAGE_FLAGS_ADD "UPDATE Messages SET status = (status | ?) WHERE  
usernnum = ? AND foldernum = ? AND messagenum = ?"
```

Definition at line 61 of file queries.h.

```
#define UPDATE_MESSAGE_FLAGS_REMOVE "UPDATE Messages SET status = (((status | ?) ^ ?)  
WHERE usernum = ? AND foldernum = ? AND messagenum = ?"
```

Definition at line 62 of file queries.h.

```
#define UPDATE_MESSAGE_FLAGS_REPLACE "UPDATE Messages SET status = (((status | ?) ^  
?) | ?) WHERE usernum = ? AND foldernum = ? AND messagenum = ?"
```

Definition at line 63 of file queries.h.

```
#define UPDATE_MESSAGE_FOLDER "UPDATE Messages SET foldernum = ? WHERE  
messagenum = ? AND usernum = ? AND foldernum = ?"
```

Definition at line 64 of file queries.h.

```
#define UPDATE_MESSAGE_VISIBILITY "UPDATE Messages SET visible = 0 WHERE  
messagenum = ?"
```

Definition at line 60 of file queries.h.

```
#define UPDATE_SIGNATURE_FLAGS_ADD "UPDATE Messages SET status = (status | ?)  
WHERE usernum = ? AND signum = ?"
```

Definition at line 124 of file queries.h.

```
#define UPDATE_SIGNATURE_FLAGS_REMOVE "UPDATE Messages SET status = ((status | ?) ^  
?) WHERE usernum = ? AND signum = ?"
```

Definition at line 125 of file queries.h.

```
#define UPDATE_USER_LOCK "UPDATE Users SET locked = ? WHERE usernum = ?"
```

Definition at line 33 of file queries.h.

```
#define UPDATE_USER_QUOTA_ADD "UPDATE Users SET size = size + ?, overquota = IF(size <  
quota, 0, 1) WHERE usernum = ?"
```

Definition at line 34 of file queries.h.

```
#define UPDATE_USER_QUOTA_SUBTRACT "UPDATE Users SET size = size - ?, overquota =  
IF(size < quota, 0, 1) WHERE usernum = ?"
```

Definition at line 35 of file queries.h.

```
#define UPDATE_USER_STORAGE_KEYS "INSERT INTO `Keys` (usernum, storage_pub,  
storage_priv) VALUES (?, ?, ?) ON DUPLICATE KEY UPDATE storage_pub = ?, storage_priv = ?"
```

Definition at line 32 of file queries.h.

```
#define UPSERT_CONTACT_DETAIL "INSERT INTO `Contact_Details` (`contactnum`, `key`,  
`value`, `flags`) VALUES (?, ?, ?, 0) ON DUPLICATE KEY UPDATE `value` = VALUES(`value`),  
`flags` = VALUES(`flags`)"
```

Definition at line 110 of file queries.h.

```
#define UPSERT_USER_CONFIG "INSERT INTO `User_Config` (`usernum`, `key`, `value`, `flags`,  
`timestamp`) VALUES (?, ?, ?, ?, NOW()) ON DUPLICATE KEY UPDATE `value` = VALUES(`value`),  
`flags` = VALUES(`flags`)"
```

Definition at line 118 of file queries.h.

Variable Documentation

struct { ... } common

chr_t* queries[]

Definition at line 15 of file stmts.c.

Referenced by sanity_check(), stmt_rebuild(), stmt_start(), and stmt_stop().

MYSQL_STMT_STMTS_INIT

Definition at line 346 of file queries.h.

magma/servers/http/commands.h File Reference

The data structure involved with parsing and routing POP commands.

Detailed Description

The data structure involved with parsing and routing POP commands.

Definition in file **commands.h**.

magma/servers/imap/commands.h File Reference

The data structure involved with parsing and routing POP commands.

Variables

- `command_t imap_commands []`

Detailed Description

The data structure involved with parsing and routing POP commands.

Definition in file `commands.h`.

Variable Documentation

`command_t imap_commands[]`

```
Initial value: {
    { .string = "ID", .length = 2, .function = &imap_id},
    { .string = "COPY", .length = 4, .function = &imap_copy},
    { .string = "IDLE", .length = 4, .function = &imap_idle},
    { .string = "LIST", .length = 4, .function = &imap_list},
    { .string = "LSUB", .length = 4, .function = &imap_lsub},
    { .string = "NOOP", .length = 4, .function = &imap_noop},
    { .string = "CHECK", .length = 5, .function = &imap_check},
    { .string = "CLOSE", .length = 5, .function = &imap_close},
    { .string = "FETCH", .length = 5, .function = &imap_fetch},
    { .string = "LOGIN", .length = 5, .function = &imap_login},
    { .string = "STORE", .length = 5, .function = &imap_store},
    { .string = "APPEND", .length = 6, .function = &imap_append},
    { .string = "CREATE", .length = 6, .function = &imap_create},
    { .string = "DELETE", .length = 6, .function = &imap_delete},
    { .string = "RENAME", .length = 6, .function = &imap_rename},
    { .string = "SEARCH", .length = 6, .function = &imap_search},
    { .string = "SELECT", .length = 6, .function = &imap_select},
    { .string = "LOGOUT", .length = 6, .function = &imap_logout},
    { .string = "STATUS", .length = 6, .function = &imap_status},
    { .string = "EXAMINE", .length = 7, .function = &imap_examine},
    { .string = "EXPUNGE", .length = 7, .function = &imap_expunge},
    { .string = "STARTTLS", .length = 8, .function = &imap_starttls},
    { .string = "SUBSCRIBE", .length = 9, .function = &imap_subscribe},
    { .string = "CAPABILITY", .length = 10, .function = &imap_capability},
    { .string = "UNSUBSCRIBE", .length = 11, .function = &imap_unsubscribe}
}
```

Definition at line 16 of file `commands.h`.

Referenced by `imap_process()`, and `imap_sort()`.

magma/servers/molten/commands.h File Reference

The data structure involved with parsing and routing Molten commands.

Variables

- `command_t molten_commands []`
-

Detailed Description

The data structure involved with parsing and routing Molten commands.

Definition in file **commands.h**.

Variable Documentation

`command_t molten_commands[]`

```
Initial value: {
    {
        .string = "QUIT",
        .length = 4,
        .function = &molten_quit
    }, {
        .string = "STATS",
        .length = 5,
        .function = &molten_stats
    }
}
```

Definition at line 16 of file `commands.h`.

Referenced by `molten_parse()`, and `molten_sort()`.

magma/servers/pop/commands.h File Reference

The data structure involved with parsing and routing POP commands.

Variables

- `command_t pop_commands []`
-

Detailed Description

The data structure involved with parsing and routing POP commands.

Definition in file `commands.h`.

Variable Documentation

`command_t pop_commands[]`

Definition at line 17 of file `commands.h`.

Referenced by `pop_process()`, and `pop_sort()`.

magma/servers/smtp/commands.h File Reference

The data structure for parsing and routing SMTP commands.

Variables

- `command_t smtp_commands []`
-

Detailed Description

The data structure for parsing and routing SMTP commands.

Definition in file `commands.h`.

Variable Documentation

`command_t smtp_commands[]`

Definition at line 16 of file `commands.h`.

Referenced by `smtp_process()`, and `smtp_sort()`.

magma/servers/http/content.c File Reference

Functions for handling the management of web server content.

```
#include "magma.h"
```

Functions

- void **http_free_content** (**http_content_t** *page)
- *LOW: We should use basename() and dirname() to cleanup path strings.* void **http_page_free** (**http_page_t** *page)
- *Free an http page object and all its underlying data.* **http_content_t** * **http_get_static** (**stringer_t** *location)
- *Get a cached copy of a static web page.* **http_content_t** * **http_get_template** (**chr_t** *location)
- *Return a template by location.* **http_page_t** * **http_page_get** (**chr_t** *location)
- *Get a template page and prepare its xml document root for use.* **bool_t** **http_load_file** (**int_t** template, **chr_t** *filename)
- *Load file content into the http server repository.* **bool_t** **http_content_load_directory** (**int_t** template, **chr_t** *directory)
- *Load the content from a directory into the http server repository, recursively.* **bool_t** **http_content_load_fonts** (**void**)
- *Load all fonts into the http server repository.* **bool_t** **http_content_start** (**void**)
- *Prime the the web app templates, static content, and fonts directories for future use.* void **http_content_stop** (**void**)
- *Purge the http contents repository of stored fonts, templates, and static web pages.* **bool_t** **http_content_refresh** (**void**)

Refresh the stored contents of the web app templates, static content, and fonts directories. Variables

- struct {
- **inx_t** * fonts
- **inx_t** * pages
- **inx_t** * templates
- } content

Detailed Description

Functions for handling the management of web server content.

Definition in file **content.c**.

Function Documentation

bool_t **http_content_load_directory** (**int_t** *template*, **chr_t** * *directory*)

Load the content from a directory into the http server repository, recursively.

content.c

See also:

http_load_file()

Parameters:

template a value specifying whether the specified directory contains templates or regular web content.
directory a null-terminated string specifying the pathname of the directory to be scanned recursively.

Returns:

0 on failure or 1 on success.

Definition at line 317 of file content.c.

References `http_content_load_directory()`, `http_load_file()`, `log_info`, `MEMORYBUF`, `NULLER`, `PLACER`, `st_char_get()`, `st_cmp_cs_ends()`, `st_cmp_cs_eq()`, `st_free()`, and `st_merge`.

Referenced by `http_content_load_directory()`, `http_content_refresh()`, and `http_content_start()`.

bool_t http_content_load_fonts (void)

Load all fonts into the http server repository.

Note:

This function will load all fonts of extension ".ttf" from the directory configured in **magma.http.fonts**.

Returns:

0 on failure or 1 on success.

Definition at line 366 of file content.c.

References `magma_t::content`, `content`, `magma_t::http`, `inx_insert()`, `magma_t::log`, `log_info`, `log_pedantic`, `M_TYPE_UINT64`, `magma`, `MEMORYBUF`, `NULLER`, `PLACER`, `st_cmp_ci_ends()`, `st_cmp_cs_ends()`, `st_free()`, `st_merge`, `multi_t::u64`, and `multi_t::val`.

Referenced by `http_content_refresh()`, and `http_content_start()`.

bool_t http_content_refresh (void)

Refresh the stored contents of the web app templates, static content, and fonts directories.

Returns:

true if all content directories were successfully refreshed, or false on failure.

Definition at line 453 of file content.c.

References `content`, `magma_t::http`, `http_content_load_directory()`, `http_content_load_fonts()`, `http_free_content()`, `inx_alloc()`, `inx_free()`, `M_INX_LINKED`, `magma`, and `st_free()`.

Referenced by `signal_refresh()`.

bool_t http_content_start (void)

Prime the the web app templates, static content, and fonts directories for future use.

Note:

This function makes sure that the web templates, static content, and fonts directory have been properly set, and loads and caches their content for future use.

Returns:

true on success or false on failure.

Definition at line 411 of file content.c.

References `content`, `magma_t::http`, `http_content_load_directory()`, `http_content_load_fonts()`, `http_free_content()`, `inx_alloc()`, `log_pedantic`, `M_INX_LINKED`, `magma`, `NULLER`, `PLACER`, `st_cmp_cs_ends()`, and `st_free()`.

Referenced by `process_start()`.

void http_content_stop (void)

Purge the http contents repository of stored fonts, templates, and static web pages.

Returns:

This function returns no value.

Definition at line 439 of file content.c.

References `content`, and `inx_cleanup()`.

Referenced by `process_stop()`.

void http_free_content (http_content_t * *page*)

LOW: We should use `basename()` and `dirname()` to cleanup path strings.

LOW: These functions should all be renamed to `http_content_XXXX`. The file and directory functions should be updated to use the x64/reentrant alternatives. Specifically `open64/fstat64/readdir64_r`. An update function that can be triggered with a `SIGHUP` would also be nice. Free a stored piece of http content and all its underlying data.

Parameters:

page a pointer to the loaded content page to be freed.

Returns:

This function returns no value.

Definition at line 32 of file content.c.

References `http_content_t::location`, `mm_free()`, `http_content_t::resource`, `st_cleanup()`, and `http_content_t::type`.

Referenced by `http_content_refresh()`, `http_content_start()`, and `http_load_file()`.

http_content_t* http_get_static (stringer_t * *location*)

Get a cached copy of a static web page.

Parameters:

location a pointer to a managed string containing the location of the requested resource.

Returns:

NULL on failure, or a pointer to an http content object with the contents of the requested resource on success.

Definition at line 72 of file content.c.

References content, inx_find(), M_TYPE_STRINGER, and http_content_t::type.

Referenced by http_response().

http_content_t* http_get_template (chr_t * *location*)

Return a template by location.

Parameters:

location a string specifying the location of the template.

Returns:

NULL on failure, or a pointer to an **http_content_t** object containing the template.

Definition at line 88 of file content.c.

References content, inx_find(), M_TYPE_STRINGER, NULLER, and http_content_t::type.

Referenced by http_page_get(), register_print_message(), register_print_step1(), register_print_step2(), and register_print_step3().

bool_t http_load_file (int_t *template*, chr_t * *filename*)

Load file content into the http server repository.

Note:

Each file that is loaded will be cached for retrieval by http clients, with its mime type determined automatically. Any files passed as a template will have the ".template" extension trimmed from the end of its resource path, and any file named 'index.html' will be read as the default directory index path. Each file will also be added to its respective parent page or template inx holder.

Parameters:

template a value specifying whether or not the specified file is a template.

filename a null-terminated string specifying the pathname of the file to be loaded.

Returns:

0 on failure or 1 on success.

Definition at line 154 of file content.c.

References magma_t::content, content, CONTIGUOUS, data, HEAP, magma_t::http, http_free_content(), inx_insert(), magma_t::log, log_info, log_pedantic, M_TYPE_STRINGER, magma, MANAGED_T, MEMORYBUF, mm_alloc(), ns_length_get(), NULLER, PLACER, multi_t::st, st_alloc, st_char_get(), st_cmp_ci_ends(), st_cmp_cs_ends(), st_cmp_cs_eq(), st_dupe(), st_dupe_opts(), st_free(), st_import(), st_length_get(), st_length_int(), st_length_set(), and multi_t::val.

Referenced by http_content_load_directory().

void http_page_free (http_page_t * *page*)

Free an http page object and all its underlying data.

Parameters:

page a pointer to the http page object to be freed.

Definition at line 48 of file content.c.

References `http_page_t::doc_ctx`, `http_page_t::doc_obj`, `mm_free()`, `xml_free_doc()`, `xml_free_parser_ctx()`, `xml_free_xpath_ctx()`, and `http_page_t::xpath_ctx`.

Referenced by `contact_print_form()`, `contact_print_message()`, `http_page_get()`, `portal_print_login()`, `statistics_process()`, `teacher_print_form()`, and `teacher_print_message()`.

`http_page_t* http_page_get (chr_t * location)`

Get a template page and prepare its xml document root for use.

Note:

Each page is affixed with an xpath with a namespace after passing through the xml parser.

Parameters:

location a pointer to a null-terminated string with the pathname of the template to be returned.

Returns:

NULL on failure, or a pointer to the http page object of the requested template.

Definition at line 105 of file `content.c`.

References `http_page_t::content`, `http_page_t::doc_ctx`, `http_page_t::doc_obj`, `http_get_template()`, `http_page_free()`, `log_pedantic`, `mm_alloc()`, `http_content_t::resource`, `st_char_get()`, `st_length_get()`, `xml_create_doc()`, `xml_create_parser_ctx()`, `xml_create_xpath_ctx()`, `xml_xpath_set_namespace()`, and `http_page_t::xpath_ctx`.

Referenced by `contact_business_add_error()`, `contact_print_form()`, `contact_print_message()`, `portal_print_login()`, `statistics_process()`, `teacher_add_error()`, `teacher_print_form()`, and `teacher_print_message()`.

Variable Documentation

`struct { ... } content`

Referenced by `http_content_load_fonts()`, `http_content_refresh()`, `http_content_start()`, `http_content_stop()`, `http_get_static()`, `http_get_template()`, `http_load_file()`, `http_response()`, `mail_discover_encoding()`, `mail_discover_type()`, `mail_get_boundary()`, `mail_mime_boundary()`, `portal_endpoint_config_edit()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_edit()`, `register_print_message()`, `register_print_step1()`, `register_print_step2()`, and `register_print_step3()`.

`inx_t* fonts`

Definition at line 16 of file `content.c`.

`inx_t * pages`

Definition at line 16 of file `content.c`.

`inx_t * templates`

Definition at line 16 of file `content.c`.

magma/servers/http/errors.c File Reference

Error page templates.

```
#include "magma.h"
```

Functions

- void **http_print_301** (**connection_t** *con, **chr_t** *location, **int_t** ssl)
- *Redirect the browser to a new location with an HTTP 301 response.* void **http_print_400** (**connection_t** *con)
- *Return an HTTP 400 bad client request response to the client.* void **http_print_403** (**connection_t** *con)
- *Return an HTTP 403 access denied response to the client.* void **http_print_404** (**connection_t** *con)
- *Return an HTTP 404 location not found response to the client.* void **http_print_405** (**connection_t** *con)
- *Return an HTTP 405 method not allowed response to the client.* void **http_print_500** (**connection_t** *con)
- *Return an HTTP 500 internal server error response to the client.* void **http_print_500_log** (**connection_t** *con, **chr_t** *logmsg)
- *Return an HTTP 500 internal server error response to the client, with additional logging information.* void **http_print_501** (**connection_t** *con)

Return an HTTP 501 method not implemented response to the client.

Detailed Description

Error page templates.

Definition in file **errors.c**.

Function Documentation

void http_print_301 (**connection_t** * con, **chr_t** * location, **int_t** ssl)

Redirect the browser to a new location with an HTTP 301 response.

errors.c

Note:

SSL downgrades are not possible; in order for an ssl upgrade, magma.web.ssl_redirect must first be set.

Parameters:

con the client's http connection handle.

location the url to which the client will be redirected.

ssl if 1, direct to https:// else direct to http:// address.

Returns:

This function returns no value.

LOW: This function should probably move to the response file and be updated to use the cookie/xss helper functions.

Definition at line 23 of file errors.c.

References con_print(), con_secure(), connection_t::http, HTTP_COMPLETE, http_print_403(), http_print_500(), log_pedantic, magma, magma_t::ssl_redirect, st_char_get(), st_cleanup(), st_dupe(), st_free(), st_length_int(), st_merge, and magma_t::web.

Referenced by `contact_process()`, `portal_endpoint()`, `portal_process()`, `portal_upload()`, `register_process()`, and `teacher_process()`.

void http_print_400 (connection_t * con)

Return an HTTP 400 bad client request response to the client.

Parameters:

con the client's http connection handle.

Returns:

This function returns no value.

Definition at line 84 of file `errors.c`.

References `con_write_st()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, and `PLACER`.

Referenced by `http_requeue()`.

void http_print_403 (connection_t * con)

Return an HTTP 403 access denied response to the client.

Parameters:

con the client's http connection handle.

Returns:

This function returns no value.

Definition at line 98 of file `errors.c`.

References `con_write_st()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, and `PLACER`.

Referenced by `http_print_301()`, and `http_requeue()`.

void http_print_404 (connection_t * con)

Return an HTTP 404 location not found response to the client.

Parameters:

con the client's http connection handle.

Returns:

This function returns no value.

Definition at line 112 of file `errors.c`.

References `con_write_st()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, and `PLACER`.

Referenced by `http_requeue()`.

void http_print_405 (connection_t * con)

Return an HTTP 405 method not allowed response to the client.

Parameters:

con the client's http connection handle.

Returns:

This function returns no value.

Definition at line 126 of file errors.c.

References `con_write_st()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, and `PLACER`.

Referenced by `http_requeue()`.

void http_print_500 (connection_t * *con*)

Return an HTTP 500 internal server error response to the client.

Parameters:

con the client's http connection handle.

Returns:

This function returns no value.

Definition at line 140 of file errors.c.

References `con_write_st()`, `debug_hook()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, `log_options`, `M_LOG_CRITICAL`, `M_LOG_STACK_TRACE`, and `PLACER`.

Referenced by `contact_print_message()`, `http_print_301()`, `http_requeue()`, `portal_print_login()`, `register_print_captcha()`, `register_print_message()`, `register_print_step3()`, `statistics_process()`, `teacher_print_form()`, and `teacher_print_message()`.

void http_print_500_log (connection_t * *con*, chr_t * *logmsg*)

Return an HTTP 500 internal server error response to the client, with additional logging information.

Parameters:

con the client's http connection handle.

logmsg a pointer to a null-terminated string with a message describing the cause of the http 500 error.

Returns:

This function returns no value.

Definition at line 159 of file errors.c.

References `con_write_st()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, `log_options`, `M_LOG_CRITICAL`, and `PLACER`.

Referenced by `contact_print_message()`, `register_print_step1()`, and `register_print_step2()`.

void http_print_501 (connection_t * *con*)

Return an HTTP 501 method not implemented response to the client.

Parameters:

con the client's http connection handle.

Returns:

This function returns no value.

Definition at line 174 of file errors.c.

References `con_write_st()`, `connection_t::http`, `HTTP_CONNECTION_CLOSE`, `http_response_header()`, and `PLACER`.

Referenced by `http_requeue()`.

magma/servers/http/http.c File Reference

Functions used to handle HTTP commands and actions.

```
#include "magma.h"
```

Functions

- void **http_close** (**connection_t** *con)
 - *Close a connection corresponding to an http session.* void **http_requeue** (**connection_t** *con)
 - *The main http server requeue entry point state machine for processing client data.* void **http_body** (**connection_t** *con)
 - *Get the body of the http request by reading the value of the Content-Length header.* void **http_process** (**connection_t** *con)
 - *Process data sent by an http client.* void **http_init** (**connection_t** *con)
- Handle a new http client connection.*
-

Detailed Description

Functions used to handle HTTP commands and actions.

Definition in file **http.c**.

Function Documentation

void http_body (connection_t * con)

Get the body of the http request by reading the value of the Content-Length header.

http.c

Note:

Any request errors will be handled directly without returns. This function sets the value of the connection's http.body member.

Parameters:

con a pointer to the connection object of the remote http client.

Returns:

This function returns no value.

Definition at line 78 of file http.c.

References `connection_t::buffer`, `con_read()`, `CONTIGUOUS`, `data`, `HEAP`, `magma_t::http`, `connection_t::http`, `http_data_get()`, `HTTP_DATA_HEADER`, `HTTP_ERROR_400`, `HTTP_RESPOND`, `JOINTED`, `length`, `magma_t::log`, `log_pedantic`, `magma`, `MANAGED_T`, `MAPPED_T`, `connection_t::network`, `size_conv_bl()`, `st_alloc_opts()`, `st_append_opts()`, `st_char_get()`, `st_data_get()`, `st_length_get()`, `st_length_int()`, and `http_data_t::value`.

Referenced by `http_requeue()`.

void http_close (connection_t * con)

Close a connection corresponding to an http session.

Returns:

This function returns no value.

Definition at line 19 of file http.c.

References `con_destroy()`.

Referenced by `http_process()`, and `http_requeue()`.

void http_init (connection_t * con)

Handle a new http client connection.

Parameters:

con a pointer to the http client connection that was just accepted.

Returns:

This function returns no value.

Definition at line 172 of file http.c.

References `con_reverse_enqueue()`, and `http_process()`.

Referenced by `protocol_enqueue()`.

void http_process (connection_t * con)

Process data sent by an http client.

Note:

This is performed in two stages: first read the http method with `http_parse_method()`, then transfer control to `http_parse_header()`. If a failure occurs reading a line of data, or too many protocol violations occur, `http_close()` is called to drop the connection.

Parameters:

con the connection object from which to read data.

Returns:

This function returns no value.

Definition at line 131 of file http.c.

References `con_read_line()`, `server_t::cutoff`, `enqueue()`, `connection_t::http`, `http_close()`, `http_parse_header()`, `HTTP_PARSE_HEADER`, `http_parse_method()`, `http_process()`, `HTTP_READY`, `http_requeue()`, `connection_t::line`, `log_pedantic`, `connection_t::network`, `pl_empty()`, `connection_t::protocol`, `requeue()`, `connection_t::server`, `connection_t::spins`, `server_t::violations`, and `connection_t::violations`.

Referenced by `http_init()`, `http_process()`, and `http_requeue()`.

void http_requeue (connection_t * con)

The main http server requeue entry point state machine for processing client data.

Returns:

This function returns no value.

Definition at line 29 of file http.c.

References `con_status()`, `server_t::cutoff`, `enqueue()`, `connection_t::http`, `http_body()`, `http_close()`, `HTTP_CLOSE`, `HTTP_COMPLETE`, `HTTP_ERROR_400`, `HTTP_ERROR_403`, `HTTP_ERROR_404`, `HTTP_ERROR_405`, `HTTP_ERROR_500`, `HTTP_ERROR_501`, `http_parse_pairs()`, `HTTP_PARSE_PAIRS`, `http_print_400()`, `http_print_403()`, `http_print_404()`, `http_print_405()`, `http_print_500()`, `http_print_501()`, `http_process()`, `HTTP_READ_BODY`, `http_requeue()`, `HTTP_RESPOND`, `http_response()`, `http_session_reset()`, `connection_t::protocol`, `requeue()`, `connection_t::server`, `status`, `server_t::violations`, and `connection_t::violations`.

Referenced by `http_process()`, and `http_requeue()`.

magma/servers/http/parse.c File Reference

Functions used to parse an HTTP request.

```
#include "magma.h"
```

Functions

- **int_t http_parse_origin** (**stringer_t** *s, **placer_t** *output)
- *Get the origin of a resource from a url.* void **http_parse_pairs** (**connection_t** *con)
- *Parse all the GET and POST parameters present in an http client request, and store them with the connection.* void **http_parse_context** (**connection_t** *con, **stringer_t** *application, **stringer_t** *path)
- *Attempt to retrieve a connected user's associated session, by searching the cookie, the POST "session" variable, and the URL.* void **http_parse_header** (**connection_t** *con)
- *Parse and process the current line of input from an http client connection as an http request header, storing data in the connection's http.headers member.* void **http_parse_method** (**connection_t** *con)
- *Parse an http request and determine the request method and location.* **placer_t** **get_header_value_noopt** (**stringer_t** *vstring)
- *Get the simple value of an http header, with any optional parameters stripped away.* **placer_t** **get_header_opt** (**stringer_t** *vstring, **stringer_t** *optname)
- *Get the value of a named optional parameter from an http header value.* **bool_t** **multipart_get_boundary** (**connection_t** *con, **placer_t** *output)

Get the boundary delimiter for a request by a connection specifying a Content-Type of multipart/form-data.

Detailed Description

Functions used to parse an HTTP request.

Definition in file **parse.c**.

Function Documentation

placer_t **get_header_opt** (**stringer_t** * vstring, **stringer_t** * optname)

Get the value of a named optional parameter from an http header value.

Note:

Only the value after the ":" in a header field is passed to this function (in other words, the header name is omitted).

Parameters:

vstring a managed string containing the single http header line value to be parsed.
optname a managed string with the name of the optional parameter to be found.

Returns:

a placer pointing to the named optional parameter of the specified http header value.

Definition at line 337 of file parse.c.

References **pl_null()**, **pl_trim_start()**, **placer_t**, **st_cmp_cs_eq()**, **tok_get_count_st()**, and **tok_get_st()**.

Referenced by **multipart_get_boundary()**, and **portal_upload()**.

placer_t get_header_value_noopt (stringer_t * vstring)

Get the simple value of an http header, with any optional parameters stripped away.

Note:

Only the value after the ":" in a header field is passed to this function (in other words, the header name is omitted).

Parameters:

vstring a managed string containing the single http header line value to be parsed.

Returns:

a placer pointing to the option-free header value of the specified input line.

Definition at line 314 of file parse.c.

References `pl_init()`, `placer_t`, `st_char_get()`, `st_length_get()`, `tok_get_count_st()`, and `tok_get_st()`.

Referenced by `http_parse_header()`, and `multipart_get_boundary()`.

void http_parse_context (connection_t * con, stringer_t * application, stringer_t * path)

Attempt to retrieve a connected user's associated session, by searching the cookie, the POST "session" variable, and the URL.

Parameters:

con a pointer to the connection object to be queried for a session id.

application a managed string containing the application associated with the connection's pending request.

path a managed string containing the path associated with the connection's pending request.

Returns:

This function returns no value.

TODO: Develop better logic for handling cookies. Namely add the ability to forcibly trigger the Set-Cookie entity and if necessary delete the existing cookie.

Definition at line 120 of file parse.c.

References `connection_t::http`, `HTTP_METHOD_POST`, `json_decref_d`, `json_loads_d`, `json_object_get_d`, `json_string_value_d`, `NULLER`, `pl_init()`, `placer_t`, `sess_get()`, `st_char_get()`, `st_cmp_ci_starts()`, `st_length_get()`, `tok_get_pl()`, and `tok_get_st()`.

Referenced by `portal_endpoint()`, `portal_process()`, and `portal_upload()`.

void http_parse_header (connection_t * con)

Parse and process the current line of input from an http client connection as an http request header, storing data in the connection's `http.headers` member.

Note:

If no more http headers can be read, control is returned by setting the connection `http.mode` to `HTTP_RESPOND`. Special actions are taken to store the "Host", "User-Agent", "Cookie", and "Connection" headers.

Parameters:

con the connection object of the http client to be read, and to store the results of the operation.

Returns:

This function returns no value.

LOW: Should we bother to throw an error if *con*->http.connection isn't HTTP_CONNECTION_NEUTRAL?

Definition at line 178 of file parse.c.

References *con*_print(), CONTIGUOUS, data, get_header_value_noopt(), HEAP, magma_t::http, connection_t::http, HTTP_CONNECTION_CLOSE, HTTP_CONNECTION_KEEPALIVE, http_data_free(), http_data_header_parse(), HTTP_ERROR_500, HTTP_RESPOND, int32_conv_bl(), inx_alloc(), inx_insert(), magma_t::log, log_info, lower_st(), M_INX_LINKED, M_TYPE_STRINGER, magma, MANAGED_T, http_data_t::name, PLACER, placer_t, multi_t::st, st_char_get(), st_cmp_ci_eq(), st_dupe_opts(), st_length_get(), st_length_int(), st_length_set(), st_search_cs(), multi_t::val, and http_data_t::value.

Referenced by http_process().

void http_parse_method (connection_t * *con*)

Parse an http request and determine the request method and location.

Note:

This function returns no value but sets the internal method, location, and state of the underlying connection object.

Parameters:

con the client connection making an http request.

Returns:

This function returns no value.

Definition at line 270 of file parse.c.

References CONTIGUOUS, HEAP, magma_t::http, connection_t::http, HTTP_METHOD_CONNECT, HTTP_METHOD_DELETE, HTTP_METHOD_GET, HTTP_METHOD_HEAD, HTTP_METHOD_OPTIONS, HTTP_METHOD_POST, HTTP_METHOD_PUT, HTTP_METHOD_TRACE, HTTP_METHOD_UNSUPPORTED, HTTP_PARSE_HEADER, connection_t::line, magma_t::log, log_info, magma, MANAGED_T, connection_t::network, PLACER, placer_t, st_char_get(), st_cmp_ci_starts(), st_dupe_opts(), st_length_int(), tok_get_count_st(), and tok_get_pl().

Referenced by http_process().

int_t http_parse_origin (stringer_t * *s*, placer_t * *output*)

Get the origin of a resource from a url.

Note:

Usually we'll be give the value of the Origin header field to work with so we'll end up returning the entire string, but if we had to fall back and use the Referrer instead this should strip off the path portion.

Parameters:

s a managed string containing the url to be parsed.

output a pointer to a placer that will be set to point to the origin string inside the user-supplied url.

Returns:

0 on success or -1 on failure.

Definition at line 23 of file parse.c.

References pl_init(), PLACER, st_cmp_ci_starts(), st_data_get(), st_empty_out(), and st_uchar_get().

Referenced by http_response_allow_cross().

void http_parse_pairs (connection_t * con)

Parse all the GET and POST parameters present in an http client request, and store them with the connection.

Note:

All valid field parameters will be stored in the connection's http.pairs object.

Parameters:

con a pointer to the connection object generating the http requesting being processed.

Returns:

This function returns no value.

Definition at line 78 of file parse.c.

References count, data, connection_t::http, http_data_free(), HTTP_DATA_GET, HTTP_DATA_POST, http_data_value_parse(), HTTP_ERROR_500, inx_alloc(), M_INX_LINKED, pl_init(), PLACER, placer_t, st_char_get(), st_length_get(), st_search_cs(), tok_get_count_st(), and tok_get_st().

Referenced by http_requeue(), and http_response().

bool_t multipart_get_boundary (connection_t * con, placer_t * output)

Get the boundary delimiter for a request by a connection specifying a Content-Type of multipart/form-data.

Parameters:

con a pointer to the connection object of the client making the http request.

output a pointer to the address of a managed string that will receive a copy of the value of the boundary string on success.

Returns:

true on success or false on failure.

Definition at line 374 of file parse.c.

References get_header_opt(), get_header_value_noopt(), http_data_get(), HTTP_DATA_HEADER, NULLER, pl_empty(), placer_t, and http_data_t::value.

Referenced by portal_upload().

magma/servers/imap/parse.c File Reference

Functions used to handle IMAP commands and actions.

```
#include "magma.h"
```

Functions

- **int_t imap_get_type_ar** (imap_arguments_t *array, size_t element)
- *TODO: The logic here is just plain ugly. Specifically the logic used to read from the network and advance the buffers.* void * **imap_get_ptr** (imap_arguments_t *array, size_t element)
- *Return the value of a specified object in an array as a generic pointer.* stringer_t * **imap_get_st_ar** (imap_arguments_t *array, size_t element)
- *Return the value of a specified object in an array as a managed string.* imap_arguments_t * **imap_get_ar_ar** (imap_arguments_t *array, size_t element)
- *Return the value of a specified object in an array as an imap arguments array.* int_t **imap_parse_astring** (stringer_t **output, chr_t **start, size_t *length)
- *Extract the contents of an atomic string and advance the position of the parser stream.* int_t **imap_parse_nstring** (stringer_t **output, chr_t **start, size_t *length, chr_t type)
- **int_t imap_parse_qstring** (stringer_t **output, chr_t **start, size_t *length)
- *Extract the contents of a quoted string and advance the position of the parser stream.* int_t **imap_parse_literal** (connection_t *con, stringer_t **output, chr_t **start, size_t *length)
- *Extract the contents of a literal string and advance the position of the parser stream.* int_t **imap_parse_array** (int_t recursion, connection_t *con, imap_arguments_t **array, chr_t **start, size_t *length)
- *Extract the contents of an array argument and advance the position of the parser stream.* int_t **imap_parse_arguments** (connection_t *con, chr_t **start, size_t *length)
- *Parse a chunk of input into an array of IMAP arguments.* void **imap_command_log_safe** (stringer_t *line)
- *LOW: Have the merge related kinks in the argument parsing system been resolved? If so, can this function be returned to the graveyard?* int_t **imap_command_parser** (connection_t *con)

Parse an input line containing an IMAP command.

Detailed Description

Functions used to handle IMAP commands and actions.

Definition in file **parse.c**.

Function Documentation

void imap_command_log_safe (stringer_t * *line*)

LOW: Have the merge related kinks in the argument parsing system been resolved? If so, can this function be returned to the graveyard?

Definition at line 790 of file parse.c.

References chr_whitespace(), log_info, mm_dupe(), mm_free(), PLACER, st_char_get(), st_empty_out(), st_length_int(), and st_search_ci().

Referenced by imap_command_parser().

int_t imap_command_parser (connection_t * con)

Parse an input line containing an IMAP command.

parse.c

Note:

This function updates the protocol-specific IMAP structure with parsed values for the session's tag, command, and arguments fields. Special handling is performed for any command that is preceded by a "UID" prefix.

Parameters:

con the IMAP client connection issuing the command.

Returns:

1 on success or < 0 on error. -1: the tag could not be read. -2: the IMAP command could not be read. -3: the arguments to the IMAP command could not be read.

Definition at line 890 of file parse.c.

References `ar_free()`, `magma_t::imap`, `connection_t::imap`, `imap_command_log_safe()`, `imap_parse_arguments()`, `imap_parse_astring()`, `length`, `connection_t::line`, `magma_t::log`, `log_pedantic`, `magma`, `connection_t::network`, `PLACER`, `st_cleanup()`, `st_cmp_ci_eq()`, `st_data_get()`, `st_free()`, and `st_length_get()`.

Referenced by `imap_process()`.

imap_arguments_t* imap_get_ar_ar (imap_arguments_t * array, size_t element)

Return the value of a specified object in an array as an imap arguments array.

Parameters:

array a pointer to an imap arguments array.

element the zero-based index of the object in the imap arguments array to be examined.

Returns:

NULL on failure, or a pointer to the specified object's value as an imap arguments array on success.

Definition at line 104 of file parse.c.

References `ar_length_get()`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_get_type_ar()`, and `log_pedantic`.

Referenced by `imap_fetch_body()`, `imap_id()`, `imap_parse_dataitems()`, `imap_search_messages_inner()`, and `imap_status()`.

void* imap_get_ptr (imap_arguments_t * array, size_t element)

Return the value of a specified object in an array as a generic pointer.

Parameters:

array a pointer to an imap arguments array.

element the zero-based index of the object in the imap arguments array to be examined.

Returns:

NULL on failure, or a pointer to the specified object's value on success.

Definition at line 50 of file parse.c.

References `ar_length_get()`, and `log_pedantic`.

Referenced by `imap_append()`, `imap_fetch_body()`, and `imap_store()`.

`stringer_t* imap_get_st_ar (imap_arguments_t * array, size_t element)`

Return the value of a specified object in an array as a managed string.

Parameters:

array a pointer to an imap arguments array.

element the zero-based index of the object in the imap arguments array to be examined.

Returns:

NULL on failure, or a pointer to the specified object's value as a managed string on success.

Definition at line 75 of file `parse.c`.

References `ar_length_get()`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_get_type_ar()`, and `log_pedantic`.

Referenced by `imap_append()`, `imap_copy()`, `imap_create()`, `imap_delete()`, `imap_examine()`, `imap_fetch()`, `imap_fetch_body()`, `imap_fetch_body_header()`, `imap_fetch_body_tag()`, `imap_flag_parse()`, `imap_id()`, `imap_list()`, `imap_login()`, `imap_lsub()`, `imap_parse_dataitems()`, `imap_rename()`, `imap_search_messages_inner()`, `imap_select()`, `imap_status()`, `imap_store()`, and `imap_subscribe()`.

`int_t imap_get_type_ar (imap_arguments_t * array, size_t element)`

TODO: The logic here is just plain ugly. Specifically the logic used to read from the network and advance the buffers.

Get the type code for a specified object in an array.

Note:

Valid return values include `IMAP_ARGUMENT_TYPE_ARRAY`, `IMAP_ARGUMENT_TYPE_ASTRING`, `IMAP_ARGUMENT_TYPE_QSTRING`, `IMAP_ARGUMENT_TYPE_NSTRING`, and `IMAP_ARGUMENT_TYPE_LITERAL`.

Parameters:

array a pointer to an imap arguments array.

element the zero-based index of the object in the imap arguments array to be examined.

Returns:

`IMAP_ARGUMENT_TYPE_EMPTY` on failure, or the element type code of the specified object on success.

Definition at line 24 of file `parse.c`.

References `ar_length_get()`, `IMAP_ARGUMENT_TYPE_EMPTY`, and `log_pedantic`.

Referenced by `imap_append()`, `imap_copy()`, `imap_create()`, `imap_delete()`, `imap_examine()`, `imap_fetch()`, `imap_fetch_body()`, `imap_fetch_body_header()`, `imap_flag_parse()`, `imap_get_ar_ar()`, `imap_get_st_ar()`, `imap_id()`, `imap_list()`, `imap_login()`, `imap_lsub()`, `imap_parse_dataitems()`, `imap_rename()`, `imap_search_messages_inner()`, `imap_select()`, `imap_status()`, `imap_store()`, and `imap_subscribe()`.

`int_t imap_parse_arguments (connection_t * con, chr_t ** start, size_t * length)`

Parse a chunk of input into an array of IMAP arguments.

See also:

imap_parse_qstring(), imap_parse_literal(), imap_parse_array(), imap_parse_astring()

Note:

This function scans a string for an array of arguments, performing the following logic when a particular character is encountered: `:` Parse argument as quoted string with **imap_parse_qstring()**. `{` : Parse argument as literal string with **imap_parse_literal()**. `(` or `[` : Parse argument as array with **imap_parse_astring()**. `other` : Defaults to parsing argument as atomic string with **imap_parse_atomic()**. The supplied start and length pointers will be updated to reflect the input stream if the quoted string is parsed successfully.

Parameters:

start the address of a pointer to the start of the buffer to be parsed, that will be continually updated to point to the next argument in the sequence during the parsing loop.

length a pointer to a `size_t` variable that contains the length of the string to be parsed, that will be continually updated with the input stream position during the parsing loop.

Returns:

-1 on parsing error or 1 on success.

Definition at line 713 of file `parse.c`.

References `ar_append()`, `ar_free()`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `IMAP_ARGUMENT_TYPE_ASTRING`, `IMAP_ARGUMENT_TYPE_LITERAL`, `IMAP_ARGUMENT_TYPE_QSTRING`, `imap_parse_array()`, `imap_parse_astring()`, `imap_parse_literal()`, and `imap_parse_qstring()`.

Referenced by `imap_command_parser()`.

```
int_t imap_parse_array (int_t recursion, connection_t * con, imap_arguments_t ** array,  
chr_t ** start, size_t * length)
```

Extract the contents of an array argument and advance the position of the parser stream.

Note:

This function expects as input a string beginning either with `'` or `[`.

Parameters:

length a pointer to a `size_t` variable that contains the length of the string to be parsed, and that will be updated to reflect the length of the remainder of the input string that follows the parsed literal string.

recursion d

con the client IMAP connection passing the array as input to the server.

array -

start -

length -

Returns:

Definition at line 599 of file `parse.c`.

References `ar_append()`, `ar_free()`, `IMAP_ARGUMENT_TYPE_ARRAY`, `IMAP_ARGUMENT_TYPE_LITERAL`, `IMAP_ARGUMENT_TYPE_NSTRING`, `IMAP_ARGUMENT_TYPE_QSTRING`, `IMAP_ARRAY_RECURSION_LIMIT`, `imap_parse_array()`, `imap_parse_literal()`, `imap_parse_nstring()`, `imap_parse_qstring()`, `log_pedantic`, and `type()`.

Referenced by `imap_parse_arguments()`, and `imap_parse_array()`.

int_t imap_parse_astring (stringer_t ** output, chr_t ** start, size_t * length)

Extract the contents of an atomic string and advance the position of the parser stream.

Note:

This function scans a string, expecting printable ASCII characters until it encounters a space, , , (, or [. If any other character is encountered before that point, an error will be returned. The supplied start and length pointers will be updated to reflect the input stream if the quoted string is parsed successfully.

Parameters:

output the address of a managed string that will receive a copy of the atomic string's contents on success, or NULL on failure.

start the address of a pointer to the start of the buffer to be parsed, that will also be updated to point to the next argument in the sequence on success.

length a pointer to a size_t variable that contains the length of the string to be parsed, and which will be updated to reflect the length of the remainder of the input string that follows the parsed atomic string.

Returns:

-1 on general error or if an invalid character was encountered, or 1 if the supplied atomic string was valid.

Definition at line 158 of file parse.c.

References log_pedantic, and st_import().

Referenced by imap_command_parser(), and imap_parse_arguments().

int_t imap_parse_literal (connection_t * con, stringer_t ** output, chr_t ** start, size_t * length)

Extract the contents of a literal string and advance the position of the parser stream.

Note:

This function expects as input a string beginning with '{' and followed by a numerical string, an optional '+', and a closing '}'. After reading in the numerical size parameter, it then attempts to read in that many bytes of input from the network stream.

Parameters:

con the client IMAP connection passing the literal string as input to the server.

output the address of a managed string that will receive a copy of the literal string's contents on success, or NULL on failure or if it is zero length.

start the address of a pointer to the start of the buffer to be parsed (beginning with '{'), that will also be updated to point to the next argument in the sequence on success.

length a pointer to a size_t variable that contains the length of the string to be parsed, and that will be updated to reflect the length of the remainder of the input string that follows the parsed literal string.

Returns:

-1 on general or parse error or if an enclosing pair of double quotes was not found, or 1 if the supplied quoted string was valid.

Definition at line 385 of file parse.c.

References connection_t::buffer, con_read(), con_read_line(), con_write_bl(), connection_t::line, line_pl_st(), log_pedantic, mm_copy(), mm_move(), connection_t::network, number, pl_empty(), pl_length_get(), pl_null(), st_alloc, st_char_get(), st_free(), st_length_set(), and uint64_conv_bl().

Referenced by imap_parse_arguments(), and imap_parse_array().

int_t imap_parse_nstring (stringer_t ** *output*, chr_t ** *start*, size_t * *length*, chr_t *type*)

Definition at line 201 of file parse.c.

References log_error, and st_import().

Referenced by imap_parse_array().

int_t imap_parse_qstring (stringer_t ** *output*, chr_t ** *start*, size_t * *length*)

Extract the contents of a quoted string and advance the position of the parser stream.

Note:

This function scans a string beginning with `""` for a terminating `""`, returning an error if `or` is encountered first. It is also able to handle escaped characters. The supplied *start* and *length* pointers will be updated to reflect the input stream if the quoted string is parsed successfully.

Parameters:

output the address of a managed string that will receive a copy of the quoted string's contents on success, or NULL on failure or if it is zero length.

start the address of a pointer to the start of the buffer to be parsed (beginning with `\`), that will also be updated to point to the next argument in the sequence on success.

length a pointer to a `size_t` variable that contains the length of the string to be parsed, and which will be updated to reflect the length of the remainder of the input string that follows the parsed quoted string.

Returns:

-1 on general error or if an enclosing pair of double quotes was not found, or 1 if the supplied quoted string was valid.

Definition at line 255 of file parse.c.

References log_pedantic, st_alloc, st_char_get(), and st_length_set().

Referenced by imap_parse_arguments(), and imap_parse_array().

magma/servers/pop/parse.c File Reference

The POP protocol parsers.

```
#include "magma.h"
```

Functions

- **bool_t pop_num_parse** (**connection_t** *con, uint64_t *outnum, **bool_t** required)
- *Parse the argument to a POP3 command as an unsigned number.* **bool_t pop_top_parse** (**connection_t** *con, uint64_t *number, uint64_t *lines)
- *Parse the arguments to a POP3 TOP command.* **stringer_t** * **pop_pass_parse** (**connection_t** *con)
- *Parse a POP3 PASS command.* **stringer_t** * **pop_user_parse** (**connection_t** *con)

Parse a POP3 USER command.

Detailed Description

The POP protocol parsers.

Definition in file **parse.c**.

Function Documentation

bool_t pop_num_parse (**connection_t** * con, uint64_t * outnum, **bool_t** required)

Parse the argument to a POP3 command as an unsigned number.

parse.c

Parameters:

con the connection across which the pop command was issued.

outnum a pointer to an unsigned 64-bit integer to receive the numerical argument of the pop command on success.

required specifies whether or not a parameter is required.

Returns:

true if the pop command contained a valid unsigned number or false on failure.

Definition at line 22 of file parse.c.

References `length`, `connection_t::line`, `log_pedantic`, `connection_t::network`, `st_data_get()`, `st_length_get()`, and `uint64_conv_bl()`.

Referenced by `pop_dele()`, `pop_list()`, `pop_retr()`, and `pop_uidl()`.

stringer_t* **pop_pass_parse** (**connection_t** * con)

Parse a POP3 PASS command.

Note:

This function will stop reading in password characters when an invalid character is encountered.

Parameters:

con the POP3 client connection that issued the PASS command.

Returns:

a managed string containing the user supplied password, or NULL on failure.

Definition at line 196 of file parse.c.

References `length`, `connection_t::line`, `log_pedantic`, `connection_t::network`, `st_data_get()`, `st_import()`, and `st_length_get()`.

Referenced by `pop_pass()`.

bool_t pop_top_parse (connection_t * *con*, uint64_t * *number*, uint64_t * *lines*)

Parse the arguments to a POP3 TOP command.

Parameters:

con the POP3 client connection that issued the TOP command.

number a pointer to an unsigned 64 bit integer to receive the value of the TOP command's message sequence number.

lines a pointer to an unsigned 64 bit integer to receive the value of the TOP command's line number count.

Returns:

true on success or false if an error was encountered.

Definition at line 94 of file parse.c.

References `length`, `connection_t::line`, `log_pedantic`, `connection_t::network`, `st_data_get()`, `st_length_get()`, and `uint64_conv_bl()`.

Referenced by `pop_top()`.

stringer_t* pop_user_parse (connection_t * *con*)

Parse a POP3 USER command.

Note:

The username can be at most 255 characters and will be returned in lowercase. This function will stop reading in username characters when an invalid character is encountered.

Parameters:

con the POP3 client connection that issued the USER command.

Returns:

a managed string containing the validated username, or NULL on failure.

Definition at line 257 of file parse.c.

References `length`, `connection_t::line`, `log_pedantic`, `lower_chr()`, `connection_t::network`, `st_data_get()`, `st_import()`, and `st_length_get()`.

Referenced by `pop_user()`.

magma/servers/smtp/parse.c File Reference

Functions used to parse command parameters from SMTP clients.

```
#include "magma.h"
```

Functions

- **stringer_t * smtp_parse_rcpt_to** (connection_t *con)
- **stringer_t * smtp_parse_auth** (stringer_t *data)
- *Parse the parameter specified to an SMTP AUTH PLAIN or SMTP AUTH LOGIN command.* **stringer_t * smtp_parse_mail_from_path** (connection_t *con)
- **stringer_t * smtp_parse_helo_domain** (connection_t *con)

Parse the domain specified as the parameter to an SMTP HELO or EHLO command.

Detailed Description

Functions used to parse command parameters from SMTP clients.

Definition in file **parse.c**.

Function Documentation

stringer_t* smtp_parse_auth (stringer_t * data)

Parse the parameter specified to an SMTP AUTH PLAIN or SMTP AUTH LOGIN command.

parse.c

Note:

This function will stop reading input when an invalid base64-encoding character is encountered.

Parameters:

con the SMTP client connection issuing the AUTH command.

Returns:

a managed string containing the domain specified by the AUTH command, or NULL on failure.

Definition at line 118 of file parse.c.

References `length`, `log_pedantic`, `st_empty_out()`, and `st_import()`.

Referenced by `smtp_auth_login()`, and `smtp_auth_plain()`.

stringer_t* smtp_parse_helo_domain (connection_t * con)

Parse the domain specified as the parameter to an SMTP HELO or EHLO command.

Note:

Valid domain names may only contain letters, numbers, periods and hyphens. This function will return a lowercase domain name, and stop reading input when an invalid character is encountered.

Parameters:

con the SMTP client connection issuing the command.

Returns:

a managed string containing the domain specified by the HELO command, or NULL on failure.

Definition at line 315 of file parse.c.

References connection_t::command, magma_t::helo_length_limit, length, connection_t::line, log_pedantic, lower_chr(), magma, connection_t::network, pl_char_get(), pl_empty(), pl_length_get(), pl_length_int(), pl_trim_end(), magma_t::smtp, st_import(), and command_t::string.

Referenced by smtp_ehlo(), and smtp_helo().

stringer_t* smtp_parse_mail_from_path (connection_t * con)

Extract the provided path from the command line Reverse-path = Path Forward-path = Path Path = "<" [A-d-l ":"] Mailbox ">" A-d-l = At-domain *("," A-d-l) At-domain = @ domain

Parameters:

con A connection structure which presumably contains a the MAIL FROM value inside the line buffer.

Returns:

Returns a stringer with the path that was extracted or NULL to indicate a problem.

Definition at line 173 of file parse.c.

References magma_t::address_length_limit, connection_t::command, CONSTANT, length, connection_t::line, log_pedantic, lower_chr(), magma, connection_t::network, pl_char_get(), pl_empty(), pl_length_get(), pl_length_int(), pl_trim(), pl_trim_end(), PLACER, placer_t, connection_t::smtp, magma_t::smtp, st_cmp_ci_starts(), st_cmp_cs_eq(), st_import(), command_t::string, smtp_session_t::suggested_eight_bit, smtp_session_t::suggested_length, tok_get_bl(), tok_get_count_bl(), tok_get_pl(), and uint64_conv_pl().

Referenced by smtp_mail_from().

stringer_t* smtp_parse_rcpt_to (connection_t * con)

LOW: The list of characters allowed in an email address needs to be verified against the RFC's.
 LOW: The parser should use different lists of valid characters for the the local and domain portions of an email address. Extract the provided path from the command line Reverse-path = Path Forward-path = Path Path = "<" [A-d-l ":"] Mailbox ">" A-d-l = At-domain *("," A-d-l) At-domain = @ domain

Parameters:

con A connection structure which presumably contains a the RCPT TO value inside the line buffer.

Returns:

Returns a stringer with the path that was extracted or NULL to indicate a problem.

Definition at line 29 of file parse.c.

References magma_t::address_length_limit, connection_t::command, length, connection_t::line, log_pedantic, lower_chr(), magma, connection_t::network, pl_char_get(), pl_empty(), pl_length_get(), pl_length_int(), pl_trim_end(), magma_t::smtp, st_import(), and command_t::string.

Referenced by smtp_rcpt_to().

magma/web/portal/parse.c File Reference

json-rpc request parameter parsers for the portal.

```
#include "magma.h"
```

Functions

- **inx_t * portal_parse_json_str_array** (json_t *json, size_t *nout)
- *Parse a json string array into a linked list of managed strings.* **int_t portal_parse_context** (stringer_t *context)
- *Parse the context of a requested folder.* **int_t portal_parse_action** (stringer_t *action)
- **int_t portal_parse_sections** (json_t *array, uint32_t *sections)

Parse the json-rpc messages.load parameter "section" from an array of strings into a bitmask of section flags.

Detailed Description

json-rpc request parameter parsers for the portal.

Definition in file **parse.c**.

Function Documentation

int_t portal_parse_action (stringer_t * action)

Note:

This function is currently not called anywhere in the code and may be subject to removal.

Definition at line 126 of file parse.c.

References **PLACER**, **PORTAL_ENDPOINT_ACTION_ADD**, **PORTAL_ENDPOINT_ACTION_INVALID**, **PORTAL_ENDPOINT_ACTION_LIST**, **PORTAL_ENDPOINT_ACTION_REMOVE**, **PORTAL_ENDPOINT_ACTION_REPLACE**, and **st_cmp_ci_eq()**.

int_t portal_parse_context (stringer_t * context)

Parse the context of a requested folder.

Note:

If no context is specified, the "mail" context is assumed (**PORTAL_ENDPOINT_CONTEXT_MAIL**).

Parameters:

context a managed string containing the context (supported values are "mail", "contacts", "settings", and "help").

Returns:

PORTAL_ENDPOINT_CONTEXT_INVALID on failure, or the flag of the determined context on success.

Definition at line 95 of file parse.c.

References **PLACER**, **PORTAL_ENDPOINT_CONTEXT_CONTACTS**, **PORTAL_ENDPOINT_CONTEXT_HELP**, and **PORTAL_ENDPOINT_CONTEXT_INVALID**.

PORTAL_ENDPOINT_CONTEXT_MAIL, PORTAL_ENDPOINT_CONTEXT_SETTINGS, and
st_cmp_ci_eq().

Referenced by portal_endpoint_folders_add(), portal_endpoint_folders_remove(),
portal_endpoint_folders_rename(), and portal_endpoint_folders_tags().

inx_t* portal_parse_json_str_array (json_t * json, size_t * nout)

Parse a json string array into a linked list of managed strings.

parse.c

Parameters:

json a json object containing an array of strings.

nout if not NULL, an optional pointer to a size_t to receive the item count of the json string array.

Returns:

NULL on failure, or a pointer to an inx holder containing the specified json array contents as a collection of managed strings.

Definition at line 21 of file parse.c.

References count, inx_alloc(), inx_free(), inx_insert(), json_array_get_d, json_array_size_d,
json_string_value_d, log_error, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, ns_length_get(),
st_free(), st_import(), multi_t::u64, and multi_t::val.

Referenced by portal_endpoint_messages_send().

int_t portal_parse_sections (json_t * array, uint32_t * sections)

Parse the json-rpc messages.load parameter "section" from an array of strings into a bitmask of section flags.

Parameters:

array a pointer to the json object representing the "section" string array of the json request message.

sections a pointer to an unsigned 32 bit integer that will receive the translated section flags on success.

Returns:

< 0 on error (-3 for an internal server error, -2 if an unknown flag was received, or -1 on syntax error), 0 for an empty sections array, or 1 on success.

Definition at line 162 of file parse.c.

References count, json_array_get_d, json_array_size_d, json_string_value_d, NULLER, PLACER,
PORTAL_ENDPOINT_MESSAGE_ATTACHMENTS, PORTAL_ENDPOINT_MESSAGE_BODY,
PORTAL_ENDPOINT_MESSAGE_HEADER, PORTAL_ENDPOINT_MESSAGE_INFO,
PORTAL_ENDPOINT_MESSAGE_META, PORTAL_ENDPOINT_MESSAGE_SECURITY,
PORTAL_ENDPOINT_MESSAGE_SERVER, PORTAL_ENDPOINT_MESSAGE_SOURCE, and
st_cmp_ci_eq().

Referenced by portal_endpoint_messages_load().

magma/servers/http/response.c File Reference

Functions for fulfilling responses to http client requests.

```
#include "magma.h"
```

Functions

- **chr_t * http_response_status (int_t status)**
- *Get a descriptive string for a numerical http status code.* **stringer_t * http_response_cookie (connection_t *con)**
- **stringer_t * http_response_allow_cross (connection_t *con)**
- **stringer_t * http_response_connection (connection_t *con, int_t force)**
- *Get an appropriate value for the Connection header of an http response.* **void http_response_options (connection_t *con)**
- *Return a response to an http OPTIONS request.* **void http_response_header (connection_t *con, int_t status, stringer_t *type, size_t len)**
- *Send a full set of http response headers to the remote client.* **void http_response (connection_t *con)**

Make a response to an http client request.

Detailed Description

Functions for fulfilling responses to http client requests.

Definition in file **response.c**.

Function Documentation

void http_response (connection_t * con)

Make a response to an http client request.

response.c

Note:

The following http methods aren't supported: PUT, DELETE, HEAD, TRACE, and CONNECT. The http server will first attempt to retrieve the requested url as a static page; otherwise the following special locations are supported: /portal, /portal/camel, /register, /contact, /report_abuse, /teacher, and /statistics.

Returns:

This function returns no value.

Definition at line 424 of file response.c.

References magma_t::abuse, magma_t::admin, con_write_st(), magma_t::contact, contact_process(), content, connection_t::http, HTTP_ERROR_403, HTTP_ERROR_404, HTTP_ERROR_405, HTTP_ERROR_500, HTTP_ERROR_501, http_get_static(), HTTP_METHOD_CONNECT, HTTP_METHOD_DELETE, HTTP_METHOD_HEAD, HTTP_METHOD_OPTIONS, HTTP_METHOD_POST, HTTP_METHOD_PUT, HTTP_METHOD_TRACE, HTTP_METHOD_UNSUPPORTED, http_parse_pairs(), HTTP_READ_BODY, HTTP_RESPOND, http_response_header(), http_response_options(), magma, NULLER, PLACER, portal_endpoint(), portal_process(), portal_upload(), register_process(), magma_t::registration, http_content_t::resource, st_cmp_ci_starts(), st_cmp_cs_eq(), st_cmp_cs_starts(), st_length_get(), magma_t::statistics, statistics_process(), teacher_process(), http_content_t::type, and magma_t::web.

Referenced by `http_requeue()`.

stringer_t* http_response_allow_cross (connection_t * con)

Definition at line 250 of file `response.c`.

References `http_data_get()`, `HTTP_DATA_HEADER`, `http_parse_origin()`, `MANAGEDBUF`, `PLACER`, `placer_t`, `st_append`, `st_char_get()`, `st_cmp_ci_eq()`, `st_length_int()`, `st_quick()`, `upper_st()`, and `http_data_t::value`.

Referenced by `http_response_header()`, and `http_response_options()`.

stringer_t* http_response_connection (connection_t * con, int_t force)

Get an appropriate value for the Connection header of an http response.

Note:

If `magma.http.close` was set in the configuration options, the connection will be closed immediately.

Parameters:

con a pointer to the connection object of the outgoing response.

force either `HTTP_CONNECTION_CLOSE`, `HTTP_CONNECTION_NEUTRAL`, or `HTTP_CONNECTION_KEEPALIVE`. `HTTP_CONNECTION_CLOSE` will result in the connection being terminated immediately, `HTTP_CONNECTION_KEEPALIVE` will result in a "Connection: keep-alive" response, and `HTTP_CONNECTION_NEUTRAL` will not result in any output.

Returns:

a pointer to a managed string containing the http Connection header field that should be used for the response.

Definition at line 293 of file `response.c`.

References `CONTIGUOUS`, `HEAP`, `connection_t::http`, `magma_t::http`, `HTTP_CLOSE`, `HTTP_CONNECTION_CLOSE`, `HTTP_CONNECTION_KEEPALIVE`, `magma`, `MANAGED_T`, `PLACER`, and `st_dupe_opts()`.

Referenced by `http_response_header()`, and `http_response_options()`.

stringer_t* http_response_cookie (connection_t * con)

Definition at line 195 of file `response.c`.

References `con_secure()`, `HEAP`, `connection_t::http`, `HTTP_COOKIE_DELETE`, `HTTP_COOKIE_SET`, `JOINED`, `MANAGED_T`, `MANAGEDBUF`, `PLACER`, `st_append`, `st_aprint_opts()`, `st_char_get()`, `st_length_int()`, `st_quick()`, and `time_print_gmt()`.

Referenced by `http_response_header()`.

void http_response_header (connection_t * con, int_t status, stringer_t * type, size_t len)

Send a full set of http response headers to the remote client.

Note:

If the mode is HTTP_RESPOND it will be changed to HTTP_COMPLETE, to tell the http requeue function to reset the context and enqueue request processor.

Parameters:

con a pointer to the connection object across which the response will be sent.
status an integer containing the http status code for the response.
type a managed string containing the value of the Content-Type header.
len the value of the Content-Length header.

Returns:

This function returns no value.

Definition at line 369 of file response.c.

References `con_print()`, `magma_t::http`, `connection_t::http`, `HTTP_COMPLETE`, `HTTP_CONNECTION_NEUTRAL`, `HTTP_RESPOND`, `http_response_allow_cross()`, `http_response_connection()`, `http_response_cookie()`, `http_response_status()`, `magma`, `MANAGEDBUF`, `mm_wipe()`, `st_char_get()`, `st_cleanup()`, `st_length_int()`, and `time_print_gmt()`.

Referenced by `contact_print_form()`, `contact_print_message()`, `http_print_400()`, `http_print_403()`, `http_print_404()`, `http_print_405()`, `http_print_500()`, `http_print_500_log()`, `http_print_501()`, `http_response()`, `portal_endpoint()`, `portal_endpoint_attachments_progress()`, `portal_endpoint_error()`, `portal_endpoint_response()`, `portal_endpoint_search()`, `portal_print_login()`, `register_print_message()`, `register_print_step1()`, `register_print_step2()`, `register_print_step3()`, `statistics_process()`, and `teacher_print_form()`.

void http_response_options (connection_t * con)

Return a response to an http OPTIONS request.

Parameters:

con the client connection which made the OPTIONS request.

Returns:

This function returns no value.

LOW: I couldn't find a definitive answer about whether the OPTION response should always return a Content-Type of text/plain or use the Content-Type associated with the location provided in the request.

Definition at line 321 of file response.c.

References `build_stamp()`, `build_version()`, `con_print()`, `magma_t::http`, `connection_t::http`, `HTTP_COMPLETE`, `HTTP_CONNECTION_NEUTRAL`, `HTTP_RESPOND`, `http_response_allow_cross()`, `http_response_connection()`, `magma`, `MANAGEDBUF`, `mm_wipe()`, `st_char_get()`, `st_cleanup()`, `st_length_int()`, and `time_print_gmt()`.

Referenced by `http_response()`.

chr_t* http_response_status (int_t status)

Get a descriptive string for a numerical http status code.

Parameters:

status the value of the http status code to be looked up.

Returns:

NULL on failure, or a pointer to a null-terminated string containing a description of the specified http status code on success.

Definition at line 20 of file response.c.

References `log_pedantic`.

Referenced by `http_response_header()`.

magma/servers/imap/commands.c File Reference

The functions involved with parsing and routing POP commands.

```
#include "magma.h"
#include "commands.h"
```

Functions

- **int_t imap_compare** (const void *compare, const void *command)
- *Compare the names of two commands.* void **imap_sort** (void)
- *Sort the IMAP command table to be ready for binary searches.* void **imap_requeue** (connection_t *con)
- *Requeue an imap connection for processing, or log it out if there was an error or excess of protocol violations.* void **imap_process** (connection_t *con)

Perform client command processing on an established imap session.

Detailed Description

The functions involved with parsing and routing POP commands.

Definition in file **commands.c**.

Function Documentation

int_t imap_compare (const void * *compare*, const void * *command*)

Compare the names of two commands.

commands.c

Note:

This is an internal function used to sort imap commands and search for them.

Parameters:

compare a pointer to the first command to be compared.

command a pointer to the second command to be compared.

Returns:

-1 if compare < command, 1 if command < compare, or 0 if the two commands are equal.

Definition at line 23 of file commands.c.

References command_t::length, PLACER, st_cmp_ci_eq(), st_cmp_ci_starts(), and command_t::string.

Referenced by imap_process(), and imap_sort().

void imap_process (connection_t * *con*)

Perform client command processing on an established imap session.

Note:

This function will read the next line of user input, parse the command, and then attempt to execute it with the appropriate handler.

Parameters:

con a pointer to the connection object underlying the imap session.

Returns:

This function returns no value.

Definition at line 66 of file commands.c.

References `connection_t::command`, `command`, `con_print()`, `con_read_line()`, `con_write_bl()`, `server_t::cutoff`, `enqueue()`, `command_t::function`, `connection_t::imap`, `imap_command_parser()`, `imap_commands`, `imap_compare()`, `imap_invalid()`, `imap_logout()`, `imap_process()`, `imap_requeue()`, `command_t::length`, `connection_t::line`, `connection_t::network`, `pl_empty()`, `connection_t::protocol`, `requeue()`, `connection_t::server`, `connection_t::spins`, `st_char_get()`, `st_length_get()`, `st_length_int()`, `command_t::string`, `server_t::violations`, and `connection_t::violations`.

Referenced by `imap_process()`, and `imap_requeue()`.

void imap_requeue (connection_t * con)

Requeue an imap connection for processing, or log it out if there was an error or excess of protocol violations.

Parameters:

con a pointer to the connection object of the imap session.

Returns:

This function returns no value.

Definition at line 48 of file commands.c.

References `con_status()`, `server_t::cutoff`, `enqueue()`, `imap_logout()`, `imap_process()`, `connection_t::protocol`, `connection_t::server`, `status`, `server_t::violations`, and `connection_t::violations`.

Referenced by `imap_init()`, and `imap_process()`.

void imap_sort (void)

Sort the IMAP command table to be ready for binary searches.

Returns:

This function returns no value.

Definition at line 38 of file commands.c.

References `imap_commands`, and `imap_compare()`.

Referenced by `protocol_init()`.

magma/servers/molten/commands.c File Reference

Functions used to parse protocol specific data out of the inbound network stream.

```
#include "magma.h"
#include "commands.h"
```

Functions

- **int_t molten_compare** (const void *compare, const void *command)
 - *commands.c* void **molten_sort** (void)
 - *Sort the Molten command table to be ready for binary searches.* void **molten_parse** (connection_t *con)
-

Detailed Description

Functions used to parse protocol specific data out of the inbound network stream.

Definition in file **commands.c**.

Function Documentation

int_t molten_compare (const void * *compare*, const void * *command*)

commands.c

Definition at line 17 of file commands.c.

References command_t::length, PLACER, st_cmp_ci_eq(), st_cmp_ci_starts(), and command_t::string.

Referenced by molten_parse(), and molten_sort().

void molten_parse (connection_t * *con*)

Definition at line 37 of file commands.c.

References connection_t::command, command, con_read_line(), enqueue(), command_t::function, command_t::length, connection_t::line, molten_commands, molten_compare(), molten_invalid(), molten_parse(), molten_quit(), connection_t::network, pl_char_get(), pl_empty(), pl_length_get(), and command_t::string.

Referenced by molten_init(), molten_invalid(), molten_parse(), and molten_stats().

void molten_sort (void)

Sort the Molten command table to be ready for binary searches.

Returns:

This function returns no value.

Definition at line 32 of file commands.c.

References molten_commands, and molten_compare().
Referenced by protocol_init().

magma/servers/pop/commands.c File Reference

Functions involved with parsing and dispatching POP commands.

```
#include "magma.h"
#include "commands.h"
```

Functions

- **int_t pop_compare** (const void *compare, const void *command)
 - *commands.c* void **pop_sort** (void)
 - *Sort the POP3 command table to be ready for binary searches.* void **pop_requeue** (connection_t *con)
 - void **pop_process** (connection_t *con)
-

Detailed Description

Functions involved with parsing and dispatching POP commands.

Definition in file **commands.c**.

Function Documentation

int_t pop_compare (const void * *compare*, const void * *command*)

commands.c

Definition at line 16 of file commands.c.

References command_t::length, PLACER, st_cmp_ci_eq(), st_cmp_ci_starts(), and command_t::string.

Referenced by pop_process(), and pop_sort().

void pop_process (connection_t * *con*)

Definition at line 48 of file commands.c.

References connection_t::command, command, con_read_line(), server_t::cutoff, enqueue(), command_t::function, command_t::length, connection_t::line, connection_t::network, pl_char_get(), pl_empty(), pl_length_get(), connection_t::pop, pop_commands, pop_compare(), pop_invalid(), pop_process(), pop_quit(), pop_requeue(), connection_t::protocol, requeue(), connection_t::server, connection_t::spins, command_t::string, server_t::violations, and connection_t::violations.

Referenced by pop_process(), and pop_requeue().

void pop_requeue (connection_t * *con*)

Definition at line 36 of file commands.c.

References con_status(), server_t::cutoff, enqueue(), pop_process(), pop_quit(), connection_t::protocol, connection_t::server, status, server_t::violations, and connection_t::violations.

Referenced by pop_init(), and pop_process().

void pop_sort (void)

Sort the POP3 command table to be ready for binary searches.

Returns:

This function returns no value.

Definition at line 31 of file commands.c.

References pop_commands, and pop_compare().

Referenced by protocol_init().

magma/servers/smtp/commands.c File Reference

The functions involved with parsing and routing SMTP commands.

```
#include "magma.h"
#include "commands.h"
```

Functions

- **int_t smtp_compare** (const void *compare, const void *command)
- void **smtp_sort** (void)
- *Sort the SMTP command table to be ready for binary searches.* void **smtp_requeue** (connection_t *con)
- *commands.c* void **smtp_process** (connection_t *con)

The main entry point in the smtp server for processing commands issued by clients.

Detailed Description

The functions involved with parsing and routing SMTP commands.

Definition in file **commands.c**.

Function Documentation

int_t smtp_compare (const void * compare, const void * command)

Definition at line 16 of file commands.c.

References `command_t::length`, `PLACER`, `st_cmp_ci_eq()`, `st_cmp_ci_starts()`, and `command_t::string`.

Referenced by `smtp_process()`, and `smtp_sort()`.

void smtp_process (connection_t * con)

The main entry point in the smtp server for processing commands issued by clients.

Parameters:

con a pointer to the connection object of the client issuing the smtp command.

Returns:

This function returns no value.

Definition at line 53 of file commands.c.

References `connection_t::command`, `command`, `con_read_line()`, `server_t::cutoff`, `enqueue()`, `command_t::function`, `command_t::length`, `connection_t::line`, `connection_t::network`, `pl_char_get()`, `pl_empty()`, `pl_length_get()`, `connection_t::protocol`, `requeue()`, `connection_t::server`, `smtp_commands`, `smtp_compare()`, `smtp_data()`, `smtp_invalid()`, `smtp_process()`, `smtp_quit()`, `smtp_requeue()`, `connection_t::spins`, `command_t::string`, `server_t::violations`, and `connection_t::violations`.

Referenced by `smtp_process()`, and `smtp_requeue()`.

void smtp_requeue (connection_t * con)

commands.c

Definition at line 36 of file commands.c.

References `con_status()`, `server_t::cutoff`, `enqueue()`, `connection_t::protocol`, `connection_t::server`, `smtp_process()`, `smtp_quit()`, `status`, `server_t::violations`, and `connection_t::violations`.

Referenced by `smtp_data()`, `smtp_init()`, and `smtp_process()`.

void smtp_sort (void)

Sort the SMTP command table to be ready for binary searches.

Returns:

This function returns no value.

Definition at line 31 of file commands.c.

References `smtp_commands`, and `smtp_compare()`.

Referenced by `protocol_init()`.

magma/servers/imap/fetch.c File Reference

Functions used to handle IMAP commands/actions.

```
#include "magma.h"
```

Functions

- **int_t** **imap_valid_sequence** (**stringer_t** *range)
- void **imap_fetch_free_items** (**imap_fetch_dataitems_t** *items)
- **imap_fetch_dataitems_t** * **imap_parse_dataitems** (**imap_arguments_t** *arguments)
- **stringer_t** * **imap_fetch_envelope** (**stringer_t** *header)
- **stringer_t** * **imap_fetch_bodystructure** (**mail_mime_t** *mime)
- **stringer_t** * **imap_fetch_body_header** (**placer_t** header, **imap_arguments_t** *array, **int_t** not)
- **stringer_t** * **imap_fetch_body_mime** (**placer_t** header)
- **stringer_t** * **imap_fetch_body_tag** (**stringer_t** *tag, **array_t** *items)
- **placer_t** **imap_fetch_body_portion** (**stringer_t** *part)
- **mail_mime_t** * **imap_fetch_body_part** (**mail_message_t** *message, **placer_t** portion)
- **int_t** **imap_fetch_parse_partial** (**stringer_t** *partial, **size_t** *start, **size_t** *length)
- **stringer_t** * **imap_fetch_return_header** (**connection_t** *con, **meta_message_t** *meta, **mail_message_t** **message, **stringer_t** **header, **imap_fetch_response_t** *output)
- **stringer_t** * **imap_fetch_return_text** (**connection_t** *con, **meta_message_t** *meta, **mail_message_t** **message, **stringer_t** **header, **imap_fetch_response_t** *output)
- **mail_message_t** * **imap_fetch_return_message** (**connection_t** *con, **meta_message_t** *meta, **mail_message_t** **message, **stringer_t** **header, **imap_fetch_response_t** *output)
- **mail_mime_t** * **imap_fetch_return_mime** (**connection_t** *con, **meta_message_t** *meta, **mail_message_t** **message, **stringer_t** **header, **imap_fetch_response_t** *output)
- **imap_fetch_response_t** * **imap_fetch_body** (**array_t** *outer, **array_t** *partial, **connection_t** *con, **meta_message_t** *meta, **mail_message_t** **message, **stringer_t** **header, **imap_fetch_response_t** *output)
- **imap_fetch_response_t** * **imap_fetch_message** (**connection_t** *con, **meta_message_t** *meta, **imap_fetch_dataitems_t** *items)
- **inx_t** * **imap_duplicate_messages** (**inx_t** *messages)
- *Create a copy of a collection of messages.* **inx_t** * **imap_narrow_messages** (**inx_t** *messages, **uint64_t** selected, **stringer_t** *range, **int_t** uid)

Detailed Description

Functions used to handle IMAP commands/actions.

Definition in file **fetch.c**.

Function Documentation

inx_t* **imap_duplicate_messages** (**inx_t** * *messages*)

Create a copy of a collection of messages.

fetch.c

See also:

meta_message_dupe()

Parameters:

messages a collection of meta message objects to be duplicated.

Returns:

the copied collection of meta messages on success, or NULL on failure.

Definition at line 1245 of file fetch.c.

References `inx_alloc()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_free()`, `inx_insert()`, `log_error`, `M_INX_LINKED`, `M_TYPE_UINT64`, `meta_message_t::messagenum`, `meta_message_dupe()`, `meta_message_free()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `imap_fetch()`, and `imap_store()`.

imap_fetch_response_t* imap_fetch_body (array_t * *outer*, array_t * *partial*, connection_t * *con*, meta_message_t * *meta*, mail_message_t ** *message*, stringer_t ** *header*, imap_fetch_response_t * *output*)

Definition at line 767 of file fetch.c.

References `ar_field_ar()`, `ar_field_type()`, `ar_length_get()`, `ARRAY_TYPE_ARRAY`, `mail_mime_t::body`, `mail_mime_t::header`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_fetch_body_header()`, `imap_fetch_body_mime()`, `imap_fetch_body_part()`, `imap_fetch_body_portion()`, `imap_fetch_body_tag()`, `imap_fetch_parse_partial()`, `imap_fetch_response_add()`, `imap_fetch_response_free()`, `imap_fetch_return_header()`, `imap_fetch_return_message()`, `imap_fetch_return_mime()`, `imap_fetch_return_text()`, `imap_get_ar_ar()`, `imap_get_ptr()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `length`, `mail_destroy()`, `mail_destroy_header()`, `mail_message_t::mime`, `number`, `pl_data_get()`, `pl_empty()`, `pl_init()`, `pl_length_get()`, `pl_null()`, `PLACER`, `placer_t`, `st_char_get()`, `st_cleanup()`, `st_cmp_ci_eq()`, `st_free()`, `st_import()`, `st_length_get()`, and `st_merge`.

Referenced by `imap_fetch_message()`.

stringer_t* imap_fetch_body_header (placer_t *header*, imap_arguments_t * *array*, int_t *not*)

Definition at line 391 of file fetch.c.

References `ar_length_get()`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_get_st_ar()`, `imap_get_type_ar()`, `mail_header_pop()`, `mm_cmp_ci_eq()`, `number`, `pl_char_get()`, `pl_data_get()`, `pl_empty()`, `placer_t`, `st_char_get()`, `st_cleanup()`, `st_length_get()`, and `st_merge`.

Referenced by `imap_fetch_body()`.

stringer_t* imap_fetch_body_mime (placer_t *header*)

Definition at line 442 of file fetch.c.

References `mail_header_fetch_all()`, `PLACER`, `st_cleanup()`, and `st_merge`.

Referenced by `imap_fetch_body()`.

mail_mime_t* imap_fetch_body_part (mail_message_t * *message*, placer_t *portion*)

Definition at line 560 of file fetch.c.

References `ar_field_ptr()`, `mail_mime_t::children`, `mail_message_t::mime`, `number`, `placer_t`, `tok_get_count_st()`, `tok_get_st()`, and `uint32_conv_st()`.

Referenced by `imap_fetch_body()`.

`placer_t imap_fetch_body_portion (stringer_t * part)`

Definition at line 517 of file `fetch.c`.

References `length`, `pl_init()`, `pl_null()`, `st_char_get()`, and `st_empty_out()`.

Referenced by `imap_fetch_body()`.

`stringer_t* imap_fetch_body_tag (stringer_t * tag, array_t * items)`

Definition at line 469 of file `fetch.c`.

References `ar_length_get()`, `imap_get_st_ar()`, `number`, `st_cleanup()`, `st_free()`, `st_import()`, `st_merge`, and `upper_st()`.

Referenced by `imap_fetch_body()`.

`stringer_t* imap_fetch_bodystructure (mail_mime_t * mime)`

Definition at line 234 of file `fetch.c`.

References `ar_field_ptr()`, `ar_field_st()`, `ar_free()`, `ar_length_get()`, `mail_mime_t::body`, `mail_mime_t::children`, `mail_mime_t::header`, `imap_build_array()`, `imap_build_array_isliteral()`, `imap_fetch_bodystructure()`, `items`, `length`, `mail_header_fetch_cleaned()`, `mail_mime_content_encoding()`, `mail_mime_content_id()`, `mail_mime_type_group()`, `mail_mime_type_parameters()`, `mail_mime_type_sub()`, `ns_length_get()`, `pl_init()`, `PLACER`, `st_char_get()`, `st_cleanup()`, `st_cmp_cs_eq()`, `st_data_get()`, `st_free()`, `st_import()`, `st_length_get()`, `st_merge`, and `upper_st()`.

Referenced by `imap_fetch_bodystructure()`, and `imap_fetch_message()`.

`stringer_t* imap_fetch_envelope (stringer_t * header)`

Definition at line 189 of file `fetch.c`.

References `imap_build_array()`, `imap_parse_address()`, `mail_header_fetch_cleaned()`, `pl_init()`, `pl_null()`, `PLACER`, `placer_t`, `st_char_get()`, `st_cleanup()`, and `st_length_get()`.

Referenced by `imap_fetch_message()`.

`void imap_fetch_free_items (imap_fetch_dataitems_t * items)`

Definition at line 37 of file `fetch.c`.

References `mm_cleanup()`, `mm_free()`, `imap_fetch_dataitems_t::normal`, `imap_fetch_dataitems_t::normal_partial`, `imap_fetch_dataitems_t::peek`, and `imap_fetch_dataitems_t::peek_partial`.

Referenced by `imap_fetch()`, and `imap_parse_dataitems()`.

`imap_fetch_response_t* imap_fetch_message (connection_t * con, meta_message_t * meta, imap_fetch_dataitems_t * items)`

Definition at line 1048 of file `fetch.c`.

References `imap_fetch_dataitems_t::body`, `mail_mime_t::body`, `imap_fetch_dataitems_t::bodystructure`, `CONTIGUOUS`, `meta_message_t::created`, `imap_fetch_dataitems_t::envelope`, `imap_fetch_dataitems_t::flags`, `HEAP`, `connection_t::imap`, `imap_fetch_body()`, `imap_fetch_bodystructure()`, `imap_fetch_envelope()`, `imap_fetch_response_add()`, `imap_fetch_response_free()`, `imap_fetch_return_header()`, `imap_fetch_dataitems_t::internaldate`, `log_pedantic`, `mail_destroy()`, `mail_destroy_header()`, `mail_load_message()`, `mail_mime_update()`, `MAIL_STATUS_ANSWERED`, `MAIL_STATUS_DELETED`, `MAIL_STATUS_DRAFT`, `MAIL_STATUS_FLAGGED`, `MAIL_STATUS_RECENT`, `MAIL_STATUS_SEEN`, `MANAGED_T`, `meta_message_t::messagenum`, `mail_message_t::mime`, `imap_fetch_dataitems_t::normal`, `imap_fetch_dataitems_t::normal_partial`, `ns_length_get()`, `imap_fetch_dataitems_t::peek`, `imap_fetch_dataitems_t::peek_partial`, `PLACER`, `imap_fetch_dataitems_t::rfc822`, `imap_fetch_dataitems_t::rfc822_header`, `imap_fetch_dataitems_t::rfc822_size`, `imap_fetch_dataitems_t::rfc822_text`, `connection_t::server`, `meta_message_t::size`, `st_aprint_opts()`, `st_import()`, `st_length_get()`, `st_merge`, `meta_message_t::status`, `mail_message_t::text`, `imap_fetch_dataitems_t::uid`, and `meta_message_t::updated`.

Referenced by `imap_fetch()`.

`int_t imap_fetch_parse_partial (stringer_t * partial, size_t * start, size_t * length)`

Definition at line 614 of file `fetch.c`.

References `int32_conv_bl()`, `number`, `st_char_get()`, and `st_length_get()`.

Referenced by `imap_fetch_body()`.

`stringer_t* imap_fetch_return_header (connection_t * con, meta_message_t * meta, mail_message_t ** message, stringer_t ** header, imap_fetch_response_t * output)`

Definition at line 690 of file `fetch.c`.

References `connection_t::imap`, `imap_fetch_response_free()`, `mail_destroy()`, `mail_load_header()`, `st_char_get()`, and `st_import()`.

Referenced by `imap_fetch_body()`, and `imap_fetch_message()`.

`mail_message_t* imap_fetch_return_message (connection_t * con, meta_message_t * meta, mail_message_t ** message, stringer_t ** header, imap_fetch_response_t * output)`

Definition at line 733 of file `fetch.c`.

References `connection_t::imap`, `imap_fetch_response_free()`, `mail_destroy()`, `mail_destroy_header()`, `mail_load_message()`, `mail_mime_update()`, and `connection_t::server`.

Referenced by `imap_fetch_body()`.

`mail_mime_t* imap_fetch_return_mime (connection_t * con, meta_message_t * meta, mail_message_t ** message, stringer_t ** header, imap_fetch_response_t * output)`

Definition at line 747 of file `fetch.c`.

References `connection_t::imap`, `imap_fetch_response_free()`, `mail_destroy()`, `mail_destroy_header()`, `mail_load_message()`, `mail_mime_update()`, and `connection_t::server`.

Referenced by `imap_fetch_body()`.

**stringer_t* imap_fetch_return_text (connection_t * con, meta_message_t * meta,
mail_message_t ** message, stringer_t ** header, imap_fetch_response_t * output)**

Definition at line 719 of file fetch.c.

References connection_t::imap, imap_fetch_response_free(), mail_destroy(), mail_destroy_header(), mail_load_message(), and connection_t::server.

Referenced by imap_fetch_body().

**inx_t* imap_narrow_messages (inx_t * messages, uint64_t selected, stringer_t * range,
int_t uid)**

Definition at line 1284 of file fetch.c.

References meta_message_t::foldernum, inx_alloc(), inx_cleanup(), inx_count(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_free(), inx_insert(), log_error, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, meta_message_t::messagenum, number, pl_char_get(), pl_empty(), pl_null(), placer_t, meta_message_t::sequencenum, st_char_get(), st_length_int(), tok_get_count_st(), tok_get_st(), multi_t::u64, uint64_conv_st(), and multi_t::val.

Referenced by imap_copy(), imap_fetch(), and imap_store().

imap_fetch_dataitems_t* imap_parse_dataitems (imap_arguments_t * arguments)

Definition at line 49 of file fetch.c.

References ar_append(), ar_length_get(), ARRAY_TYPE_ARRAY, ARRAY_TYPE_POINTER, imap_fetch_dataitems_t::body, imap_fetch_dataitems_t::bodystructure, imap_fetch_dataitems_t::envelope, imap_fetch_dataitems_t::flags, IMAP_ARGUMENT_TYPE_ARRAY, imap_fetch_free_items(), imap_get_ar_ar(), imap_get_st_ar(), imap_get_type_ar(), imap_fetch_dataitems_t::internaldate, log_error, mm_alloc(), imap_fetch_dataitems_t::normal, imap_fetch_dataitems_t::normal_partial, number, imap_fetch_dataitems_t::peek, imap_fetch_dataitems_t::peek_partial, PLACER, imap_fetch_dataitems_t::rfc822, imap_fetch_dataitems_t::rfc822_header, imap_fetch_dataitems_t::rfc822_size, imap_fetch_dataitems_t::rfc822_text, st_cmp_ci_eq(), st_cmp_cs_starts(), type(), and imap_fetch_dataitems_t::uid.

Referenced by imap_fetch().

int_t imap_valid_sequence (stringer_t * range)

Definition at line 16 of file fetch.c.

References length, and st_empty_out().

Referenced by imap_copy(), imap_fetch(), imap_search_messages_inner(), and imap_store().

magma/servers/imap/fetch_response.c File Reference

Functions used to handle IMAP commands/actions.

```
#include "magma.h"
```

Functions

- void **imap_fetch_response_free** (imap_fetch_response_t *response)
- **imap_fetch_response_t** * **imap_fetch_response_add** (imap_fetch_response_t *response, stringer_t *key, stringer_t *value)

fetch_response.c

Detailed Description

Functions used to handle IMAP commands/actions.

Definition in file **fetch_response.c**.

Function Documentation

imap_fetch_response_t* **imap_fetch_response_add** (imap_fetch_response_t * *response*, stringer_t * *key*, stringer_t * *value*)

fetch_response.c

Definition at line 30 of file fetch_response.c.

References `CONTIGUOUS`, `HEAP`, `imap_fetch_response_t::key`, `log_error`, `MANAGED_T`, `mm_alloc()`, `mm_free()`, `imap_fetch_response_t::next`, `st_dupe_opts()`, `st_free()`, and `imap_fetch_response_t::value`.

Referenced by `imap_fetch_body()`, and `imap_fetch_message()`.

void **imap_fetch_response_free** (imap_fetch_response_t * *response*)

Definition at line 15 of file fetch_response.c.

References `imap_fetch_response_t::key`, `mm_free()`, `imap_fetch_response_t::next`, `st_cleanup()`, and `imap_fetch_response_t::value`.

Referenced by `imap_fetch()`, `imap_fetch_body()`, `imap_fetch_message()`, `imap_fetch_return_header()`, `imap_fetch_return_message()`, `imap_fetch_return_mime()`, and `imap_fetch_return_text()`.

magma/servers/imap/flags.c File Reference

Functions to handle IMAP commands/actions.

```
#include "magma.h"
```

Functions

- `uint32_t imap_get_flag (stringer_t *string)`
- `int_t imap_flag_action (stringer_t *string)`
- *flags.c* `uint32_t imap_flag_parse (void *ptr, int_t type)`
- `void imap_update_flags (meta_user_t *user, inx_t *messages, uint64_t foldernum, int_t action, uint32_t flags)`

Detailed Description

Functions to handle IMAP commands/actions.

Definition in file **flags.c**.

Function Documentation

`int_t imap_flag_action (stringer_t * string)`

flags.c

Definition at line 46 of file flags.c.

References `IMAP_FLAG_ADD`, `IMAP_FLAG_REMOVE`, `IMAP_FLAG_REPLACE`, `IMAP_FLAG_SILENT`, `PLACER`, and `st_cmp_ci_eq()`.

Referenced by `imap_store()`.

`uint32_t imap_flag_parse (void * ptr, int_t type)`

Definition at line 70 of file flags.c.

References `ar_length_get()`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_get_flag()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `MAIL_STATUS_EMPTY`, and `number`.

Referenced by `imap_append()`, and `imap_store()`.

`uint32_t imap_get_flag (stringer_t * string)`

Definition at line 15 of file flags.c.

References `MAIL_STATUS_ANSWERED`, `MAIL_STATUS_DELETED`, `MAIL_STATUS_DRAFT`, `MAIL_STATUS_FLAGGED`, `MAIL_STATUS_RECENT`, `MAIL_STATUS_SEEN`, `PLACER`, and `st_cmp_ci_eq()`.

Referenced by `imap_flag_parse()`.

void imap_update_flags (meta_user_t * *user*, inx_t * *messages*, uint64_t *foldernum*, int_t *action*, uint32_t *flags*)

Definition at line 117 of file flags.c.

References meta_message_t::foldernum, IMAP_FLAG_ADD, IMAP_FLAG_REMOVE, IMAP_FLAG_REPLACE, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), log_error, MAIL_STATUS_ANSWERED, MAIL_STATUS_DELETED, MAIL_STATUS_DRAFT, MAIL_STATUS_EMPTY, MAIL_STATUS_FLAGGED, MAIL_STATUS_RECENT, MAIL_STATUS_SEEN, meta_data_flags_add(), meta_data_flags_remove(), meta_data_flags_replace(), meta_message_t::status, and meta_user_t::usernum.

Referenced by imap_store().

magma/web/portal/flags.c File Reference

Functions for handling message flags and tags.

```
#include "magma.h"
```

Functions

- `json_t * portal_message_flags_array (meta_message_t *meta)`
- *Return a json array of flag descriptions for a mail message's flags bitmask.* `int_t portal_parse_flags (json_t *array, uint32_t *flags)`
- `json_t * portal_message_tags_array (meta_message_t *meta)`

Detailed Description

Functions for handling message flags and tags.

Definition in file **flags.c**.

Function Documentation

`json_t* portal_message_flags_array (meta_message_t * meta)`

Return a json array of flag descriptions for a mail message's flags bitmask.

flags.c

TODO: The messages.load method uses the flags/tags helper functions, but the messages.list and the messages.tags/flags methods still need to be updated.

Parameters:

meta a pointer to the meta message object of the mail message to have its flags parsed.

Returns:

NULL on failure, or a pointer to the json object of the specified mail message's flags.

Definition at line 23 of file flags.c.

References `json_array_append_new_d`, `json_array_d`, `json_string_d`, `MAIL_MARK_BLACKHOLED`, `MAIL_MARK_INFECTED`, `MAIL_MARK_JUNK`, `MAIL_MARK_PHISHING`, `MAIL_MARK_SPOOFED`, `MAIL_STATUS_ANSWERED`, `MAIL_STATUS_APPENDED`, `MAIL_STATUS_DELETED`, `MAIL_STATUS_DRAFT`, `MAIL_STATUS_EMPTY`, `MAIL_STATUS_ENCRYPTED`, `MAIL_STATUS_FLAGGED`, `MAIL_STATUS_RECENT`, `MAIL_STATUS_SEEN`, and `meta_message_t::status`.

Referenced by `portal_endpoint_messages_flag()`, `portal_endpoint_messages_list()`, and `portal_message_meta()`.

`json_t* portal_message_tags_array (meta_message_t * meta)`

Definition at line 91 of file flags.c.

References `ar_field_st()`, `ar_length_get()`, `count`, `json_array_append_new_d`, `json_array_d`, `json_string_d`, `st_char_get()`, and `meta_message_t::tags`.

Referenced by portal_message_meta().

int_t portal_parse_flags (json_t * *array*, uint32_t * *flags*)

Definition at line 51 of file flags.c.

References count, json_array_get_d, json_array_size_d, json_string_value_d, MAIL_MARK_BLACKHOLED, MAIL_MARK_INFECTED, MAIL_MARK_JUNK, MAIL_MARK_PHISHING, MAIL_MARK_SPOOFED, MAIL_STATUS_ANSWERED, MAIL_STATUS_APPENDED, MAIL_STATUS_DELETED, MAIL_STATUS_DRAFT, MAIL_STATUS_FLAGGED, MAIL_STATUS_RECENT, MAIL_STATUS_SEEN, NULLER, PLACER, and st_cmp_ci_eq().

Referenced by portal_endpoint_messages_flag().

magma/servers/imap/imap.c File Reference

Functions used to handle IMAP commands/actions.

```
#include "magma.h"
```

Functions

- void **imap_starttls** (**connection_t** *con)
 - *Create a secure connection for an IMAP session.* void **imap_invalid** (**connection_t** *con)
 - *Respond to an invalid imap command from a client.* void **imap_logout** (**connection_t** *con)
 - *Terminate an IMAP session gracefully with a BYE message and destroy the underlying connection.* void **imap_login** (**connection_t** *con)
 - *Attempt to perform a user login on an imap client connection.* void **imap_noop** (**connection_t** *con)
 - void **imap_check** (**connection_t** *con)
 - void **imap_list** (**connection_t** *con)
 - void **imap_lsub** (**connection_t** *con)
 - void **imap_create** (**connection_t** *con)
 - *Create a new imap folder in response to the imap "CREATE" command.* void **imap_delete** (**connection_t** *con)
 - *Handle the imap "DELETE" command and delete the specified imap folder.* void **imap_rename** (**connection_t** *con)
 - void **imap_status** (**connection_t** *con)
 - void **imap_subscribe** (**connection_t** *con)
 - void **imap_unsubscribe** (**connection_t** *con)
 - void **imap_examine** (**connection_t** *con)
 - void **imap_select** (**connection_t** *con)
 - void **imap_store** (**connection_t** *con)
 - void **imap_close** (**connection_t** *con)
 - void **imap_expunge** (**connection_t** *con)
 - void **imap_copy** (**connection_t** *con)
 - void **imap_append** (**connection_t** *con)
 - *imap.c* void **imap_fetch** (**connection_t** *con)
 - void **imap_search** (**connection_t** *con)
 - void **imap_idle** (**connection_t** *con)
 - void **imap_id** (**connection_t** *con)
 - void **imap_capability** (**connection_t** *con)
 - *Display the capability string for the IMAP server.* void **imap_init** (**connection_t** *con)
- The main imap entry point for all inbound client connections, as dispatched by the generic protocol handler (display banner greeting).*
-

Detailed Description

Functions used to handle IMAP commands/actions.

Definition in file **imap.c**.

Function Documentation

void **imap_append** (**connection_t** * con)

imap.c

BUG: If the user is over their quota check whether rollout is enabled and if so. If enabled make room for the appended message using the rollout logic.

Definition at line 1400 of file imap.c.

References `ar_length_get()`, `con_print()`, `meta_message_t::foldernum`, `meta_folder_t::foldernum`, `connection_t::imap`, `imap_append_message()`, `IMAP_ARGUMENT_TYPE_ARRAY`, `IMAP_ARGUMENT_TYPE_LITERAL`, `imap_flag_parse()`, `imap_get_ptr()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `log_pedantic`, `MAIL_STATUS_EMPTY`, `MAIL_STATUS_RECENT`, `meta_folders_by_name()`, `META_USER_OVERQUOTA`, `meta_user_unlock()`, `meta_user_wlock()`, `st_char_get()`, `st_length_int()`, and `meta_message_t::status`.

void imap_capability (connection_t * con)

Display the capability string for the IMAP server.

Note:

This string always needs to stay in sync with the banner greeting.

Returns:

This function returns no value.

Definition at line 1764 of file imap.c.

References `con_print()`, `con_secure()`, `connection_t::imap`, `st_char_get()`, and `st_length_int()`.

void imap_check (connection_t * con)

Definition at line 237 of file imap.c.

References `con_print()`, `connection_t::imap`, `imap_session_update()`, `st_char_get()`, and `st_length_int()`.

void imap_close (connection_t * con)

Definition at line 1025 of file imap.c.

References `ar_length_get()`, `con_print()`, `meta_message_t::foldernum`, `connection_t::imap`, `imap_message_expunge()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_DELETED`, `MAIL_STATUS_RECENT`, `meta_messages_update_sequences()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_MESSAGES`, `serial_get()`, `serial_increment()`, `st_char_get()`, `st_length_int()`, `meta_message_t::status`, `user_lock()`, and `user_unlock()`.

void imap_copy (connection_t * con)

Definition at line 1234 of file imap.c.

References `ar_length_get()`, `con_print()`, `count`, `meta_message_t::foldernum`, `meta_folder_t::foldernum`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_get_st_ar()`, `imap_get_type_ar()`, `imap_message_copier()`, `imap_narrow_messages()`, `imap_range_build()`, `imap_valid_sequence()`, `inx_count()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_free()`, `log_pedantic`,

MAIL_STATUS_RECENT, meta_message_t::messagenum, meta_folders_by_name(), meta_user_unlock(), meta_user_wlock(), mm_alloc(), mm_free(), meta_message_t::sequencenum, st_char_get(), st_length_get(), st_length_int(), meta_message_t::status, user_lock(), and user_unlock().

void imap_create (connection_t * con)

Create a new imap folder in response to the imap "CREATE" command.

See also:

imap_folder_create()

Parameters:

con the connection across which the folder creation request was made.

Returns:

This function returns no value.

Definition at line 412 of file imap.c.

References ar_length_get(), con_print(), FOLDER_LENGTH_LIMIT, connection_t::imap, IMAP_ARGUMENT_TYPE_ARRAY, imap_folder_create(), IMAP_FOLDER_RECURSION_LIMIT, imap_get_st_ar(), imap_get_type_ar(), meta_user_unlock(), meta_user_wlock(), OBJECT_FOLDERS, serial_get(), serial_increment(), st_char_get(), and st_length_int().

void imap_delete (connection_t * con)

Handle the imap "DELETE" command and delete the specified imap folder.

See also:

imap_folder_remove()

Parameters:

con a pointer to the connection object of the imap session generating the delete request.

Returns:

This function returns no value.

Definition at line 476 of file imap.c.

References ar_length_get(), con_print(), connection_t::imap, IMAP_ARGUMENT_TYPE_ARRAY, imap_folder_remove(), imap_get_st_ar(), imap_get_type_ar(), meta_user_unlock(), meta_user_wlock(), OBJECT_FOLDERS, serial_get(), serial_increment(), st_char_get(), and st_length_int().

void imap_examine (connection_t * con)

Definition at line 740 of file imap.c.

References ar_length_get(), con_print(), imap_folder_status_t::first, imap_folder_status_t::foldernum, meta_message_t::foldernum, connection_t::imap, IMAP_ARGUMENT_TYPE_ARRAY, imap_folder_status(), imap_get_st_ar(), imap_get_type_ar(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), MAIL_STATUS_RECENT, imap_folder_status_t::messages, meta_user_rlock(), meta_user_unlock(), meta_user_wlock(), imap_folder_status_t::recent, st_char_get(), st_length_int(), meta_message_t::status, status, and imap_folder_status_t::uidnext.

void imap_expunge (connection_t * con)

Definition at line 1135 of file imap.c.

References `ar_length_get()`, `con_print()`, `meta_message_t::foldernum`, `connection_t::imap`, `imap_message_expunge()`, `imap_session_update()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_DELETED`, `meta_messages_update_sequences()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_MESSAGES`, `meta_message_t::sequencenum`, `serial_get()`, `serial_increment()`, `st_char_get()`, `st_length_int()`, `meta_message_t::status`, `user_lock()`, and `user_unlock()`.

void imap_fetch (connection_t * con)

Definition at line 1485 of file imap.c.

References `ar_length_get()`, `con_print()`, `con_status()`, `con_write_bl()`, `con_write_st()`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_duplicate_messages()`, `imap_fetch_free_items()`, `imap_fetch_message()`, `imap_fetch_response_free()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `imap_narrow_messages()`, `imap_parse_dataitems()`, `imap_valid_sequence()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_reset()`, `inx_cursor_value_next()`, `inx_free()`, `items`, `imap_fetch_response_t::key`, `MAIL_STATUS_SEEN`, `meta_data_flags_add()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_user_wlock()`, `imap_fetch_response_t::next`, `imap_fetch_dataitems_t::normal`, `OBJECT_MESSAGES`, `PLACER`, `imap_fetch_dataitems_t::rfc822`, `imap_fetch_dataitems_t::rfc822_header`, `imap_fetch_dataitems_t::rfc822_text`, `meta_message_t::sequencenum`, `serial_get()`, `serial_increment()`, `st_char_get()`, `st_cmp_cs_eq()`, `st_length_int()`, `status`, `meta_message_t::status`, `meta_message_t::updated`, and `imap_fetch_response_t::value`.

void imap_id (connection_t * con)

Definition at line 1718 of file imap.c.

References `ar_length_get()`, `build_stamp()`, `build_version()`, `con_addr_presentation()`, `con_print()`, `count`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_get_ar_ar()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `log_info`, `MANAGEDBUF`, `st_append_opts()`, `st_char_get()`, `st_free()`, `st_length_get()`, `st_length_int()`, `st_sprint()`, and `time_print_local()`.

void imap_idle (connection_t * con)

Definition at line 1701 of file imap.c.

References `con_print()`, `connection_t::imap`, `st_char_get()`, and `st_length_int()`.

void imap_init (connection_t * con)

The main imap entry point for all inbound client connections, as dispatched by the generic protocol handler (display banner greeting).

Parameters:

con a pointer to the connection object of the newly connected client.

Returns:

This function returns no value.

Definition at line 1783 of file imap.c.

References `build_version()`, `con_print()`, `con_reverse_enqueue()`, `con_secure()`, `server_t::domain`, `imap_requeue()`, `connection_t::server`, `st_char_get()`, `st_length_get()`, and `st_length_int()`.

Referenced by `protocol_enqueue()`.

void imap_invalid (connection_t * con)

Respond to an invalid imap command from a client.

Parameters:

con a pointer to the client connection that issued the bad command.

Returns:

This function returns no value.

Definition at line 62 of file `imap.c`.

References `con_print()`, `server_t::delay`, `connection_t::imap`, `connection_t::protocol`, `connection_t::server`, `st_char_get()`, `st_length_int()`, `server_t::violations`, and `connection_t::violations`.

Referenced by `imap_process()`, and `imap_starttls()`.

void imap_list (connection_t * con)

Definition at line 252 of file `imap.c`.

References `ar_length_get()`, `con_print()`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_folder_name_escaped()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `imap_narrow_folders()`, `inx_count()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_reset()`, `inx_cursor_value_next()`, `inx_free()`, `log_pedantic`, `magma_folder_name()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_folder_t::name`, `NULLER`, `meta_folder_t::parent`, `PLACER`, `st_char_get()`, `st_cmp_ci_eq()`, `st_free()`, and `st_length_int()`.

void imap_login (connection_t * con)

Attempt to perform a user login on an imap client connection.

Parameters:

con a pointer to the connection object of the remote session.

Returns:

This function returns no value.

LOW: `con->imap.bypass == 0` used to be here.

Definition at line 102 of file `imap.c`.

References `ar_length_get()`, `credential_t::auth`, `con_addr_presentation()`, `con_print()`, `con_secure()`, `CONTIGUOUS`, `credential_alloc_auth()`, `credential_free()`, `HEAP`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_get_st_ar()`, `imap_get_type_ar()`, `inx_count()`, `log_pedantic`, `MANAGED_T`, `MANAGEDBUF`, `meta_data_update_log()`, `meta_get()`, `META_GET_FOLDERS`, `META_GET_MESSAGES`, `META_PROT_IMAP`, `meta_remove()`, `meta_user_rlock()`, `META_USER_SSL`, `meta_user_unlock()`, `st_char_get()`, `st_cleanup()`, `st_dupe_opts()`, and `st_length_int()`.

void imap_logout (connection_t * con)

Terminate an IMAP session gracefully with a BYE message and destroy the underlying connection.

Parameters:

con the IMAP connection to be terminated.

Returns:

This function returns no value.

Definition at line 76 of file imap.c.

References `con_destroy()`, `con_print()`, `con_status()`, `con_write_bl()`, `connection_t::imap`, `st_char_get()`, and `st_length_int()`.

Referenced by `imap_process()`, and `imap_requeue()`.

void imap_lsub (connection_t * con)

Definition at line 341 of file imap.c.

References `ar_length_get()`, `con_print()`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_folder_name_escaped()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `imap_narrow_folders()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_reset()`, `inx_cursor_value_next()`, `inx_free()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_folder_t::name`, `NULLER`, `meta_folder_t::parent`, `PLACER`, `st_char_get()`, `st_cmp_ci_eq()`, `st_free()`, and `st_length_int()`.

void imap_noop (connection_t * con)

Definition at line 226 of file imap.c.

References `con_print()`, `connection_t::imap`, `imap_session_update()`, `st_char_get()`, and `st_length_int()`.

void imap_rename (connection_t * con)

Definition at line 528 of file imap.c.

References `ar_length_get()`, `con_print()`, `FOLDER_LENGTH_LIMIT`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `IMAP_FOLDER_RECURSION_LMIIT`, `imap_folder_rename()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_FOLDERS`, `serial_get()`, `serial_increment()`, `st_char_get()`, and `st_length_int()`.

void imap_search (connection_t * con)

Definition at line 1650 of file imap.c.

References `ar_length_get()`, `con_print()`, `con_status()`, `con_write_st()`, `HEAP`, `connection_t::imap`, `imap_search_messages()`, `inx_cursor_alloc()`, `inx_cursor_value_next()`, `inx_free()`, `JOINTED`, `MANAGED_T`, `MANAGEDBUF`, `meta_message_t::messagenum`, `meta_message_t::sequencenum`, `st_append_opts()`, `st_aprint_opts()`, `st_char_get()`, `st_free()`, `st_length_get()`, `st_length_int()`, `st_length_set()`, `st_sprint()`, and `status`.

void imap_select (connection_t * con)

Definition at line 811 of file imap.c.

References `ar_length_get()`, `con_print()`, `imap_folder_status_t::first`, `imap_folder_status_t::foldernum`, `meta_message_t::foldernum`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_folder_status()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_RECENT`, `imap_folder_status_t::messages`, `meta_data_flags_remove()`, `meta_user_unlock()`, `meta_user_wlock()`, `imap_folder_status_t::recent`, `st_char_get()`, `st_length_int()`, `meta_message_t::status`, `status`, and `imap_folder_status_t::uidnext`.

void imap_starttls (connection_t * con)

Create a secure connection for an IMAP session.

TODO: Review error messages and update them with the appropriate response code. LOW: When should we check the serial number to see if the local data is stale and needs to be refreshed?

Note:

RFC 2595 / section 3.1 specifies that STARTTLS is only available in a non-authenticated state.

Parameters:

con the connection on top of which the ssl session will be established.

Returns:

This function returns no value.

Definition at line 24 of file imap.c.

References `connection_t::buffer`, `con_print()`, `con_secure()`, `connection_t::imap`, `imap_invalid()`, `connection_t::line`, `log_pedantic`, `M_SSL_BIO_NOCLOSE`, `connection_t::network`, `pl_null()`, `connection_t::server`, `connection_t::sockd`, `connection_t::ssl`, `ssl_alloc()`, `st_char_get()`, `st_length_get()`, `st_length_set()`, `stats_increment_by_name()`, and `connection_t::status`.

void imap_status (connection_t * con)

Definition at line 594 of file imap.c.

References `ar_length_get()`, `con_print()`, `imap_folder_status_t::foldernum`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_folder_status()`, `imap_get_ar_ar()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `imap_folder_status_t::messages`, `meta_user_rlock()`, `meta_user_unlock()`, `NULLER`, `number`, `PLACER`, `imap_folder_status_t::recent`, `st_append_opts()`, `st_char_get()`, `st_cmp_ci_eq()`, `st_free()`, `st_length_get()`, `st_length_int()`, `status`, `imap_folder_status_t::uidnext`, `imap_folder_status_t::unseen`, and `values`.

void imap_store (connection_t * con)

LOW: Shouldn't we be checking for stale status info so the update doesn't make decisions based on incorrect status data? On the other hand the actual IMAP logic is passed all the way through to the DB so even if the server ends up with incorrect status information, the database should remain accurate.

Definition at line 890 of file imap.c.

References `ar_length_get()`, `con_print()`, `connection_t::imap`, `IMAP_ARGUMENT_TYPE_ARRAY`, `imap_duplicate_messages()`, `imap_flag_action()`, `imap_flag_parse()`, `IMAP_FLAG_SILENT`, `imap_get_ptr()`, `imap_get_st_ar()`, `imap_get_type_ar()`, `imap_narrow_messages()`, `imap_session_update()`, `imap_update_flags()`, `imap_valid_sequence()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_free()`, `MAIL_STATUS_ANSWERED`, `MAIL_STATUS_DELETED`, `MAIL_STATUS_DRAFT`,

MAIL_STATUS_FLAGGED, MAIL_STATUS_RECENT, MAIL_STATUS_SEEN,
meta_message_t::messagenum, meta_user_unlock(), meta_user_wlock(), OBJECT_MESSAGES,
meta_message_t::sequencenum, serial_get(), serial_increment(), st_char_get(), st_length_int(), and
meta_message_t::status.

void imap_subscribe (connection_t * con)

Definition at line 681 of file imap.c.

References ar_length_get(), con_print(), FOLDER_LENGTH_LIMIT, connection_t::imap,
IMAP_ARGUMENT_TYPE_ARRAY, imap_folder_create(), IMAP_FOLDER_RECURSION_LIMIT,
imap_get_st_ar(), imap_get_type_ar(), meta_folders_by_name(), meta_user_unlock(), meta_user_wlock(),
st_char_get(), and st_length_int().

void imap_unsubscribe (connection_t * con)

Definition at line 730 of file imap.c.

References con_print(), connection_t::imap, st_char_get(), and st_length_int().

magma/servers/imap/output.c File Reference

Functions used to handle IMAP commands/actions.

```
#include "magma.h"
```

Functions

- **stringer_t * imap_build_array_isliteral (placer_t data)**
- **stringer_t * imap_build_array (chr_t *format,...)**

output.c

Detailed Description

Functions used to handle IMAP commands/actions.

Definition in file **output.c**.

Function Documentation

stringer_t* imap_build_array (chr_t * *format*, ...)

output.c

Definition at line 42 of file output.c.

References `imap_build_array_isliteral()`, `length`, `log_error`, `log_pedantic`, `ns_length_get()`, `number`, `pl_data_get()`, `pl_empty()`, `pl_init()`, `pl_length_get()`, `placer_t`, `st_alloc`, `st_char_get()`, `st_free()`, `st_length_get()`, and `st_length_set()`.

Referenced by `imap_fetch_bodystructure()`, `imap_fetch_envelope()`, `imap_parse_address()`, and `imap_parse_address_part()`.

stringer_t* imap_build_array_isliteral (placer_t *data*)

Definition at line 15 of file output.c.

References `length`, `pl_data_get()`, `pl_empty()`, `pl_length_get()`, and `st_merge`.

Referenced by `imap_build_array()`, and `imap_fetch_bodystructure()`.

magma/servers/imap/parse_address.c File Reference

Functions used to handle IMAP commands/actions.

```
#include "magma.h"
```

Functions

- void **imap_parse_address_put** (**stringer_t** *buffer, **chr_t** c)
- *LOW: Do we need an imap_parse_address_group() function?* **stringer_t** * **imap_parse_address_part** (**placer_t** input)
- **placer_t** **imap_parse_address_breaker** (**stringer_t** *address, **uint32_t** part)
- **stringer_t** * **imap_parse_address** (**stringer_t** *address)

parse_address.c

Detailed Description

Functions used to handle IMAP commands/actions.

Definition in file **parse_address.c**.

Function Documentation

stringer_t* **imap_parse_address** (**stringer_t** * *address*)

parse_address.c

Definition at line 216 of file **parse_address.c**.

References **imap_build_array()**, **imap_parse_address_breaker()**, **imap_parse_address_part()**, **pl_empty()**, **placer_t**, **st_free()**, **st_merge**, and **imap_fetch_response_t::value**.

Referenced by **imap_fetch_envelope()**.

placer_t **imap_parse_address_breaker** (**stringer_t** * *address*, **uint32_t** *part*)

Definition at line 163 of file **parse_address.c**.

References **length**, **pl_init()**, **pl_null()**, **placer_t**, **st_char_get()**, **st_length_get()**, and **status**.

Referenced by **imap_parse_address()**.

stringer_t* **imap_parse_address_part** (**placer_t** *input*)

Definition at line 32 of file **parse_address.c**.

References **imap_build_array()**, **imap_parse_address_put()**, **length**, **pl_data_get()**, **pl_length_get()**, **placer_t**, **st_alloc**, **st_char_get()**, **st_free()**, **st_length_get()**, **st_length_set()**, and **tok_get_st()**.

Referenced by **imap_parse_address()**.

void **imap_parse_address_put** (**stringer_t** * *buffer*, **chr_t** *c*)

LOW: Do we need an `imap_parse_address_group()` function?

Definition at line 17 of file `parse_address.c`.

References `st_avail_get()`, `st_char_get()`, `st_length_get()`, and `st_length_set()`.

Referenced by `imap_parse_address_part()`.

magma/servers/imap/range.c File Reference

Functions used to handle IMAP commands/actions.

```
#include "magma.h"
```

Functions

- **stringer_t * imap_range_build** (size_t *length*, uint64_t **numbers*)
range.c
-

Detailed Description

Functions used to handle IMAP commands/actions.

Definition in file **range.c**.

Function Documentation

stringer_t* imap_range_build (size_t *length*, uint64_t * *numbers*)

range.c

Definition at line 15 of file range.c.

References count, log_pedantic, st_free(), and st_merge.

Referenced by imap_copy().

magma/servers/molten/molten.c File Reference

Functions used to handle Molten commands/actions.

```
#include "magma.h"
```

Functions

- void **molten_stats** (**connection_t** *con)
- void **molten_invalid** (**connection_t** *con)
- void **molten_quit** (**connection_t** *con)
- void **molten_init** (**connection_t** *con)

molten.c

Detailed Description

Functions used to handle Molten commands/actions.

Definition in file **molten.c**.

Function Documentation

void molten_init (connection_t * con)

molten.c

Definition at line 58 of file molten.c.

References `enqueue()`, and `molten_parse()`.

Referenced by `protocol_enqueue()`.

void molten_invalid (connection_t * con)

Definition at line 46 of file molten.c.

References `con_write_bl()`, `enqueue()`, `molten_parse()`, and `molten_quit()`.

Referenced by `molten_parse()`.

void molten_quit (connection_t * con)

Definition at line 52 of file molten.c.

References `con_destroy()`.

Referenced by `molten_invalid()`, `molten_parse()`, and `molten_stats()`.

void molten_stats (connection_t * con)

Definition at line 15 of file molten.c.

References `con_print()`, `con_write_bl()`, `derived_count()`, `derived_name()`, `derived_value()`, `enqueue()`, `length`, `molten_parse()`, `molten_quit()`, `stats_get_count()`, `stats_get_name()`, and `stats_get_value_by_num()`.

magma/servers/molten/molten.h File Reference

The entry point for the Molten server module.

Functions

- void **molten_init** (**connection_t** *con)
- *molten.c* void **molten_invalid** (**connection_t** *con)
- void **molten_quit** (**connection_t** *con)
- void **molten_stats** (**connection_t** *con)
- **int_t** **molten_compare** (const void *compare, const void *command)
- *commands.c* void **molten_parse** (**connection_t** *con)
- void **molten_sort** (void)
- *Sort the Molten command table to be ready for binary searches.* void **molten_session_destroy** (**connection_t** *con)

sessions.c

Detailed Description

The entry point for the Molten server module.

Definition in file **molten.h**.

Function Documentation

int_t **molten_compare** (const void * *compare*, const void * *command*)

commands.c

Definition at line 17 of file *commands.c*.

References *command_t::length*, *PLACER*, *st_cmp_ci_eq()*, *st_cmp_ci_starts()*, and *command_t::string*.

Referenced by *molten_parse()*, and *molten_sort()*.

void **molten_init** (**connection_t** * *con*)

molten.c

Definition at line 58 of file *molten.c*.

References *enqueue()*, and *molten_parse()*.

Referenced by *protocol_enqueue()*.

void **molten_invalid** (**connection_t** * *con*)

Definition at line 46 of file *molten.c*.

References *con_write_bl()*, *enqueue()*, *molten_parse()*, and *molten_quit()*.

Referenced by *molten_parse()*.

void molten_parse (connection_t * con)

Definition at line 37 of file commands.c.

References `connection_t::command`, `command`, `con_read_line()`, `enqueue()`, `command_t::function`, `command_t::length`, `connection_t::line`, `molten_commands`, `molten_compare()`, `molten_invalid()`, `molten_parse()`, `molten_quit()`, `connection_t::network`, `pl_char_get()`, `pl_empty()`, `pl_length_get()`, and `command_t::string`.

Referenced by `molten_init()`, `molten_invalid()`, `molten_parse()`, and `molten_stats()`.

void molten_quit (connection_t * con)

Definition at line 52 of file molten.c.

References `con_destroy()`.

Referenced by `molten_invalid()`, `molten_parse()`, and `molten_stats()`.

void molten_session_destroy (connection_t * con)

`sessions.c`

Definition at line 15 of file `sessions.c`.

Referenced by `con_destroy()`.

void molten_sort (void)

Sort the Molten command table to be ready for binary searches.

Returns:

This function returns no value.

Definition at line 32 of file `commands.c`.

References `molten_commands`, and `molten_compare()`.

Referenced by `protocol_init()`.

void molten_stats (connection_t * con)

Definition at line 15 of file `molten.c`.

References `con_print()`, `con_write_bl()`, `derived_count()`, `derived_name()`, `derived_value()`, `enqueue()`, `length`, `molten_parse()`, `molten_quit()`, `stats_get_count()`, `stats_get_name()`, and `stats_get_value_by_num()`.

magma/servers/pop/mailbox.c File Reference

Utility functions used to retrieve POP3 message-related statistics.

```
#include "magma.h"
```

Functions

- `uint64_t pop_total_messages (inx_t *messages)`
- *Get the number of messages available to a POP3 user.* `uint64_t pop_total_size (inx_t *messages)`
- *Get the total size of all messages available to a POP3 user.* `uint64_t pop_get_last (inx_t *messages)`
- *Get the POP3 sequence number of the last message that isn't flagged as recent.* `meta_message_t * pop_get_message (inx_t *messages, uint64_t get)`

Get a message by its pop sequence number.

Detailed Description

Utility functions used to retrieve POP3 message-related statistics.

Definition in file **mailbox.c**.

Function Documentation

`uint64_t pop_get_last (inx_t * messages)`

Get the POP3 sequence number of the last message that isn't flagged as recent.

mailbox.c

Note:

This function only counts messages that aren't deleted or hidden, and weren't created by the IMAP APPEND command.

Parameters:

messages an inx holder containing a collection of messages to be analyzed.

Returns:

the sequence number of the last message that isn't flagged as recent, or 0 on failure.

Definition at line 79 of file mailbox.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_APPENDED`, `MAIL_STATUS_HIDDEN`, `MAIL_STATUS_RECENT`, and `meta_message_t::status`.

Referenced by `pop_last()`.

`meta_message_t* pop_get_message (inx_t * messages, uint64_t get)`

Get a message by its pop sequence number.

Parameters:

messages an inx holder containing the collection of the user's messages to be traversed.

number the zero-based pop sequence number of the message to be retrieved.

Returns:

NULL on failure or the meta message object of the message if it was found.
Definition at line 118 of file mailbox.c.

References `count`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_APPENDED`, and `meta_message_t::status`.

Referenced by `pop_delete()`, `pop_list()`, `pop_retr()`, `pop_top()`, and `pop_uidl()`.

uint64_t pop_total_messages (inx_t * *messages*)

Get the number of messages available to a POP3 user.

Note:

This function only counts messages that aren't deleted or hidden, and weren't created by the IMAP APPEND command.

Parameters:

messages an inx holder containing a collection of messages to be analyzed.

Returns:

the total number of messages available to the POP3 user, or 0 on failure.

Definition at line 21 of file mailbox.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_APPENDED`, `MAIL_STATUS_HIDDEN`, and `meta_message_t::status`.

Referenced by `pop_list()`, `pop_stat()`, and `pop_uidl()`.

uint64_t pop_total_size (inx_t * *messages*)

Get the total size of all messages available to a POP3 user.

Note:

This function only counts messages that aren't deleted or hidden, and weren't created by the IMAP APPEND command.

Parameters:

messages an inx holder containing a collection of messages to be analyzed.

Returns:

the total size, in bytes, of all messages available to the POP3 user, or 0 on failure.

Definition at line 50 of file mailbox.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_APPENDED`, `MAIL_STATUS_HIDDEN`, `meta_message_t::size`, and `meta_message_t::status`.

Referenced by `pop_stat()`.

magma/servers/pop/pop.c File Reference

Functions used to handle POP commands/actions.

```
#include "magma.h"
```

Functions

- void **pop_starttls** (**connection_t** *con)
- *TODO: Review error messages and update them with the appropriate response code.* void **pop_noop** (**connection_t** *con)
- Execute a POP3 no-operation command. void **pop_invalid** (**connection_t** *con)
- A function handler for invalid POP3 commands. void **pop_rset** (**connection_t** *con)
- Reset the user's mailbox, in response to a POP3 RSET command. void **pop_quit** (**connection_t** *con)
- Gracefully destroy a POP3 session, whether because of an error or in response to a user QUIT command. void **pop_user** (**connection_t** *con)
- Accept a username for POP3 authentication. void **pop_pass** (**connection_t** *con)
- Accept and verify a password for POP3 authentication. void **pop_capa** (**connection_t** *con)
- Display the POP3 server capabilities, in response to a POP3 CAPA command. void **pop_stat** (**connection_t** *con)
- Display a user's message statistics, in response to a POP3 STAT command. void **pop_last** (**connection_t** *con)
- Get the sequence number of the last read message, in response to a POP3 LAST command. void **pop_list** (**connection_t** *con)
- Get the list of a user's messages, in response to a POP3 LIST command. void **pop_dele** (**connection_t** *con)
- Get a message, in response to a POP3 DELE command. void **pop_uidl** (**connection_t** *con)
- Get the UIDL for a message or collection of messages, in response to a POP3 UIDL command. void **pop_top** (**connection_t** *con)
- Get the top lines of a message or collection of messages, in response to a POP3 TOP command. void **pop_retr** (**connection_t** *con)
- Retrieve a user's message, in response to a POP3 RETR command. void **pop_init** (**connection_t** *con)

Initialize a new POP3 connection.

Detailed Description

Functions used to handle POP commands/actions.

Definition in file **pop.c**.

Function Documentation

void pop_capa (connection_t * con)

Display the POP3 server capabilities, in response to a POP3 CAPA command.

pop.c

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 303 of file pop.c.

References `build_version()`, `con_print()`, `con_secure()`, and `connection_t::pop`.

void pop_dele (connection_t * con)

Get a message, in response to a POP3 DELE command.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 439 of file `pop.c`.

References `con_write_bl()`, `MAIL_STATUS_HIDDEN`, `meta_user_unlock()`, `meta_user_wlock()`, `number`, `connection_t::pop`, `pop_get_message()`, `pop_invalid()`, `pop_num_parse()`, and `meta_message_t::status`.

void pop_init (connection_t * con)

Initialize a new POP3 connection.

Parameters:

con the newly connected POP3 client connection.

Returns:

This function returns no value.

Definition at line 684 of file `pop.c`.

References `con_write_bl()`, and `pop_requeue()`.

Referenced by `protocol_enqueue()`.

void pop_invalid (connection_t * con)

A function handler for invalid POP3 commands.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 72 of file `pop.c`.

References `con_write_bl()`, `server_t::delay`, `connection_t::protocol`, `connection_t::server`, `server_t::violations`, and `connection_t::violations`.

Referenced by `pop_dele()`, `pop_last()`, `pop_list()`, `pop_pass()`, `pop_process()`, `pop_retr()`, `pop_rset()`, `pop_starttls()`, `pop_stat()`, `pop_top()`, `pop_uidl()`, and `pop_user()`.

void pop_last (connection_t * con)

Get the sequence number of the last read message, in response to a POP3 LAST command.

See also:

pop_get_last()

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 351 of file pop.c.

References `con_print()`, `meta_user_rlock()`, `meta_user_unlock()`, `number`, `connection_t::pop`, `pop_get_last()`, and `pop_invalid()`.

void pop_list (connection_t * *con*)

Get the list of a user's messages, in response to a POP3 LIST command.

See also:

pop_total_messages(), **pop_get_message()**

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 376 of file pop.c.

References `con_print()`, `con_write_bl()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_APPENDED`, `MAIL_STATUS_HIDDEN`, `meta_user_rlock()`, `meta_user_unlock()`, `number`, `connection_t::pop`, `pop_get_message()`, `pop_invalid()`, `pop_num_parse()`, `pop_total_messages()`, `meta_message_t::size`, and `meta_message_t::status`.

void pop_noop (connection_t * *con*)

Execute a POP3 no-operation command.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 61 of file pop.c.

References `con_write_bl()`.

void pop_pass (connection_t * *con*)

Accept and verify a password for POP3 authentication.

Note:

This command is only allowed for sessions which have not yet been authenticated, but which have already supplied a username. If the username/password combo was validated, the account information is retrieved and checked to see if it is locked. After successful authentication, this function will prohibit insecure connections for any user configured to use SSL only, and enforce the existence of only one POP3 session at a time. Finally, the database Log table for this user's POP3 access is updated, and all the user's messages are retrieved.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 175 of file pop.c.

References `credential_t::auth`, `con_addr_presentation()`, `con_secure()`, `con_write_bl()`, `credential_alloc_auth()`, `credential_free()`, `credential_username()`, `inx_count()`, `log_pedantic`, `MANAGEDBUF`, `meta_data_update_log()`, `meta_get()`, `META_GET_MESSAGES`, `META_LOCKED`, `meta_messages_login_update()`, `META_PROT_POP`, `meta_remove()`, `META_USER_SSL`, `meta_user_unlock()`, `meta_user_wlock()`, `connection_t::pop`, `pop_invalid()`, `pop_pass_parse()`, `st_char_get()`, `st_empty()`, `st_free()`, `st_length_int()`, and `st_wipe()`.

void pop_quit (connection_t * con)

Gracefully destroy a POP3 session, whether because of an error or in response to a user QUIT command.

Parameters:

con the POP3 client connection to be shut down. This function returns no value.

Definition at line 112 of file pop.c.

References `con_destroy()`, `con_status()`, and `con_write_bl()`.

Referenced by `pop_process()`, and `pop_queue()`.

void pop_retr (connection_t * con)

Retrieve a user's message, in response to a POP3 RETR command.

Note:

This function will fail if a deleted message was specified by the user.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 614 of file pop.c.

References `con_print()`, `con_write_bl()`, `con_write_st()`, `mail_destroy()`, `mail_load_message()`, `MAIL_STATUS_HIDDEN`, `meta_user_rlock()`, `meta_user_unlock()`, `number`, `PLACER`, `connection_t::pop`, `pop_get_message()`, `pop_invalid()`, `pop_num_parse()`, `connection_t::server`, `st_char_get()`, `st_length_get()`, `st_replace()`, `meta_message_t::status`, and `mail_message_t::text`.

void pop_rset (connection_t * con)

Reset the user's mailbox, in response to a POP3 RSET command.

See also:

pop_session_reset()

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 87 of file pop.c.

References `con_write_bl()`, `connection_t::pop`, `pop_invalid()`, and `pop_session_reset()`.

void pop_starttls (connection_t * con)

TODO: Review error messages and update them with the appropriate response code.

Initialize a TLS session for an unauthenticated POP3 session.

Note:

RFC 2595 / section 4 dictates that the STLS/STARTTLS command should only be available in the authorization state.

Parameters:

con the connection of the POP3 client requesting the transport layer security upgrade.

Returns:

This function returns no value (all error messages are written directly to the requesting client).

Definition at line 23 of file pop.c.

References `connection_t::buffer`, `con_secure()`, `con_write_bl()`, `connection_t::line`, `log_pedantic`, `M_SSL_BIO_NOCLOSE`, `connection_t::network`, `pl_null()`, `connection_t::pop`, `pop_invalid()`, `pop_session_reset()`, `connection_t::server`, `connection_t::sockd`, `connection_t::ssl`, `ssl_alloc()`, `st_length_set()`, `stats_increment_by_name()`, and `connection_t::status`.

void pop_stat (connection_t * con)

Display a user's message statistics, in response to a POP3 STAT command.

See also:

pop_total_messages(), **pop_total_size()**

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 325 of file pop.c.

References `con_print()`, `count`, `meta_user_rlock()`, `meta_user_unlock()`, `connection_t::pop`, `pop_invalid()`, `pop_total_messages()`, and `pop_total_size()`.

void pop_top (connection_t * con)

Get the top lines of a message or collection of messages, in response to a POP3 TOP command.

Note:

This function will fail if a deleted message was specified by the user.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 543 of file pop.c.

References `con_print()`, `con_write_bl()`, `con_write_st()`, `mail_destroy()`, `mail_load_message_top()`, `MAIL_STATUS_HIDDEN`, `meta_user_rlock()`, `meta_user_unlock()`, `number`, `PLACER`, `connection_t::pop`, `pop_get_message()`, `pop_invalid()`, `pop_top_parse()`, `connection_t::server`, `st_char_get()`, `st_length_get()`, `st_replace()`, `meta_message_t::status`, and `mail_message_t::text`.

void pop_uidl (connection_t * con)

Get the UIDL for a message or collection of messages, in response to a POP3 UIDL command.

Parameters:

con the POP3 client connection issuing the command.

Returns:

This function returns no value.

Definition at line 478 of file pop.c.

References `con_print()`, `con_write_bl()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MAIL_STATUS_APPENDED`, `MAIL_STATUS_HIDDEN`, `meta_message_t::messagenum`, `meta_user_rlock()`, `meta_user_unlock()`, `number`, `connection_t::pop`, `pop_get_message()`, `pop_invalid()`, `pop_num_parse()`, `pop_total_messages()`, and `meta_message_t::status`.

void pop_user (connection_t * con)

Accept a username for POP3 authentication.

Note:

This command is only allowed for sessions which have not yet been authenticated. If the username has already been supplied pre-authentication, the old value will be overwritten with the new one.

Parameters:

con the POP3 client connection issuing the command. This function returns no value.

Definition at line 136 of file pop.c.

References `con_write_bl()`, `credential_address()`, `connection_t::pop`, `pop_invalid()`, `pop_user_parse()`, `st_cleanup()`, and `st_free()`.

magma/servers/smtp/accept.c File Reference

Functions used to handle SMTP commands/actions.

#include "magma.h"

Functions

- **int_t smtp_store_message** (smtp_inbound_prefs_t *prefs, stringer_t **local)
- *Store a received SMTP message as a generic mail message, both on disk and in the database.* **int_t smtp_rollout** (smtp_inbound_prefs_t *prefs)
- *Delete the oldest mail message owned by a user until their storage usage falls below their storage quota.* **bool_t smtp_store_spamsig** (smtp_inbound_prefs_t *prefs, int_t spam)
- *Generate a random key for a spam signature and store it in the database.* **int_t smtp_accept_message** (connection_t *con, smtp_inbound_prefs_t *prefs)

accept.c

Detailed Description

Functions used to handle SMTP commands/actions.

Definition in file **accept.c**.

Function Documentation

int_t smtp_accept_message (connection_t * con, smtp_inbound_prefs_t * prefs)

accept.c

Definition at line 201 of file accept.c.

References magma_t::abuse, magma_t::admin, smtp_inbound_prefs_t::autoreply, smtp_inbound_prefs_t::bounces, smtp_session_t::bypass, smtp_session_t::checked, magma_t::contact, smtp_session_t::dkim, smtp_inbound_prefs_t::dkim, dkim_check(), smtp_inbound_prefs_t::dkimaction, dspam_check(), smtp_inbound_prefs_t::filters, smtp_inbound_prefs_t::foldernum, smtp_inbound_prefs_t::forwarded, smtp_message_t::id, smtp_inbound_prefs_t::inbox, log_error, log_pedantic, magma, mail_add_inbound_headers(), smtp_session_t::mailfrom, smtp_inbound_prefs_t::mark, smtp_session_t::message, smtp_inbound_prefs_t::overquota, smtp_inbound_prefs_t::phish, smtp_inbound_prefs_t::phishaction, PLACER, smtp_session_t::rbl, smtp_inbound_prefs_t::rbl, smtp_inbound_prefs_t::rblaction, smtp_inbound_prefs_t::rcptto, smtp_inbound_prefs_t::recv_size_limit, smtp_inbound_prefs_t::rollout, connection_t::server, smtp_inbound_prefs_t::signum, connection_t::smtp, SMTP_ACTION_BOUNCE, SMTP_ACTION_DELETE, SMTP_ACTION_MARK, SMTP_ACTION_MARK_READ, smtp_check_filters(), smtp_forward_message(), SMTP_MARK_NONE, SMTP_MARK_PHISH, SMTP_MARK_RBL, SMTP_MARK_READ, SMTP_MARK_SPAM, SMTP_MARK_SPOOF, SMTP_MARK_VIRUS, SMTP_OUTCOME_BOUNCE_DKIM, SMTP_OUTCOME_BOUNCE_PHISH, SMTP_OUTCOME_BOUNCE_RBL, SMTP_OUTCOME_BOUNCE_SPAM, SMTP_OUTCOME_BOUNCE_SPF, SMTP_OUTCOME_BOUNCE_VIRUS, SMTP_OUTCOME_PERM_FAILURE, SMTP_OUTCOME_SUCESS, SMTP_OUTCOME_TEMP_LOCKED, SMTP_OUTCOME_TEMP_OVERQUOTA, SMTP_OUTCOME_TEMP_SERVER, smtp_reply(), smtp_rollout(), smtp_store_message(), smtp_store_spamsig(), smtp_update_receive_stats(), smtp_inbound_prefs_t::spam, smtp_inbound_prefs_t::spam_checked, smtp_inbound_prefs_t::spamaction, smtp_inbound_prefs_t::spamkey, smtp_inbound_prefs_t::spamsig, smtp_session_t::spf,

smtp_inbound_prefs_t::spf, smtp_inbound_prefs_t::spfaction, st_cmp_ci_eq(), st_cmp_cs_eq(), st_free(), st_length_get(), smtp_message_t::text, smtp_inbound_prefs_t::usernum, smtp_session_t::virus, smtp_inbound_prefs_t::virus, virus_check(), and smtp_inbound_prefs_t::virusaction.

Referenced by smtp_data_inbound().

int_t smtp_rollout (smtp_inbound_prefs_t * prefs)

Delete the oldest mail message owned by a user until their storage usage falls below their storage quota.

Parameters:

prefs a pointer to the specified user's inbound mail preferences data.

Returns:

1 on success or < 0 on failure, where -1: An error occurred retrieving the rollout message list from the database. -2: The user lock could not be acquired.

Definition at line 95 of file accept.c.

References log_pedantic, mail_remove_message(), OBJECT_MESSAGES, smtp_inbound_prefs_t::quota, res_field_string(), res_field_uint32(), res_field_uint64(), res_row_next(), res_table_free(), serial_increment(), smtp_fetch_rollmessages(), st_char_get(), st_free(), smtp_inbound_prefs_t::stor_size, user_lock(), user_unlock(), and smtp_inbound_prefs_t::usernum.

Referenced by smtp_accept_message().

int_t smtp_store_message (smtp_inbound_prefs_t * prefs, stringer_t ** local)

Store a received SMTP message as a generic mail message, both on disk and in the database.

See also:

mail_store_messages()

Returns:

-1 on failure or 1 on success.

Definition at line 20 of file accept.c.

References smtp_inbound_prefs_t::foldernum, log_error, log_pedantic, MAIL_MARK_BLACKHOLED, MAIL_MARK_INFECTED, MAIL_MARK_JUNK, MAIL_MARK_PHISHING, MAIL_MARK_SPOOFED, MAIL_STATUS_RECENT, MAIL_STATUS_SEEN, mail_store_message(), smtp_inbound_prefs_t::mark, smtp_inbound_prefs_t::messagenum, meta_data_user_build_storage_keys(), OBJECT_MESSAGES, smtp_inbound_prefs_t::secure, serial_increment(), smtp_inbound_prefs_t::signum, SMTP_MARK_PHISH, SMTP_MARK_RBL, SMTP_MARK_READ, SMTP_MARK_SPAM, SMTP_MARK_SPOOF, SMTP_MARK_VIRUS, smtp_inbound_prefs_t::spamkey, status, user_lock(), user_unlock(), and smtp_inbound_prefs_t::usernum.

Referenced by smtp_accept_message().

bool_t smtp_store_spamsig (smtp_inbound_prefs_t * prefs, int_t spam)

Generate a random key for a spam signature and store it in the database.

Parameters:

prefs the user's smtp inbound preferences object, with the spam signature field set.

spam the dspam return code associated with the spam signature.

Returns:

true if the key was inserted into the database successfully, or false on failure.

Definition at line 177 of file accept.c.

References `imap_fetch_response_t::key`, `log_pedantic`, `rand_get_uint64()`, `smtp_inbound_prefs_t::signum`, `smtp_insert_spamsig()`, `smtp_inbound_prefs_t::spamkey`, and `uint64_digits()`.

Referenced by `smtp_accept_message()`.

magma/servers/smtp/checkers.c File Reference

Functions used by the SMTP protocol to check and if necessary validate data using external information.

```
#include "magma.h"
```

Functions

- **int_t smtp_check_greylist** (**connection_t** *con, **smtp_inbound_prefs_t** *prefs)
 - *Check to see if a transmitting address is in a user's greylist.* **int_t smtp_check_rbl** (**connection_t** *con)
 - *Check the SMTP connection's remote address against a collection of real-time blacklists.* **int_t smtp_check_filters** (**smtp_inbound_prefs_t** *prefs, **stringer_t** **local)
 - *checkers.c* **bool_t smtp_add_bypass_entry** (**stringer_t** *subnet)
 - *Add an entry to the SMTP subnet bypass list.* **bool_t smtp_bypass_check** (**connection_t** *con)
- Check if a connection should bypass certain SMTP checks.*
-

Detailed Description

Functions used by the SMTP protocol to check and if necessary validate data using external information.

Definition in file **checkers.c**.

Function Documentation

bool_t smtp_add_bypass_entry (**stringer_t** * subnet)

Add an entry to the SMTP subnet bypass list.

Parameters:

subnet a pointer to a managed string containing the IP address or subnet address to be bypassed for checks.

Returns:

true if the entry was valid and was added, or false on failure.

Definition at line 265 of file checkers.c.

References magma_t::bypass_subnets, inx_alloc(), inx_insert(), ip_str_subnet(), log_error, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, magma, mm_alloc(), mm_free(), magma_t::smtp, st_char_get(), multi_t::type, multi_t::u64, and multi_t::val.

Referenced by config_value_set().

bool_t smtp_bypass_check (**connection_t** * con)

Check if a connection should bypass certain SMTP checks.

Note:

This check is run against host and/or subnet masks configured in the magma.smtp.bypass_addr option.

Parameters:

con a pointer to the connection object to be checked.

Returns:

true if the specified connection meets the SMTP bypass check or false on failure or if it does not.

Definition at line 305 of file checkers.c.

References magma_t::bypass_subnets, con_addr(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), ip_matches_subnet(), magma, and magma_t::smtp.

Referenced by smtp_init().

int_t smtp_check_filters (smtp_inbound_prefs_t * *prefs*, stringer_t ** *local*)

checkers.c

Apply any user specific filters. Return -1 for errors, and -2 to delete a message. Return 1 if no action was taken, and 2 if the message was moved to a different folder, 3 if the message content was modified, and 4 if the message was marked read.

Definition at line 144 of file checkers.c.

References data, smtp_inbound_prefs_t::filters, smtp_inbound_prefs_t::foldernum, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), length, log_pedantic, mail_header_end(), mail_header_fetch_all(), mail_mod_subject(), smtp_inbound_prefs_t::mark, mm_wipe(), pl_data_get(), pl_init(), pl_length_get(), pl_null(), PLACER, placer_t, SMTP_FILTER_ACTION_DELETE, SMTP_FILTER_ACTION_LABEL, SMTP_FILTER_ACTION_MARK_READ, SMTP_FILTER_ACTION_MOVE, SMTP_FILTER_LOCATION_BODY, SMTP_FILTER_LOCATION_ENTIRE, SMTP_FILTER_LOCATION_FIELD, SMTP_FILTER_LOCATION_HEADER, SMTP_MARK_NONE, SMTP_MARK_PHISH, SMTP_MARK_RBL, SMTP_MARK_READ, SMTP_MARK_SPAM, SMTP_MARK_SPOOF, SMTP_MARK_VIRUS, st_char_get(), st_cleanup(), st_cmp_ci_eq(), st_length_get(), st_length_int(), and smtp_inbound_prefs_t::usernum.

Referenced by smtp_accept_message().

int_t smtp_check_greylist (connection_t * *con*, smtp_inbound_prefs_t * *prefs*)

Check to see if a transmitting address is in a user's greylist.

Note:

The greylist is configured in the Dispatch table and specifies the minimum time, in minutes, that a transmitting smtp relay server must wait in order to be able to send more messages to the same recipient address again.

Parameters:

con the connection to have its remote address checked against the user's greylist.

prefs the smtp inbound preferences of the user

Returns:

-1 on general error, -2 if the check failed, and 1 if the check was passed.

Definition at line 27 of file checkers.c.

References BLOCK_T, smtp_session_t::bypass, cache_get(), cache_set(), con_addr_reversed(), CONTIGUOUS, smtp_inbound_prefs_t::greytime, HEAP, imap_fetch_response_t::key, log_pedantic, MANAGEDBUF, connection_t::smtp, st_alloc_opts(), st_char_get(), st_cleanup(), st_data_get(), st_length_get(), st_length_int(), st_sprint(), smtp_inbound_prefs_t::usernum, and imap_fetch_response_t::value.

Referenced by smtp_rcpt_to().

int_t smtp_check_rbl (connection_t * con)

Check the SMTP connection's remote address against a collection of real-time blacklists.

Note:

The connection's IP address will be checked against each of the servers configured in magma.smtp.blacklists.domain.

Parameters:

con the connection to have its address examined against the RBLs.

Returns:

-1 on general error, -2 if the address was blacklisted, or 1 if it passed the check.

Definition at line 95 of file checkers.c.

References magma_t::blacklists, con_addr_reversed(), log_pedantic, magma, MANAGEDBUF, mm_wipe(), magma_t::smtp, st_char_get(), and st_length_int().

Referenced by smtp_rcpt_to().

magma/servers/smtp/session.c File Reference

Functions used to handle SMTP sessions.

```
#include "magma.h"
```

Functions

- void **smtp_session_reset** (**connection_t** *con)
- *Reset an SMTP session to its initialized state.* void **smtp_session_destroy** (**connection_t** *con)
- *Destroy the data associated with an SMTP session.* **bool_t** **smtp_check_duplicate_recipient** (**connection_t** *con, **uint64_t** usernum)
- **bool_t** **smtp_add_recipient** (**connection_t** *con, **stringer_t** *address)
- *Add an entry to an SMTP session's recipients list for outbound/relayed mail.* void **smtp_add_inbound** (**connection_t** *con, **smtp_inbound_prefs_t** *inbound)
- *Add an entry to an SMTP session's inbound preferences list for local delivery.* void **smtp_add_outbound** (**connection_t** *con, **smtp_outbound_prefs_t** *outbound)
- *Attach a set of SMTP outbound mail preferences to an SMTP client connection.* void **smtp_free_recipients** (**smtp_recipients_t** *recipients)
- *Free a list of SMTP recipients and its underlying data.* void **smtp_free_outbound** (**smtp_outbound_prefs_t** *outbound)
- *Free a set of SMTP outbound mail preferences and its underlying data.* void **smtp_free_inbound** (**smtp_inbound_prefs_t** *inbound)
- *Free a list of SMTP inbound mail preferences and its underlying data.* void **smtp_list_free_filter** (**smtp_inbound_filter_t** *filter)

Free an SMTP inbound filter and its underlying data.

Detailed Description

Functions used to handle SMTP sessions.

Definition in file **session.c**.

Function Documentation

void smtp_add_inbound (connection_t * con, smtp_inbound_prefs_t * inbound)

Add an entry to an SMTP session's inbound preferences list for local delivery.

session.c

Parameters:

con the SMTP client connection specifying the added local recipient.

inbound a pointer to the SMTP inbound preferences object to be added to the SMTP session's inbound preferences list.

Returns:

true if the requested inbound preferences object was added successfully to the inbound preferences list, or false on failure.

Definition at line 157 of file **session.c**.

References **smtp_session_t::in_prefs**, **smtp_inbound_prefs_t::next**, **smtp_session_t::num_recipients**, and **connection_t::smtp**.

Referenced by smtp_rcpt_to().

void smtp_add_outbound (connection_t * con, smtp_outbound_prefs_t * outbound)

Attach a set of SMTP outbound mail preferences to an SMTP client connection.

Parameters:

con the SMTP client connection to which the outbound mail preferences should be attached.

outbound a pointer to the SMTP outbound mail preferences to be set for the specified connection.

Returns:

This function returns no value.

Definition at line 186 of file session.c.

References smtp_session_t::out_prefs, and connection_t::smtp.

Referenced by smtp_auth_login(), and smtp_auth_plain().

bool_t smtp_add_recipient (connection_t * con, stringer_t * address)

Add an entry to an SMTP session's recipients list for outbound/relayed mail.

Note:

If the recipients list does not exist, it will be created automatically.

Parameters:

con the SMTP client connection specifying the added recipient.

address a managed string containing the recipient's email address.

Returns:

true if the requested recipient was added successfully to the recipients list, or false on failure.

Definition at line 114 of file session.c.

References smtp_recipients_t::address, log_pedantic, mm_alloc(), mm_free(), smtp_recipients_t::next, smtp_session_t::num_recipients, smtp_session_t::out_prefs, smtp_outbound_prefs_t::recipients, connection_t::smtp, and st_dup().

Referenced by smtp_rcpt_to().

bool_t smtp_check_duplicate_recipient (connection_t * con, uint64_t usernum)

Definition at line 87 of file session.c.

References smtp_session_t::in_prefs, smtp_inbound_prefs_t::next, connection_t::smtp, and smtp_inbound_prefs_t::usernum.

Referenced by smtp_rcpt_to().

void smtp_free_inbound (smtp_inbound_prefs_t * inbound)

Free a list of SMTP inbound mail preferences and its underlying data.

Parameters:

inbound a pointer to the head of the SMTP inbound mail preferences list to be destroyed.

Returns:

This function returns no value.

Definition at line 235 of file session.c.

References smtp_inbound_prefs_t::address, smtp_inbound_prefs_t::domain, smtp_inbound_prefs_t::filters, smtp_inbound_prefs_t::forwarded, inx_cleanup(), mm_free(), smtp_inbound_prefs_t::next, smtp_inbound_prefs_t::rcptto, smtp_inbound_prefs_t::spamsig, and st_cleanup().

Referenced by smtp_fetch_inbound(), smtp_rcpt_to(), smtp_session_destroy(), and smtp_session_reset().

void smtp_free_outbound (smtp_outbound_prefs_t * *outbound*)

Free a set of SMTP outbound mail preferences and its underlying data.

Parameters:

outbound a pointer to the SMTP outbound mail preferences set to be destroyed.

Returns:

This function returns no value.

Definition at line 217 of file session.c.

References smtp_outbound_prefs_t::domain, mm_free(), smtp_outbound_prefs_t::recipients, smtp_free_recipients(), and st_cleanup().

Referenced by smtp_session_destroy().

void smtp_free_recipients (smtp_recipients_t * *recipients*)

Free a list of SMTP recipients and its underlying data.

Parameters:

recipients a pointer to the head of the recipients list to be destroyed.

Returns:

This function returns no value.

Definition at line 198 of file session.c.

References smtp_recipients_t::address, mm_free(), smtp_recipients_t::next, and st_cleanup().

Referenced by smtp_free_outbound(), and smtp_session_reset().

void smtp_list_free_filter (smtp_inbound_filter_t * *filter*)

Free an SMTP inbound filter and its underlying data.

Parameters:

filter a pointer to the SMTP inbound filter to be destroyed.

Returns:

This function returns no value.

Definition at line 259 of file session.c.

References `smtp_inbound_filter_t::expression`, `smtp_inbound_filter_t::field`, `smtp_inbound_filter_t::label`, `mm_free()`, and `st_cleanup()`.

Referenced by `smtp_fetch_inbound()`.

void smtp_session_destroy (connection_t * con)

Destroy the data associated with an SMTP session.

Parameters:

con the SMTP client connection to be destroyed.

Returns:

This function returns no value.

Definition at line 62 of file `session.c`.

References `smtp_session_t::helo`, `smtp_session_t::in_prefs`, `mail_destroy_message()`, `smtp_session_t::mailfrom`, `smtp_session_t::message`, `smtp_session_t::out_prefs`, `connection_t::smtp`, `smtp_free_inbound()`, `smtp_free_outbound()`, and `st_cleanup()`.

Referenced by `con_destroy()`.

void smtp_session_reset (connection_t * con)

Reset an SMTP session to its initialized state.

Parameters:

con the SMTP client connection to be reset.

Returns:

This function returns no value.

Definition at line 20 of file `session.c`.

References `smtp_session_t::checked`, `smtp_session_t::dkim`, `smtp_session_t::in_prefs`, `mail_destroy_message()`, `smtp_session_t::mailfrom`, `smtp_session_t::max_length`, `smtp_session_t::message`, `smtp_session_t::num_recipients`, `smtp_session_t::out_prefs`, `smtp_session_t::rbl`, `smtp_outbound_prefs_t::recipients`, `connection_t::smtp`, `smtp_free_inbound()`, `smtp_free_recipients()`, `smtp_session_t::spf`, `st_cleanup()`, `smtp_session_t::suggested_eight_bit`, `smtp_session_t::suggested_length`, and `smtp_session_t::virus`.

Referenced by `smtp_auth_login()`, `smtp_auth_plain()`, `smtp_data_inbound()`, `smtp_data_outbound()`, `smtp_mail_from()`, `smtp_rset()`, and `smtp_starttls()`.

magma/servers/smtp/smtp.c File Reference

Functions used to handle SMTP commands/actions.

```
#include "magma.h"
```

Functions

- void **smtp_starttls** (**connection_t** *con)
- *TODO: Review error messages and update them with the appropriate response code.* void **smtp_mail_from** (**connection_t** *con)
- *Specify the identity of a message's sender, in response to an SMTP MAIL FROM command.* void **smtp_ehlo** (**connection_t** *con)
- *Process an SMTP EHLO command.* void **smtp_helo** (**connection_t** *con)
- *Process an SMTP HELO command.* void **smtp_noop** (**connection_t** *con)
- *Perform an SMTP NOOP (no-operation) command.* void **smtp_disabled** (**connection_t** *con)
- *A stub function for an SMTP command that has not been implemented.* void **smtp_invalid** (**connection_t** *con)
- *A function that is executed when an invalid SMTP command is executed.* void **smtp_quit** (**connection_t** *con)
- *Gracefully terminate an SMTP session, especially in response to an SMTP QUIT command.* void **smtp_rset** (**connection_t** *con)
- *Reset the SMTP session, in response to an SMTP RSET command.* void **smtp_auth_plain** (**connection_t** *con)
- void **smtp_auth_login** (**connection_t** *con)
- *smtp.c* void **smtp_rcpt_to** (**connection_t** *con)
- void **smtp_data_finish** (**connection_t** *con, size_t read, int_t checker)
- int_t **smtp_data_read** (**connection_t** *con, stringer_t **message)
- void **smtp_data_outbound** (**connection_t** *con)
- void **smtp_data_inbound** (**connection_t** *con)
- void **smtp_data** (**connection_t** *con)
- void **smtp_init** (**connection_t** *con)
- *The start of the protocol handler for the SMTP server.* void **submission_init** (**connection_t** *con)

Detailed Description

Functions used to handle SMTP commands/actions.

Definition in file **smtp.c**.

Function Documentation

void smtp_auth_login (connection_t * con)

smtp.c

BUG: The code should be able to differentiate between invalid usernames which trigger a NULL return and allocation (or similar) errors which are temporary. We approximate this functionality by not rejecting "@domain.com" as a username via the credentials function, but instead check for leading at symbols explicitly below.

Definition at line 360 of file smtp.c.

References credential_t::auth, smtp_session_t::authenticated, base64_decode(), con_read_line(), con_write_bl(), credential_alloc_auth(), credential_free(), connection_t::line, smtp_session_t::max_length, connection_t::network, pl_char_get(), pl_length_get(), PLACER, smtp_outbound_prefs_t::send_size_limit, connection_t::smtp, smtp_add_outbound(), smtp_fetch_authorization(), smtp_parse_auth(), smtp_session_reset(), st_cleanup(), st_cmp_ci_eq(), st_cmp_cs_starts(), st_empty(), and st_free().

void smtp_auth_plain (connection_t * con)

BUG: The SMTP server is not handling AUTH requests in accordance with RFC 2554 <<http://tools.ietf.org/html/rfc2554>> and RFC 4954 <<http://tools.ietf.org/html/rfc4954>>, according to a user.

In this particular case it's erroring out when users present an AUTH PLAIN request without the optional initial response. This could affect other protocols as well.

According to the RFC, a client can send either the following: 1. "AUTH mechanism" (e.g. "AUTH PLAIN"), which the server acknowledges with a "334 " (note the space). The client then sends the base64-encoded authentication response on a separate line.

2. "AUTH mechanism [initial response]" (e.g. "AUTH PLAIN

dGVzdAB0ZXN0ADEyMzQ="), where user transmits the auth mechanism and the base64 authentication response in a single line, so as to minimize the back-and-forth traffic.

BUG: The code should be able to differentiate between invalid usernames which trigger a NULL return and allocation (or similar) errors which are temporary. We approximate this functionality by not rejecting "@domain.com" as a username via the credentials function, but instead check for leading at symbols explicitly below.

Definition at line 246 of file smtp.c.

References credential_t::auth, smtp_session_t::authenticated, base64_decode(), con_read_line(), con_write_bl(), credential_alloc_auth(), credential_free(), FOREIGNDATA, JOINTED, connection_t::line, smtp_session_t::max_length, mm_copy(), connection_t::network, pl_char_get(), pl_length_get(), PLACER, PLACER_T, placer_t, smtp_outbound_prefs_t::send_size_limit, connection_t::smtp, smtp_add_outbound(), smtp_fetch_authorization(), smtp_parse_auth(), smtp_session_reset(), st_cleanup(), st_cmp_ci_eq(), st_cmp_cs_starts(), st_empty(), st_free(), st_length_get(), STACK, and tok_get_st().

void smtp_data (connection_t * con)

LOW: Why are -2 and -3 identical?

Definition at line 1067 of file smtp.c.

References smtp_session_t::authenticated, con_print(), con_write_bl(), smtp_session_t::helo, smtp_session_t::in_prefs, magma, mail_add_required_headers(), mail_count_received(), mail_create_message(), mail_destroy_message(), mail_message_cleanup(), smtp_session_t::mailfrom, smtp_session_t::max_length, smtp_session_t::message, smtp_session_t::out_prefs, smtp_outbound_prefs_t::recipients, magma_t::relay_limit, requeue(), magma_t::smtp, connection_t::smtp, smtp_data_inbound(), smtp_data_outbound(), smtp_data_read(), smtp_quit(), smtp_requeue(), and st_free().

Referenced by smtp_process().

void smtp_data_finish (connection_t * con, size_t read, int_t checker)

Definition at line 721 of file smtp.c.

References `connection_t::buffer`, `con_read()`, `connection_t::line`, `connection_t::network`, `st_char_get()`, `st_data_get()`, `st_data_set()`, `st_length_set()`, and `status`.

Referenced by `smtp_data_read()`.

void smtp_data_inbound (connection_t * con)

Definition at line 1019 of file smtp.c.

References `smtp_inbound_prefs_t::bounces`, `con_print()`, `con_write_bl()`, `smtp_session_t::in_prefs`, `smtp_inbound_prefs_t::next`, `smtp_session_t::num_recipients`, `smtp_inbound_prefs_t::outcome`, `connection_t::smtp`, `smtp_accept_message()`, `smtp_bounce()`, `SMTP_OUTCOME_PERM_FAILURE`, `SMTP_OUTCOME_SUCESS`, `SMTP_OUTCOME_TEMP_LOCKED`, `SMTP_OUTCOME_TEMP_OVERQUOTA`, `SMTP_OUTCOME_TEMP_SERVER`, and `smtp_session_reset()`.

Referenced by `smtp_data()`.

void smtp_data_outbound (connection_t * con)

Definition at line 926 of file smtp.c.

References `con_print()`, `con_write_bl()`, `con_write_st()`, `smtp_outbound_prefs_t::daily_send_limit`, `smtp_message_t::from`, `mail_extract_address()`, `smtp_session_t::mailfrom`, `smtp_session_t::message`, `smtp_session_t::num_recipients`, `smtp_session_t::out_prefs`, `pattern_check()`, `PLACER`, `connection_t::smtp`, `smtp_check_authorized_from()`, `smtp_check_transmit_quota()`, `smtp_relay_message()`, `smtp_session_reset()`, `smtp_update_transmission_stats()`, `st_char_get()`, `st_cmp_ci_eq()`, `st_dupe()`, `st_free()`, `st_length_get()`, `smtp_message_t::text`, `smtp_outbound_prefs_t::usernum`, and `virus_check()`.

Referenced by `smtp_data()`.

int_t smtp_data_read (connection_t * con, stringer_t ** message)

Definition at line 777 of file smtp.c.

References `connection_t::buffer`, `con_read()`, `HEAP`, `JOINTED`, `connection_t::line`, `log_pedantic`, `MAPPED_T`, `smtp_session_t::max_length`, `connection_t::network`, `connection_t::smtp`, `smtp_data_finish()`, `st_alloc_opts()`, `st_char_get()`, `st_data_get()`, `st_data_set()`, `st_free()`, `st_length_set()`, `st_realloc()`, and `status`.

Referenced by `smtp_data()`.

void smtp_disabled (connection_t * con)

A stub function for an SMTP command that has not been implemented.

Note:

Executing a disabled command while result in a small delay and the protocol violation counter being incremented.

Returns:

This function returns no value.

Definition at line 168 of file smtp.c.

References `connection_t::command`, `con_print()`, `server_t::delay`, `command_t::length`, `connection_t::protocol`, `connection_t::server`, `command_t::string`, `server_t::violations`, and `connection_t::violations`.

void smtp_ehlo (connection_t * con)

Process an SMTP EHLO command.

See also:

smtp_parse_helo_domain()

Note:

Any prior domain specified by a HELO/EHLO command will be overwritten.

Parameters:

con the SMTP client connection issuing the command.

Returns:

This function returns no value.

Definition at line 103 of file smtp.c.

References `con_print()`, `con_secure()`, `con_write_bl()`, `server_t::domain`, `smtp_session_t::esmtplib`, `smtp_session_t::helo`, `magma`, `magma_t::message_length_limit`, `connection_t::server`, `magma_t::smtp`, `connection_t::smtp`, `smtp_parse_helo_domain()`, `st_char_get()`, `st_cleanup()`, and `st_length_int()`.

void smtp_helo (connection_t * con)

Process an SMTP HELO command.

See also:

smtp_parse_helo_domain()

Note:

Any prior domain specified by a HELO/EHLO command will be overwritten.

Parameters:

con the SMTP client connection issuing the command.

Returns:

This function returns no value.

Definition at line 132 of file smtp.c.

References `con_print()`, `con_write_bl()`, `server_t::domain`, `smtp_session_t::esmtplib`, `smtp_session_t::helo`, `connection_t::server`, `connection_t::smtp`, `smtp_parse_helo_domain()`, `st_char_get()`, `st_cleanup()`, and `st_length_int()`.

void smtp_init (connection_t * con)

The start of the protocol handler for the SMTP server.

Parameters:

con the new inbound SMTP client connection.

Returns:

This function returns no value.

Definition at line 1177 of file smtp.c.

References smtp_session_t::bypass, con_print(), con_reverse_enqueue(), server_t::domain, connection_t::server, connection_t::smtp, smtp_bypass_check(), smtp_requeue(), st_char_get(), and st_length_int().

Referenced by protocol_enqueue(), and submission_init().

void smtp_invalid (connection_t * con)

A function that is executed when an invalid SMTP command is executed.

Returns:

This function returns no value.

Definition at line 181 of file smtp.c.

References con_write_bl(), server_t::delay, connection_t::protocol, connection_t::server, server_t::violations, and connection_t::violations.

Referenced by smtp_process().

void smtp_mail_from (connection_t * con)

Specify the identity of a message's sender, in response to an SMTP MAIL FROM command.

See also:

smtp_parse_mail_from_path()

Note:

This command must be preceded by a HELO command and successful authentication. Any prior email address specified by a MAIL FROM command will be overwritten. If the SIZE parameter was specified with the MAIL FROM command, its value will be compared to the maximum value specified in the smtp.message_length_limit configuration option.

Parameters:

con the SMTP client connection issuing the command.

Returns:

This function returns no value.

Definition at line 62 of file smtp.c.

References smtp_session_t::authenticated, con_write_bl(), smtp_session_t::helo, magma, smtp_session_t::mailfrom, magma_t::message_length_limit, magma_t::smtp, connection_t::smtp, smtp_parse_mail_from_path(), smtp_session_reset(), st_free(), and smtp_session_t::suggested_length.

void smtp_noop (connection_t * con)

Perform an SMTP NOOP (no-operation) command.

Note:

This command does essentially nothing and is mostly a way to keep connections alive without timing out due to inactivity.

Returns:

This function returns no value.

Definition at line 157 of file smtp.c.

References con_write_bl().

void smtp_quit (connection_t * con)

Gracefully terminate an SMTP session, especially in response to an SMTP QUIT command.

Note:

The standards specify that the receiver **MUST** send an OK reply, and then close the transmission channel.

Parameters:

the SMTP client connection to be terminated.

Returns:

This function returns no value.

Definition at line 196 of file smtp.c.

References con_destroy(), con_status(), and con_write_bl().

Referenced by smtp_data(), smtp_process(), and smtp_requeue().

void smtp_rcpt_to (connection_t * con)

BUG: The code should be able to differentiate between invalid addresses which trigger a NULL return and allocation (or similar) errors which are temporary.

BUG: Detect messages 'from' a local user/domain and tell them to authenticate first.

Definition at line 465 of file smtp.c.

References magma_t::abuse, smtp_inbound_prefs_t::address, magma_t::admin, smtp_session_t::authenticated, smtp_session_t::bypass, smtp_session_t::checked, con_addr(), con_addr_presentation(), con_print(), con_secure(), con_write_bl(), magma_t::contact, credential_alloc_mail(), credential_free(), smtp_inbound_prefs_t::daily_rcv_limit, smtp_inbound_prefs_t::daily_rcv_limit_ip, smtp_outbound_prefs_t::daily_send_limit, smtp_inbound_prefs_t::forwarded, smtp_inbound_prefs_t::greylist, smtp_inbound_prefs_t::greytime, smtp_session_t::helo, lower_st(), magma, credential_t::mail, smtp_session_t::mailfrom, MANAGEDBUF, smtp_session_t::max_length, MEMORYBUF, smtp_session_t::num_recipients, smtp_session_t::out_prefs, smtp_inbound_prefs_t::overquota, smtp_session_t::rbl, smtp_inbound_prefs_t::rbl, smtp_inbound_prefs_t::rblaction, smtp_inbound_prefs_t::rcptto, magma_t::recipient_limit, smtp_inbound_prefs_t::rcv_size_limit, smtp_inbound_prefs_t::rollout, smtp_outbound_prefs_t::send_size_limit, smtp_outbound_prefs_t::sent_today, magma_t::smtp, connection_t::smtp, SMTP_ACTION_REJECT, smtp_add_inbound(), smtp_add_recipient(), smtp_check_duplicate_recipient(), smtp_check_greylist(), smtp_check_rbl(), smtp_check_receive_quota(), smtp_fetch_inbound(), smtp_free_inbound(), smtp_parse_rcpt_to(), smtp_session_t::spf, smtp_inbound_prefs_t::spf, spf_check(), smtp_inbound_prefs_t::spfaction, smtp_outbound_prefs_t::ssl, st_char_get(), st_cmp_ci_eq(), st_free(), st_length_int(), smtp_session_t::suggested_length, and smtp_inbound_prefs_t::usernum.

void smtp_rset (connection_t * con)

Reset the SMTP session, in response to an SMTP RSET command.

Note:

This command clears any sender, recipient, and mail data, along with all buffers and state tables. In other words, return to the state immediately after HELO.

Parameters:

con the SMTP client connection issuing the command.

Returns:

This function returns no value.

Definition at line 220 of file smtp.c.

References `con_write_bl()`, and `smtp_session_reset()`.

void smtp_starttls (connection_t * *con*)

TODO: Review error messages and update them with the appropriate response code.

Initialize a TLS session for an unauthenticated SMTP session.

Parameters:

con the connection of the SMTP endpoint requesting the transport layer security upgrade.

Returns:

This function returns no value.

Definition at line 22 of file smtp.c.

References `connection_t::buffer`, `con_secure()`, `con_write_bl()`, `connection_t::line`, `log_pedantic`, `M_SSL_BIO_NOCLOSE`, `connection_t::network`, `pl_null()`, `connection_t::server`, `smtp_session_reset()`, `connection_t::sockd`, `connection_t::ssl`, `ssl_alloc()`, `st_length_set()`, `stats_increment_by_name()`, and `connection_t::status`.

void submission_init (connection_t * *con*)

Definition at line 1194 of file smtp.c.

References `connection_t::smtp`, `smtp_init()`, and `smtp_session_t::submission`.

Referenced by `protocol_enqueue()`.

magma/servers/smtp/transmit.c File Reference

Handle replies.

```
#include "magma.h"
```

Functions

- **int_t smtp_relay_message** (**connection_t** *con, **stringer_t** **result)
- *Relay an outbound smtp message for a user.* **int_t smtp_forward_message** (**server_t** *server, **stringer_t** *address, **stringer_t** *message, **stringer_t** *id, **int_t** mark, **uint64_t** signum, **uint64_t** sigkey)
- **int_t smtp_bounce** (**connection_t** *con)
- *transmit.c* **int_t smtp_reply** (**stringer_t** *from, **stringer_t** *to, **uint64_t** usernum, **uint64_t** autoreply, **int_t** spf, **int_t** dkim)
- **int_t smtp_send_message** (**stringer_t** *to, **stringer_t** *from, **stringer_t** *message)

Relay an outbound smtp message for the user.

Detailed Description

Handle replies.

Definition in file **transmit.c**.

Function Documentation

int_t smtp_bounce (**connection_t** * con)

transmit.c

Definition at line 179 of file transmit.c.

References smtp_session_t::checked, smtp_session_t::dkim, dkim_create(), magma_t::domain, hash_crc64(), smtp_session_t::in_prefs, log_pedantic, magma, smtp_session_t::mailfrom, MANAGEDBUF, smtp_session_t::message, smtp_inbound_prefs_t::next, number, smtp_inbound_prefs_t::outcome, rand_choices(), smtp_inbound_prefs_t::rcptto, connection_t::smtp, smtp_client_close(), smtp_client_connect(), smtp_client_send_data(), smtp_client_send_helo(), smtp_client_send_nullfrom(), smtp_client_send_rcptto(), SMTP_OUTCOME_BOUNCE_DKIM, SMTP_OUTCOME_BOUNCE_PHISH, SMTP_OUTCOME_BOUNCE_RBL, SMTP_OUTCOME_BOUNCE_SPAM, SMTP_OUTCOME_BOUNCE_SPF, SMTP_OUTCOME_BOUNCE_VIRUS, SMTP_OUTCOME_PERM_FAILURE, SMTP_OUTCOME_SUCESS, SMTP_OUTCOME_TEMP_LOCKED, SMTP_OUTCOME_TEMP_OVERQUOTA, SMTP_OUTCOME_TEMP_SERVER, smtp_session_t::spf, st_char_get(), st_cleanup(), st_cmp_cs_eq(), st_free(), st_length_int(), st_merge, st_sprint(), magma_t::system, and smtp_message_t::text.

Referenced by smtp_data_inbound().

int_t smtp_forward_message (**server_t** * server, **stringer_t** * address, **stringer_t** * message, **stringer_t** * id, **int_t** mark, **uint64_t** signum, **uint64_t** sigkey)

Definition at line 115 of file transmit.c.

References `log_pedantic`, `mail_add_forward_headers()`, `smtp_client_close()`, `smtp_client_connect()`, `smtp_client_send_data()`, `smtp_client_send_helo()`, `smtp_client_send_nullfrom()`, `smtp_client_send_rcptto()`, `st_dupe()`, and `st_free()`.

Referenced by `smtp_accept_message()`.

`int_t smtp_relay_message (connection_t * con, stringer_t ** result)`

Relay an outbound smtp message for a user.

Note:

The following process occurs before the message will be sent: 1. Necessary outbound headers are attached to the message.* 2. An outbound connection to a mail relay server is established (with a premium or normal server pool). 3. Once the connection is negotiated, an RCPT TO command is issued for each of the message's recipients. 4. The mail message data is sent and the client connection is closed.

Parameters:

con a pointer to the connection object across which the outbound mail was attempted to be sent.
result a pointer to the address of a managed string that will receive the server's last response to the mail send attempt, regardless of whether or not it was successful.

Returns:

1 if the message was successfully sent or -1 on failure.

Definition at line 27 of file `transmit.c`.

References `smtp_recipients_t::address`, `CONTIGUOUS`, `HEAP`, `smtp_outbound_prefs_t::importance`, `client_t::line`, `log_pedantic`, `mail_add_outbound_headers()`, `smtp_session_t::mailfrom`, `MANAGED_T`, `smtp_session_t::message`, `smtp_recipients_t::next`, `smtp_session_t::out_prefs`, `smtp_outbound_prefs_t::recipients`, `connection_t::smtp`, `smtp_client_close()`, `smtp_client_connect()`, `smtp_client_send_data()`, `smtp_client_send_helo()`, `smtp_client_send_mailfrom()`, `smtp_client_send_rcptto()`, `st_dupe_opts()`, and `smtp_message_t::text`.

Referenced by `smtp_data_outbound()`.

`int_t smtp_reply (stringer_t * from, stringer_t * to, uint64_t usernum, uint64_t autoreply, int_t spf, int_t dkim)`

Definition at line 363 of file `transmit.c`.

References `cache_get_u64()`, `cache_set_u64()`, `dkim_create()`, `hash_crc64()`, `lock_get()`, `lock_release()`, `log_pedantic`, `MANAGEDBUF`, `rand_choices()`, `smtp_client_close()`, `smtp_client_connect()`, `smtp_client_send_data()`, `smtp_client_send_helo()`, `smtp_client_send_nullfrom()`, `smtp_client_send_rcptto()`, `smtp_fetch_autoreply()`, `st_char_get()`, `st_cleanup()`, `st_free()`, `st_length_int()`, `st_merge`, and `st_sprint()`.

Referenced by `smtp_accept_message()`.

`int_t smtp_send_message (stringer_t * to, stringer_t * from, stringer_t * message)`

Relay an outbound smtp message for the user.

Parameters:

to a managed string containing the name of the mail recipient.
from a managed string containing the address from which the email is being sent.
message a managed string containing the raw body of the mail message.

Returns:

-1 on error or 1 on success.

Definition at line 503 of file transmit.c.

References `log_pedantic`, `smtp_client_close()`, `smtp_client_connect()`, `smtp_client_send_data()`, `smtp_client_send_helo()`, `smtp_client_send_mailfrom()`, and `smtp_client_send_rcptto()`.

Referenced by `contact_business()`, and `register_business_step2()`.

magma/web/contact/abuse.c File Reference

Functions for detecting abuse of the contact form.

```
#include "magma.h"
```

Functions

- void **contact_abuse_increment_history** (**connection_t** *con)
- *Increment the contact abuse history counter for an IP address.* **bool_t** **contact_abuse_checks** (**connection_t** *con, **chr_t** *branch)

Check to see that a client from a given IP address hasn't exceeded its daily quota of contact requests.

Detailed Description

Functions for detecting abuse of the contact form.

Definition in file **abuse.c**.

Function Documentation

bool_t **contact_abuse_checks** (**connection_t** * con, **chr_t** * branch)

Check to see that a client from a given IP address hasn't exceeded its daily quota of contact requests.

abuse.c

Note:

Each IP address will be limited to at most 2 contact requests in any 24-hour period.

Parameters:

con a pointer to the connection object of the remote host making the contact request.

branch a null-terminated string specifying where the contact request was directed ("Abuse" or "Contact").

Returns:

true if the specified connection failed the abuse check or false if it did not.

Definition at line 39 of file abuse.c.

References `cache_get_u64()`, `con_addr_presentation()`, `contact_print_message()`, `MANAGEDBUF`,
`st_char_get()`, `st_length_int()`, and `st_sprint()`.

Referenced by `contact_process()`.

void **contact_abuse_increment_history** (**connection_t** * con)

Increment the contact abuse history counter for an IP address.

Parameters:

con a pointer to the connection object of the remote host making the contact request.

Returns:

This function returns no value.

Definition at line 20 of file abuse.c.

References `cache_increment()`, `con_addr_presentation()`, `MANAGEDBUF`, `st_char_get()`, `st_length_int()`, and `st_sprint()`.

Referenced by `contact_business()`.

magma/web/register/abuse.c File Reference

Functions for handling potential abuse of the new user registration process.

```
#include "magma.h"
```

Functions

- void **register_blocklist_free** (void)
- *Free a registration blocked list.* void **register_blocklist_update** (void)
- *Update the registration blocklist from the database.* bool_t **register_abuse_check_blocklist** (connection_t *con)
- *Check to see if the remote client is on the registration blocklist; if so, increment the web registration blocked counter.* void **register_abuse_increment_history** (connection_t *con)
- *Increment the registration abuse counter for the requesting IP address.* bool_t **register_abuse_checks** (connection_t *con)

Check to see if a registration request is allowed by a remote host; if not, display a banner. Variables

- inx_t * **register_blocklist** = NULL
- pthread_rwlock_t **register_blocklist_lock** = PTHREAD_RWLOCK_INITIALIZER

Detailed Description

Functions for handling potential abuse of the new user registration process.

Definition in file **abuse.c**.

Function Documentation

bool_t register_abuse_check_blocklist (connection_t * con)

Check to see if the remote client is on the registration blocklist; if so, increment the web registration blocked counter.

abuse.c

Parameters:

con the client connection to be checked.

Returns:

false if the check was passed, or true if the connection was made from an IP on the blocklist.

Definition at line 63 of file abuse.c.

References con_addr_standard(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), MANAGEDBUF, register_blocklist_lock, rwlock_lock_read(), rwlock_unlock(), st_char_get(), st_cmp_ci_starts(), st_length_get(), and stats_increment_by_name().

Referenced by register_abuse_checks().

bool_t register_abuse_checks (connection_t * con)

Check to see if a registration request is allowed by a remote host; if not, display a banner.

Parameters:

con the remote connection to be checked.

Returns:

false if registration is allowed or true if not (remote host is on blocklist or registration has been throttled).

Definition at line 117 of file abuse.c.

References `cache_get_u64()`, `con_addr_presentation()`, `con_addr_standard()`, `MANAGEDBUF`, `NULLER`, `register_abuse_check_blocklist()`, `register_print_message()`, and `st_char_get()`.

Referenced by `register_process()`.

void register_abuse_increment_history (connection_t * con)

Increment the registration abuse counter for the requesting IP address.

Parameters:

con a pointer to the connection object of the client making the registration request.

Returns:

This function returns no value.

Definition at line 100 of file abuse.c.

References `cache_increment()`, `con_addr_standard()`, `MANAGEDBUF`, `NULLER`, and `st_char_get()`.

Referenced by `register_business_step2()`.

void register_blocklist_free (void)

Free a registration blocked list.

Returns:

This function returns no value.

Definition at line 23 of file abuse.c.

References `inx_free()`.

void register_blocklist_update (void)

Update the registration blocklist from the database.

Returns:

This function returns no value.

Definition at line 37 of file abuse.c.

References `inx_free()`, `register_blocklist_lock`, `register_data_fetch_blocklist()`, `rwlock_lock_write()`, and `rwlock_unlock()`.

Variable Documentation

inx_t* register_blocklist = NULL

Definition at line 15 of file abuse.c.

pthread_rwlock_t register_blocklist_lock = PTHREAD_RWLOCK_INITIALIZER

Definition at line 16 of file abuse.c.

Referenced by register_abuse_check_blocklist(), and register_blocklist_update().

magma/web/contact/business.c File Reference

Functions for handling the logic of the contact form.

```
#include "magma.h"
```

Functions

- **http_page_t * contact_business_add_error** (**chr_t** *branch, **uchr_t** *xpath, **uchr_t** *id, **uchr_t** *message)
- *Return the contact/abuse page with a marked error indicator for the user in the event of a user submission error.* **bool_t contact_business_valid_email** (**stringer_t** *email)
- *Validate an email address.* **void contact_business** (**connection_t** *con, **chr_t** *branch)

Send the contents of a user-submitted contact or abuse form to the magma-configured contact email address.

Detailed Description

Functions for handling the logic of the contact form.

Definition in file **business.c**.

Function Documentation

void contact_business (**connection_t** * con, **chr_t** * branch)

Send the contents of a user-submitted contact or abuse form to the magma-configured contact email address.

business.c

Parameters:

con the connection of the web client making the request.

branch a null-terminated string containing the destination of the contact form: either "Abuse" or "Contact".

Returns:

This function returns no value.

Definition at line 132 of file business.c.

References magma_t::abuse, magma_t::admin, con_addr_presentation(), magma_t::contact, contact_abuse_increment_history(), contact_business_add_error(), contact_business_valid_email(), contact_print_form(), contact_print_message(), http_data_get(), HTTP_DATA_POST, log_error, magma, MANAGEDBUF, NULLER, PLACER, smtp_send_message(), st_cmp_cs_eq(), st_free(), st_merge, st_replace(), and http_data_t::value.

Referenced by contact_process().

http_page_t* contact_business_add_error (**chr_t** * branch, **uchr_t** * xpath, **uchr_t** * id, **uchr_t** * message)

Return the contact/abuse page with a marked error indicator for the user in the event of a user submission error.

Parameters:

branch a null-terminated string containing the source of the error: "Abuse" or "Contact"

xpath a null-terminated string containing the xpath of the element in the returned page that should be colored red.

id a null-terminated string containing the id of the error text element to be added to the document.

message a null-terminated string containing the actual error message text to be displayed to the user.

Returns:

NULL on failure, or a pointer to the processed contact page with error message on success.

Definition at line 23 of file business.c.

References `http_page_get()`, `NULLER`, `PLACER`, `st_cmp_cs_eq()`, `xml_node_add_sibling()`, `xml_node_free()`, `xml_node_new()`, `xml_node_set_content()`, `xml_node_set_property()`, `xml_xpath_eval()`, and `http_page_t::xpath_ctx`.

Referenced by `contact_business()`.

bool_t contact_business_valid_email (stringer_t * *email*)

Validate an email address.

Parameters:

email a managed string containing the email address to be validated.

Returns:

true if the specified email was valid and false if it was not.

Definition at line 65 of file business.c.

References `length`, `lower_st()`, `st_char_get()`, and `st_length_get()`.

Referenced by `config_validate_settings()`, and `contact_business()`.

magma/web/register/business.c File Reference

Functions for handling validation for the registration process.

```
#include "magma.h"
```

Functions

- **bool_t register_business_validate_password** (stringer_t *password)
- *Determine whether a registered password is valid.* **int_t register_business_validate_username** (stringer_t *username)
- *Determine whether a registered username is valid.* **chr_t * register_business_step1** (connection_t *con, register_session_t *reg)
- *Perform verification checking on all step 1 completed user fields.* **chr_t * register_business_step2** (connection_t *con, register_session_t *reg)

Perform verification checking on all step 2 completed user fields, and display a welcome banner on success.

Detailed Description

Functions for handling validation for the registration process.

Definition in file **business.c**.

Function Documentation

chr_t* register_business_step1 (connection_t * con, register_session_t * reg)

Perform verification checking on all step 1 completed user fields.

business.c

Note:

Checks include captcha verification, username validation, and password reentry verification and validation.

Parameters:

con the underlying client connection.

reg the underlying registration session.

Returns:

NULL on success, or a descriptive error string on failure.

Definition at line 111 of file business.c.

References data, http_data_get(), HTTP_DATA_POST, register_session_t::hvf_input, register_session_t::hvf_value, log_pedantic, register_session_t::password, register_business_validate_password(), register_business_validate_username(), register_data_check_username(), REGISTER_PASSWORD_MAX_LENGTH, REGISTER_PASSWORD_MIN_LENGTH, st_char_get(), st_cmp_ci_eq(), st_dupe(), st_length_int(), register_session_t::username, and http_data_t::value.

Referenced by register_process().

chr_t* register_business_step2 (connection_t * con, register_session_t * reg)

Perform verification checking on all step 2 completed user fields, and display a welcome banner on success.

Note:

Checks include plan type validation, and billing information processing. After step 2, the user's supplied information will be persisted into the database.

Parameters:

con the underlying client connection.
reg the underlying registration session.

Returns:

NULL on success, or a descriptive error string on failure.

Definition at line 179 of file business.c.

References magma_t::admin, magma_t::contact, data, magma_t::domain, http_data_get(), HTTP_DATA_POST, log_pedantic, magma, PLACER, register_session_t::plan, register_abuse_increment_history(), register_data_insert_user(), smtp_send_message(), st_cleanup(), st_cmp_cs_eq(), st_dupe(), st_merge, magma_t::system, tran_commit(), tran_rollback(), tran_start(), register_session_t::username, register_session_t::username, and http_data_t::value.

Referenced by register_process().

bool_t register_business_validate_password (stringer_t * password)

Determine whether a registered password is valid.

Note:

Each password must be between REGISTER_PASSWORD_MIN_LENGTH (5) and REGISTER_PASSWORD_MAX_LENGTH (200) characters long.

Parameters:

password the user's password to be evaluated.

Returns:

false on failure (too long, too short, or bad characters) or true on success.

Definition at line 21 of file business.c.

References length, REGISTER_PASSWORD_MAX_LENGTH, st_char_get(), and st_length_get().

Referenced by register_business_step1().

int_t register_business_validate_username (stringer_t * username)

Determine whether a registered username is valid.

Parameters:

username a managed string containing the proposed username to be evaluated.

Returns:

-1 or 0 on failure (too long, too short, or bad characters) or 1 on success.

Definition at line 53 of file business.c.

References length, REGISTER_USERNAME_MAX_LENGTH, st_char_get(), and st_length_get().

Referenced by register_business_step1().

magma/web/contact/contact.c File Reference

Handle the contact form.

```
#include "magma.h"
```

Functions

- void **contact_print_message** (**connection_t** *con, **chr_t** *branch, **chr_t** *message)
 - *Display the contact or abuse notification form to the requesting user.* void **contact_print_form** (**connection_t** *con, **chr_t** *branch, **http_page_t** *page)
 - *Display the contact or abuse form to the user.* void **contact_process** (**connection_t** *con, **chr_t** *branch)
- Process all user contact requests.*
-

Detailed Description

Handle the contact form.

Definition in file **contact.c**.

Function Documentation

void contact_print_form (connection_t * con, chr_t * branch, http_page_t * page)

Display the contact or abuse form to the user.

contact.c

Parameters:

con the connection across which the form will be returned.

branch a null-terminated string specifying the type of contact, either "Contact" or "Abuse".

page a pointer to an http page containing the contact document to be populated with the user's name, email, and message.

Returns:

This function returns no value.

Definition at line 76 of file contact.c.

References `con_write_st()`, `contact_print_message()`, `http_page_t::content`, `data`, `http_page_t::doc_obj`, `http_data_get()`, `HTTP_DATA_POST`, `http_page_free()`, `http_page_get()`, `http_response_header()`, `mm_cleanup()`, `NULLER`, `PLACER`, `st_char_get()`, `st_cmp_cs_eq()`, `st_free()`, `st_length_get()`, `http_content_t::type`, `http_data_t::value`, `xml_dump_doc()`, `xml_encode()`, `xml_set_xpath_ns()`, `xml_set_xpath_property()`, and `http_page_t::xpath_ctx`.

Referenced by `contact_business()`, and `contact_process()`.

void contact_print_message (connection_t * con, chr_t * branch, chr_t * message)

Display the contact or abuse notification form to the requesting user.

Note:

Both the contact and abuse forms operate on the underlying template found in "contact/message"

Parameters:

con a pointer to the connection object across which the response will be sent.

branch a null-terminated string specifying the type of contact, either "Contact" or "Abuse".

message a null-terminated string pointing to a custom message to be displayed to the user.

Returns:

This function returns no value.

Definition at line 23 of file contact.c.

References `con_write_st()`, `http_page_t::content`, `http_page_t::doc_obj`, `http_page_free()`, `http_page_get()`, `http_print_500()`, `http_print_500_log()`, `http_response_header()`, `NULLER`, `PLACER`, `st_cmp_cs_eq()`, `st_free()`, `st_length_get()`, `http_content_t::type`, `xml_dump_doc()`, `xml_set_xpath_ns()`, `xml_set_xpath_property()`, and `http_page_t::xpath_ctx`.

Referenced by `contact_abuse_checks()`, `contact_business()`, and `contact_print_form()`.

void contact_process (connection_t * *con*, chr_t * *branch*)

Process all user contact requests.

Parameters:

con a pointer to the connection object generating the contact request.

branch a null-terminated string specifying the request type (can be "Abuse" or "Contact").

Definition at line 127 of file contact.c.

References `con_secure()`, `contact_abuse_checks()`, `contact_business()`, `contact_print_form()`, `connection_t::http`, `http_data_get()`, `HTTP_DATA_POST`, `HTTP_PARSE_PAIRS`, `http_print_301()`, `NULLER`, `PLACER`, and `st_cmp_cs_eq()`.

Referenced by `http_response()`.

magma/web/contact/contact.h File Reference

Definitions for handling the web contact form.

Functions

- **bool_t** **contact_abuse_checks** (**connection_t** *con, **chr_t** *branch)
- *abuse.c* void **contact_abuse_increment_history** (**connection_t** *con)
- *Increment the contact abuse history counter for an IP address.* void **contact_business** (**connection_t** *con, **chr_t** *branch)
- *business.c* **http_page_t** * **contact_business_add_error** (**chr_t** *branch, **uchr_t** *xpath, **uchr_t** *id, **uchr_t** *message)
- *Return the contact/abuse page with a marked error indicator for the user in the event of a user submission error.* **bool_t** **contact_business_valid_email** (**stringer_t** *email)
- *Validate an email address.* void **contact_print_form** (**connection_t** *con, **chr_t** *branch, **http_page_t** *page)
- *contact.c* void **contact_print_message** (**connection_t** *con, **chr_t** *branch, **chr_t** *message)
- *Display the contact or abuse notification form to the requesting user.* void **contact_process** (**connection_t** *con, **chr_t** *branch)

Process all user contact requests.

Detailed Description

Definitions for handling the web contact form.

Definition in file **contact.h**.

Function Documentation

bool_t **contact_abuse_checks** (**connection_t** * con, **chr_t** * branch)

abuse.c

abuse.c

Note:

Each IP address will be limited to at most 2 contact requests in any 24-hour period.

Parameters:

con a pointer to the connection object of the remote host making the contact request.

branch a null-terminated string specifying where the contact request was directed ("Abuse" or "Contact").

Returns:

true if the specified connection failed the abuse check or false if it did not.

Definition at line 39 of file *abuse.c*.

References *cache_get_u64()*, *con_addr_presentation()*, *contact_print_message()*, *MANAGEDBUF*, *st_char_get()*, *st_length_int()*, and *st_sprint()*.

Referenced by *contact_process()*.

void **contact_abuse_increment_history** (**connection_t** * con)

Increment the contact abuse history counter for an IP address.

Parameters:

con a pointer to the connection object of the remote host making the contact request.

Returns:

This function returns no value.

Definition at line 20 of file abuse.c.

References `cache_increment()`, `con_addr_presentation()`, `MANAGEDBUF`, `st_char_get()`, `st_length_int()`, and `st_sprint()`.

Referenced by `contact_business()`.

void contact_business (connection_t * *con*, chr_t * *branch*)

business.c

business.c

Parameters:

con the connection of the web client making the request.

branch a null-terminated string containing the destination of the contact form: either "Abuse" or "Contact".

Returns:

This function returns no value.

Definition at line 132 of file business.c.

References `magma_t::abuse`, `magma_t::admin`, `con_addr_presentation()`, `magma_t::contact`, `contact_abuse_increment_history()`, `contact_business_add_error()`, `contact_business_valid_email()`, `contact_print_form()`, `contact_print_message()`, `http_data_get()`, `HTTP_DATA_POST`, `log_error`, `magma`, `MANAGEDBUF`, `NULLER`, `PLACER`, `smtp_send_message()`, `st_cmp_cs_eq()`, `st_free()`, `st_merge`, `st_replace()`, and `http_data_t::value`.

Referenced by `contact_process()`.

http_page_t* contact_business_add_error (chr_t * *branch*, uchr_t * *xpath*, uchr_t * *id*, uchr_t * *message*)

Return the contact/abuse page with a marked error indicator for the user in the event of a user submission error.

Parameters:

branch a null-terminated string containing the source of the error: "Abuse" or "Contact"

xpath a null-terminated string containing the xpath of the element in the returned page that should be colored red.

id a null-terminated string containing the id of the error text element to be added to the document.

message a null-terminated string containing the actual error message text to be displayed to the user.

Returns:

NULL on failure, or a pointer to the processed contact page with error message on success.

Definition at line 23 of file business.c.

References `http_page_get()`, `NULLER`, `PLACER`, `st_cmp_cs_eq()`, `xml_node_add_sibling()`, `xml_node_free()`, `xml_node_new()`, `xml_node_set_content()`, `xml_node_set_property()`, `xml_xpath_eval()`, and `http_page_t::xpath_ctx`.

Referenced by `contact_business()`.

bool_t contact_business_valid_email (stringer_t * email)

Validate an email address.

Parameters:

email a managed string containing the email address to be validated.

Returns:

true if the specified email was valid and false if it was not.

Definition at line 65 of file business.c.

References `length`, `lower_st()`, `st_char_get()`, and `st_length_get()`.

Referenced by `config_validate_settings()`, and `contact_business()`.

void contact_print_form (connection_t * con, chr_t * branch, http_page_t * page)

contact.c

contact.c

Parameters:

con the connection across which the form will be returned.

branch a null-terminated string specifying the type of contact, either "Contact" or "Abuse".

page a pointer to an http page containing the contact document to be populated with the user's name, email, and message.

Returns:

This function returns no value.

Definition at line 76 of file contact.c.

References `con_write_st()`, `contact_print_message()`, `http_page_t::content`, `data`, `http_page_t::doc_obj`, `http_data_get()`, `HTTP_DATA_POST`, `http_page_free()`, `http_page_get()`, `http_response_header()`, `mm_cleanup()`, `NULLER`, `PLACER`, `st_char_get()`, `st_cmp_cs_eq()`, `st_free()`, `st_length_get()`, `http_content_t::type`, `http_data_t::value`, `xml_dump_doc()`, `xml_encode()`, `xml_set_xpath_ns()`, `xml_set_xpath_property()`, and `http_page_t::xpath_ctx`.

Referenced by `contact_business()`, and `contact_process()`.

void contact_print_message (connection_t * con, chr_t * branch, chr_t * message)

Display the contact or abuse notification form to the requesting user.

Note:

Both the contact and abuse forms operate on the underlying template found in "contact/message"

Parameters:

con a pointer to the connection object across which the response will be sent.

branch a null-terminated string specifying the type of contact, either "Contact" or "Abuse".

message a null-terminated string pointing to a custom message to be displayed to the user.

Returns:

This function returns no value.

Definition at line 23 of file contact.c.

References `con_write_st()`, `http_page_t::content`, `http_page_t::doc_obj`, `http_page_free()`, `http_page_get()`, `http_print_500()`, `http_print_500_log()`, `http_response_header()`, `NULLER`, `PLACER`, `st_cmp_cs_eq()`, `st_free()`, `st_length_get()`, `http_content_t::type`, `xml_dump_doc()`, `xml_set_xpath_ns()`, `xml_set_xpath_property()`, and `http_page_t::xpath_ctx`.

Referenced by `contact_abuse_checks()`, `contact_business()`, and `contact_print_form()`.

`void contact_process (connection_t * con, chr_t * branch)`

Process all user contact requests.

Parameters:

con a pointer to the connection object generating the contact request.

branch a null-terminated string specifying the request type (can be "Abuse" or "Contact").

Definition at line 127 of file `contact.c`.

References `con_secure()`, `contact_abuse_checks()`, `contact_business()`, `contact_print_form()`, `connection_t::http`, `http_data_get()`, `HTTP_DATA_POST`, `HTTP_PARSE_PAIRS`, `http_print_301()`, `NULLER`, `PLACER`, and `st_cmp_cs_eq()`.

Referenced by `http_response()`.

magma/web/portal/endpoint.c File Reference

The control logic for the Portal JSON endpoint.

```
#include "magma.h"
#include "methods.h"
```

Functions

- void **portal_endpoint_sort** (void)
- *Sort the web portal JSON command table to be ready for binary searches.* **int_t portal_endpoint_compare** (const void *compare, const void *command)
- *Internal bsearch() comparison function for dispatching the proper portal handler.* **bool_t portal_validate_request** (**connection_t** *con, int err_mask, **chr_t** *method, **bool_t** has_params, size_t nparams)
- *Verify that a session underlying a portal request has been authenticated.* void **portal_endpoint_error** (**connection_t** *con, **int_t** http_code, **int_t** error_code, **chr_t** *message)
- *Return a json-rpc error response to the remote client.* void **portal_endpoint_response** (**connection_t** *con, **chr_t** *format,...)
- *Generate a json-rpc 2.0 response to a portal request.* void **portal_endpoint_auth** (**connection_t** *con)
- *Obtain credentials and log a user into portal session in response to an "auth" json-rpc portal request.* void **portal_endpoint_logout** (**connection_t** *con)
- *Log a user out of a portal session in response to a "logout" json-rpc portal request.* void **portal_endpoint_alert_list** (**connection_t** *con)
- *Get the list of a user's alert messages in response to an "alert.list" json-rpc portal request.* void **portal_endpoint_alert_acknowledge** (**connection_t** *con)
- *Process user alert message acknowledgements in response to an "alert.acknowledge" json-rpc portal request.* void **portal_endpoint_aliases** (**connection_t** *con)
- *Return the list of a user account's aliases in response to an "aliases" json-rpc portal request.* void **portal_endpoint_cookies** (**connection_t** *con)
- *Return whether or not a user is using cookies, in response to a "cookies" json-rpc portal request.* void **portal_endpoint_folders_add** (**connection_t** *con)
- *Create a new folder in response to a "folders.add" json-rpc portal request.* void **portal_endpoint_folders_list** (**connection_t** *con)
- *Return a list of a user's folders in response to a "folders.list" json-rpc portal request.* void **portal_endpoint_folders_remove** (**connection_t** *con)
- *Remove a user's folder in response to a json-rpc "folders.remove" portal request.* void **portal_endpoint_folders_rename** (**connection_t** *con)
- *Rename a user's folder in response to a json-rpc "folders.rename" portal request.* void **portal_endpoint_folders_tags** (**connection_t** *con)
- void **portal_endpoint_messages_list** (**connection_t** *con)
- *Retrieve a list of the user's messages in response to a json-rpc "messages.list" portal request.* void **portal_endpoint_messages_copy** (**connection_t** *con)
- void **portal_endpoint_messages_flag** (**connection_t** *con)
- void **portal_endpoint_messages_move** (**connection_t** *con)
- void **portal_endpoint_messages_remove** (**connection_t** *con)
- *Remove a user's mail message in response to a "messages.remove" json-rpc portal request.* void **portal_endpoint_messages_tags** (**connection_t** *con)
- *Get all of the message tags used by a specified user in response to a "messages.tags" json-rpc portal request.* void **portal_endpoint_messages_tag** (**connection_t** *con)
- *Perform a tag operation on a user's message, in response to a "messages.tag" json-rpc portal request.* void **portal_endpoint_messages_load** (**connection_t** *con)
- void **portal_endpoint_messages_compose** (**connection_t** *con)

- Compose a new message in response to a "messages.compose" json-rpc portal request. void **portal_endpoint_attachments_add** (connection_t *con)
- Add an attachment to a message being composed in response to an "attachments.add" json-rpc portal request. void **portal_endpoint_attachments_remove** (connection_t *con)
- Remove an attachment from a message being composed in response to an "attachments.remove" json-rpc portal request. void **portal_endpoint_messages_send** (connection_t *con)
- Send a composed message in response to a "messages.send" json-rpc portal request. void **portal_endpoint_contacts_copy** (connection_t *con)
- void **portal_endpoint_contacts_move** (connection_t *con)
- Move a user's contact entry to another contacts folder in response to a "contacts.move" json-rpc portal request. void **portal_endpoint_contacts_remove** (connection_t *con)
- Remove a user's contact entry in response to a "contacts.remove" json-rpc portal request. void **portal_endpoint_contacts_list** (connection_t *con)
- List a user's contacts in response to a "contacts.list" json-rpc portal request. void **portal_endpoint_contacts_add** (connection_t *con)
- Add a contact in response to a "contacts.add" json-rpc portal request. void **portal_endpoint_contacts_edit** (connection_t *con)
- void **portal_endpoint_config_edit** (connection_t *con)
- Retrieve all of a user's config options and return them to the remote client in response to a portal "contacts.load" json-rpc request. void **portal_endpoint_config_load** (connection_t *con)
- Retrieve all of a user's config options and return them to the remote client in response to a portal "contacts.load" json-rpc request. void **portal_endpoint_contacts_load** (connection_t *con)
- Return information for a portal "contacts.load" json-rpc request. void **portal_endpoint_ad** (connection_t *con)
- Return advertising information for a portal "ad" json-rpc request. void **portal_endpoint_search** (connection_t *con)
- void **portal_endpoint_scrape_add** (connection_t *con)
- void **portal_endpoint_scrape** (connection_t *con)
- void **portal_endpoint_attachments_progress** (connection_t *con)
- void **portal_settings_identity** (connection_t *con)
- Return information for a portal "settings.identity" json-rpc request. void **portal_meta** (connection_t *con)
- Return information for a portal "meta" json-rpc request. void **portal_settings_changepass** (connection_t *con)
- Change a user's password in response to a portal "settings.changepass" json-rpc request. void **portal_endpoint** (connection_t *con)
- The entry point for camel requests sent to the portal. **attachment_t * portal_get_upload_attachment** (connection_t *con)
- Get a user's attachment to a message composition uploaded via multipart form data. void **portal_upload** (connection_t *con)
- Process uploaded attachments for messages composed in conjunction with the portal interface. void **portal_debug** (connection_t *con)

A portal debug function that will be disabled and/or deleted completely in production.

Detailed Description

The control logic for the Portal JSON endpoint.

Definition in file **endpoint.c**.

Function Documentation

void portal_debug (connection_t * con)

A portal debug function that will be disabled and/or deleted completely in production.

Note:

The debug output is displayed LOCALLY in magmad's console.

Definition at line 3310 of file endpoint.c.

void portal_endpoint (connection_t * con)

The entry point for camel requests sent to the portal.

endpoint.c

Parameters:

con the connection object corresponding to the web client making the request.

Returns:

This function returns no value.

Definition at line 2958 of file endpoint.c.

References `command`, `con_addr_word()`, `con_secure()`, `command_t::function`, `connection_t::http`, `HTTP_ERROR_500`, `HTTP_MERGED`, `http_parse_context()`, `HTTP_PORTAL`, `http_print_301()`, `http_response_header()`, `json_integer_value_d`, `json_loads_d`, `json_object_get_d`, `JSON_RPC_2_ERROR_PARSE_MALFORMED`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_UNAVAIL`, `JSON_RPC_2_ERROR_SERVER_REQUEST`, `json_string_value_d`, `json_type_string_d`, `command_t::length`, `log_pedantic`, `magma`, `ns_length_get()`, `PLACER`, `magma_t::portal`, `portal_endpoint_compare()`, `portal_endpoint_error()`, `portal_methods`, `sess_create()`, `st_char_get()`, `command_t::string`, `uint64_conv_ns()`, and `magma_t::web`.

Referenced by `http_response()`.

void portal_endpoint_ad (connection_t * con)

Return advertising information for a portal "ad" json-rpc request.

^ finished / work area v ^ work area / stubs v

Note:

This function is not implemented and may be removed entirely in the future.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2763 of file endpoint.c.

References `JSON_RPC_2_ERROR_SERVER_METHOD_UNAVAIL`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_AD`, and `portal_validate_request()`.

void portal_endpoint_alert_acknowledge (connection_t * con)

Process user alert message acknowledgements in response to an "alert.acknowledge" json-rpc portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 360 of file endpoint.c.

References `count`, `connection_t::http`, `json_array_get_d`, `json_array_size_d`, `json_integer_value_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_pedantic`, `meta_data_acknowledge_alert()`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_ALERT_ACKNOWLEDGE`, `portal_endpoint_response()`, `portal_validate_request()`, `st_char_get()`, `st_length_get()`, `tran_commit()`, `tran_rollback()`, and `tran_start()`.

void portal_endpoint_alert_list (connection_t * con)

Get the list of a user's alert messages in response to an "alert.list" json-rpc portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 312 of file endpoint.c.

References `connection_t::http`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_free()`, `json_array_append_new_d`, `json_array_d`, `json_decref_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `log_pedantic`, `meta_data_fetch_alerts()`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_ALERT_LIST`, `portal_endpoint_response()`, `portal_validate_request()`, and `st_char_get()`.

void portal_endpoint_aliases (connection_t * con)

Return the list of a user account's aliases in response to an "aliases" json-rpc portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 429 of file endpoint.c.

References `connection_t::http`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `json_array_append_new_d`, `json_array_d`, `json_decref_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `json_true_d`, `log_pedantic`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_ALIASES`, `portal_endpoint_response()`, `portal_validate_request()`, and `st_char_get()`.

void portal_endpoint_attachments_add (connection_t * con)

Add an attachment to a message being composed in response to an "attachments.add" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 1878 of file endpoint.c.

References attachment_t::attach_id, composition_t::attached, composition_t::attachments, attachment_t::filename, connection_t::http, inx_find(), inx_insert(), json_object_d, json_pack_ex_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_error, log_pedantic, M_TYPE_UINT64, mm_alloc(), mutex_lock(), mutex_unlock(), ns_length_get(), portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_ATTACHMENTS_ADD, PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), portal_validate_request(), sess_release_attachment(), st_char_get(), st_import(), st_length_get(), multi_t::u64, and multi_t::val.

void portal_endpoint_attachments_progress (connection_t * *con*)

Definition at line 2823 of file endpoint.c.

References con_write_st(), connection_t::http, HTTP_ERROR_500, http_response_header(), PLACER, PORTAL_ENDPOINT_ERROR_ATTACHMENTS_PROGRESS, st_aprint(), st_free(), and st_length_get().

void portal_endpoint_attachments_remove (connection_t * *con*)

Remove an attachment from a message being composed in response to an "attachments.remove" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 1956 of file endpoint.c.

References composition_t::attachments, connection_t::http, inx_delete(), inx_find(), JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_error, log_pedantic, M_TYPE_UINT64, mutex_lock(), mutex_unlock(), portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_ATTACHMENTS_REMOVE, PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), portal_validate_request(), st_char_get(), st_length_get(), multi_t::u64, and multi_t::val.

void portal_endpoint_auth (connection_t * *con*)

Obtain credentials and log a user into portal session in response to an "auth" json-rpc portal request.

Parameters:

con a pointer to the connection object of the user attempting to log in.

Returns:

This function returns no value.

Definition at line 200 of file endpoint.c.

References `credential_t::auth`, `con_secure()`, `credential_alloc_auth()`, `meta_user_t::flags`, `connection_t::http`, `HTTP_COOKIE_SET`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `meta_user_t::lock_status`, `log_pedantic`, `meta_get()`, `META_GET_CONTACTS`, `META_GET_FOLDERS`, `META_GET_MESSAGES`, `META_PROT_WEB`, `meta_remove()`, `meta_user_ref_add()`, `META_USER_SSL`, `NULLER`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_AUTH`, `PORTAL_ENDPOINT_ERROR_MODE`, `portal_endpoint_response()`, `SESSION_STATE_AUTHENTICATED`, `SESSION_STATE_NEUTRAL`, `st_char_get()`, and `meta_user_t::username`.

int_t portal_endpoint_compare (const void * *compare*, const void * *command*)

Internal bsearch() comparison function for dispatching the proper portal handler.

Parameters:

compare a pointer to a **command_t** object with a portal method name for string comparison.

command a pointer to a **command_t** object with a portal method name for string comparison.

Returns:

Definition at line 42 of file `endpoint.c`.

References `command_t::length`, `PLACER`, `st_cmp_ci_eq()`, and `command_t::string`.

Referenced by `portal_endpoint()`, and `portal_endpoint_sort()`.

void portal_endpoint_config_edit (connection_t * *con*)

Retrieve all of a user's config options and return them to the remote client in response to a portal "contacts.load" json-rpc request.

Note:

Requires a user to be authenticated; failure will be communicated to the client via a json response.

Parameters:

con a pointer to a connection object over which the json response will be sent.

Returns:

This function returns no value.

HIGH: We aren't checking/validating the config entries! And we should probably do our own duplication checks to avoid placing unnecessary load on the database.

Definition at line 2614 of file `endpoint.c`.

References `content`, `connection_t::http`, `json_object_iter_d`, `json_object_iter_key_d`, `json_object_iter_next_d`, `json_object_iter_value_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `json_string_value_d`, `NULLER`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONFIG_EDIT`, `portal_endpoint_response()`, `portal_validate_request()`, `status`, `user_config_create()`, `user_config_edit()`, `user_config_free()`, and `user_config_t`.

void portal_endpoint_config_load (connection_t * *con*)

Retrieve all of a user's config options and return them to the remote client in response to a portal "contacts.load" json-rpc request.

Note:

Requires a user to be authenticated; failure will be communicated to the client via a json response.

Parameters:

con a pointer to a connection object over which the json response will be sent.

Returns:

This function returns no value.

Definition at line 2673 of file endpoint.c.

References `connection_t::http`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `portal_config_collection()`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONFIG_LOAD`, `portal_endpoint_response()`, `portal_validate_request()`, `user_config_create()`, `user_config_free()`, and `user_config_t`.

void portal_endpoint_contacts_add (connection_t * con)

Add a contact in response to a "contacts.add" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

HIGH: We aren't checking/validating the contact details! And we should probably do our own duplication checks to avoid placing unnecessary load on the database.

Definition at line 2422 of file endpoint.c.

References `contact_create()`, `contact_edit()`, `contact_free()`, `contact_t`, `contact_validate_detail()`, `contact_validate_name()`, `content`, `connection_t::http`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_insert()`, `json_object_get_d`, `json_object_iter_d`, `json_object_iter_key_d`, `json_object_iter_next_d`, `json_object_iter_value_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_string_value_d`, `json_type_string_d`, `json_unpack_ex_d`, `log_pedantic`, `M_TYPE_UINT64`, `magma_folder_find_number()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_user_wlock()`, `NULLER`, `OBJECT_CONTACTS`, `PLACER`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONSTRAINT_VIOLATION`, `PORTAL_ENDPOINT_ERROR_CONTACTS_ADD`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `portal_validate_request()`, `sess_serial_check()`, `st_char_get()`, `st_cmp_ci_eq()`, `st_cmp_cs_eq()`, `st_length_get()`, `status`, `multi_t::u64`, and `multi_t::val`.

void portal_endpoint_contacts_copy (connection_t * con)

Note:

If the source and target folder are equal, a copy will be made with a different name.

Definition at line 2128 of file endpoint.c.

References `contact_create()`, `contact_detail_t`, `contact_edit()`, `contact_find_number()`, `contact_free()`, `contact_t`, `connection_t::http`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_insert()`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_error`, `log_pedantic`, `M_TYPE_UINT64`, `magma_folder_find_number()`,

meta_user_rlock(), meta_user_unlock(), meta_user_wlock(), OBJECT_CONTACTS, PLACER, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_CONSTRAINT_VIOLATION, PORTAL_ENDPOINT_ERROR_CONTACTS_COPY, PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), portal_validate_request(), sess_serial_check(), st_char_get(), st_cleanup(), st_length_get(), st_merge, multi_t::u64, and multi_t::val.

void portal_endpoint_contacts_edit (connection_t * con)

HIGH: We aren't checking/validating the contact details! And we should probably do our own duplication checks to avoid placing unnecessary load on the database.

Definition at line 2538 of file endpoint.c.

References contact_edit(), contact_find_number(), contact_t, content, connection_t::http, json_object_iter_d, json_object_iter_key_d, json_object_iter_next_d, json_object_iter_value_d, json_object_size_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_string_value_d, json_unpack_ex_d, log_pedantic, magma_folder_find_number(), meta_user_rlock(), meta_user_unlock(), meta_user_wlock(), NULLER, OBJECT_CONTACTS, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_CONTACTS_EDIT, PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION, PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), portal_validate_request(), sess_serial_check(), st_char_get(), st_length_get(), and status.

void portal_endpoint_contacts_list (connection_t * con)

List a user's contacts in response to a "contacts.list" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2353 of file endpoint.c.

References contact_detail_t, contact_t, connection_t::http, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), json_array_append_new_d, json_array_d, json_decref_d, json_object_set_new_d, json_pack_ex_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_string_d, json_unpack_ex_d, log_pedantic, magma_folder_find_number(), portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_CONTACTS_LIST, PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), portal_validate_request(), st_char_get(), and st_length_get().

void portal_endpoint_contacts_load (connection_t * con)

Return information for a portal "contacts.load" json-rpc request.

Note:

This function returns the all the stored details about a specified contact entry.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2709 of file endpoint.c.

References `contact_t`, `connection_t::http`, `inx_find()`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_pedantic`, `M_TYPE_UINT64`, `magma_folder_find_number()`, `portal_contact_details()`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONTACTS_LOAD`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `portal_validate_request()`, `st_char_get()`, `st_length_get()`, `multi_t::u64`, and `multi_t::val`.

void portal_endpoint_contacts_move (connection_t * con)

Move a user's contact entry to another contacts folder in response to a "contacts.move" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2235 of file endpoint.c.

References `contact_find_number()`, `contact_move()`, `contact_t`, `connection_t::http`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_pedantic`, `magma_folder_find_number()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_CONTACTS`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONTACTS_MOVE`, `PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `portal_validate_request()`, `sess_serial_check()`, `st_char_get()`, `st_length_get()`, and `status`.

void portal_endpoint_contacts_remove (connection_t * con)

Remove a user's contact entry in response to a "contacts.remove" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2298 of file endpoint.c.

References `contact_delete()`, `contact_find_number()`, `contact_t`, `connection_t::http`, `inx_delete()`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_pedantic`, `M_TYPE_UINT64`, `magma_folder_find_number()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_CONTACTS`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONTACTS_REMOVE`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `portal_validate_request()`, `sess_serial_check()`, `st_char_get()`, `st_length_get()`, `status`, `multi_t::u64`, and `multi_t::val`.

void portal_endpoint_cookies (connection_t * con)

Return whether or not a user is using cookies, in response to a "cookies" json-rpc portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 479 of file endpoint.c.

References `connection_t::http`, `PLACER`, `PORTAL_ENDPOINT_ERROR_COOKIES`, `portal_endpoint_response()`, `portal_validate_request()`, and `st_cmp_ci_starts()`.

void portal_endpoint_error (connection_t * *con*, int_t *http_code*, int_t *error_code*, chr_t * *message*)

Return a json-rpc error response to the remote client.

Parameters:

con a pointer to the connection object across which the response will be sent.

http_code the http response code to be sent to the remote client in the response header.

error_code the numerical error code to be encoded in the json-rpc error message.

message a descriptive error string to be encoded in the json-rpc error message.

Returns:

This function returns no value.

Definition at line 130 of file endpoint.c.

References `con_write_st()`, `connection_t::http`, `HTTP_ERROR_500`, `http_response_header()`, `json_decref_d`, `json_dumps_d`, `json_pack_ex_d`, `log_options`, `log_pedantic`, `M_LOG_CRITICAL`, `M_LOG_STACK_TRACE`, `magma`, `ns_append()`, `ns_free()`, `ns_length_get()`, `NULLER`, `PLACER`, `magma_t::portal`, and `magma_t::web`.

Referenced by `portal_endpoint()`, `portal_endpoint_ad()`, `portal_endpoint_alert_acknowledge()`, `portal_endpoint_alert_list()`, `portal_endpoint_aliases()`, `portal_endpoint_attachments_add()`, `portal_endpoint_attachments_remove()`, `portal_endpoint_auth()`, `portal_endpoint_config_edit()`, `portal_endpoint_config_load()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_contacts_edit()`, `portal_endpoint_contacts_list()`, `portal_endpoint_contacts_load()`, `portal_endpoint_contacts_move()`, `portal_endpoint_contacts_remove()`, `portal_endpoint_folders_add()`, `portal_endpoint_folders_list()`, `portal_endpoint_folders_remove()`, `portal_endpoint_folders_rename()`, `portal_endpoint_folders_tags()`, `portal_endpoint_messages_compose()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_load()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_send()`, `portal_endpoint_messages_tag()`, `portal_endpoint_messages_tags()`, `portal_endpoint_scrape()`, `portal_endpoint_scrape_add()`, `portal_folder_contacts_add()`, `portal_folder_contacts_remove()`, `portal_folder_mail_add()`, `portal_folder_mail_remove()`, `portal_meta()`, `portal_settings_changepass()`, `portal_settings_identity()`, and `portal_validate_request()`.

void portal_endpoint_folders_add (connection_t * *con*)

Create a new folder in response to a "folders.add" json-rpc portal request.

Note:

Currently only contexts for new mail folders and contacts folders are supported.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 502 of file endpoint.c.

References count, connection_t::http, json_object_size_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic, ns_length_get(), NULLER, PORTAL_ENDPOINT_CONTEXT_CONTACTS, PORTAL_ENDPOINT_CONTEXT_INVALID, PORTAL_ENDPOINT_CONTEXT_MAIL, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_FOLDERS_ADD, PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD, portal_folder_contacts_add(), portal_folder_mail_add(), portal_parse_context(), portal_validate_request(), st_char_get(), st_free(), st_import(), and st_length_get().

void portal_endpoint_folders_list (connection_t * con)

Return a list of a user's folders in response to a "folders.list" json-rpc portal request.

Note:

Currently only contexts for new mail folders and contacts folders are supported.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

LOW: User created folder names `_could_` have a NULL byte. How should we handle that?

LOW: User created folder names `_could_` have a NULL byte. How should we handle that?

Definition at line 563 of file endpoint.c.

References meta_folder_t::foldernum, connection_t::http, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), json_array_append_new_d, json_array_d, json_decref_d, json_pack_ex_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic, magma_folder_t, meta_folder_t::name, NULLER, meta_folder_t::parent, PLACER, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_FOLDERS_LIST, PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD, portal_endpoint_response(), portal_validate_request(), st_char_get(), st_cmp_ci_eq(), and st_length_get().

void portal_endpoint_folders_remove (connection_t * con)

Remove a user's folder in response to a json-rpc "folders.remove" portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 646 of file endpoint.c.

References connection_t::http, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic, NULLER, PORTAL_ENDPOINT_CONTEXT_CONTACTS, PORTAL_ENDPOINT_CONTEXT_INVALID, PORTAL_ENDPOINT_CONTEXT_MAIL, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_FOLDERS_REMOVE, PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD, portal_folder_contacts_remove(), portal_folder_mail_remove(), portal_parse_context(), portal_validate_request(), st_char_get(), and st_length_get().

void portal_endpoint_folders_rename (connection_t * con)

Rename a user's folder in response to a json-rpc "folders.rename" portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

HIGH: Use the context and act appropriately!

Definition at line 687 of file endpoint.c.

References contact_folder_rename(), connection_t::http, imap_folder_rename(), JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic, magma_folder_find_number(), magma_folder_name(), magma_folder_t, meta_folders_by_number(), meta_folders_name(), meta_user_unlock(), meta_user_wlock(), ns_length_get(), NULLER, OBJECT_CONTACTS, OBJECT_FOLDERS, PORTAL_ENDPOINT_CONTEXT_CONTACTS, PORTAL_ENDPOINT_CONTEXT_INVALID, PORTAL_ENDPOINT_CONTEXT_MAIL, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_CONSTRAINT_VIOLATION, PORTAL_ENDPOINT_ERROR_FOLDERS_RENAME, PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD, PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), portal_parse_context(), portal_validate_request(), serial_get(), serial_increment(), sess_trigger(), st_char_get(), st_free(), st_import(), and st_length_get().

void portal_endpoint_folders_tags (connection_t * con)

HIGH: Right now we only have indexes with the mail folders so other context types generate an empty array result.

Definition at line 832 of file endpoint.c.

References meta_stats_tag_t::count, connection_t::http, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_free(), json_decref_d, json_integer_d, json_object_d, json_object_set_new_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic, meta_folders_by_number(), meta_folders_stats_tags(), meta_user_rlock(), meta_user_unlock(), NULLER, PORTAL_ENDPOINT_CONTEXT_MAIL, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_FOLDERS_TAGS, PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), portal_parse_context(), portal_validate_request(), st_char_get(), st_length_get(), stats, and meta_stats_tag_t::tag.

void portal_endpoint_logout (connection_t * con)

Log a user out of a portal session in response to a "logout" json-rpc portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 290 of file endpoint.c.

References `connection_t::http`, `HTTP_COOKIE_DELETE`, `META_PROT_WEB`, `meta_user_ref_dec()`, `PORTAL_ENDPOINT_ERROR_LOGOUT`, `portal_endpoint_response()`, `portal_validate_request()`, and `SESSION_STATE_TERMINATED`.

void portal_endpoint_messages_compose (connection_t * con)

Compose a new message in response to a "messages.compose" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 1818 of file endpoint.c.

References `composition_t::attachments`, `composition_t::comp_id`, `connection_t::http`, `inx_alloc()`, `inx_find()`, `inx_insert()`, `json_object_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `log_error`, `M_INX_LINKED`, `M_TYPE_UINT64`, `mm_alloc()`, `mutex_lock()`, `mutex_unlock()`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_MESSAGES_COMPOSE`, `portal_endpoint_response()`, `portal_validate_request()`, `sess_release_attachment()`, `sess_release_composition()`, `multi_t::u64`, and `multi_t::val`.

void portal_endpoint_messages_copy (connection_t * con)

HIGH: If a multiple message copy fails in the middle, go back and remove any messages that may have been copied before the error.

Definition at line 990 of file endpoint.c.

References `count`, `meta_message_t::foldernum`, `connection_t::http`, `inx_find()`, `json_array_append_new_d`, `json_array_d`, `json_array_get_d`, `json_array_size_d`, `json_decref_d`, `json_integer_value_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_pedantic`, `M_TYPE_UINT64`, `meta_message_t::messagenum`, `meta_folders_by_number()`, `META_LOCKED`, `meta_messages_copier()`, `meta_messages_update_sequences()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_MESSAGES`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_MESSAGES_COPY`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `portal_validate_request()`, `serial_get()`, `serial_increment()`, `sess_trigger()`, `st_char_get()`, `st_length_get()`, `multi_t::u64`, and `multi_t::val`.

void portal_endpoint_messages_flag (connection_t * con)

HIGH: This function needs restructuring pretty badly. We are setting collection to NULL so we can add a cleanup call below. But if the function works as intended the reference is stolen by the response. but if an error occurs and the we don't return a response the collection ends up as a memory leak without the cleanup call below.

TODO: Were holding onto the user lock while waiting on the response to be sent over the wire; and this function could probably use a rethink.

HIGH: See the sister comment a few lines back!

Definition at line 1079 of file endpoint.c.

References count, meta_message_t::foldernum, connection_t::http, inx_alloc(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_find(), inx_free(), inx_insert(), json_array_append_new_d, json_array_d, json_array_get_d, json_array_size_d, json_decref_d, json_integer_value_d, json_object_size_d, json_pack_ex_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, MAIL_STATUS_SYSTEM_FLAGS, MAIL_STATUS_USER_FLAGS, meta_message_t::messagenum, meta_data_flags_add(), meta_data_flags_remove(), meta_data_flags_replace(), meta_folders_by_number(), meta_user_rlock(), meta_user_unlock(), meta_user_wlock(), NULLER, OBJECT_MESSAGES, PLACER, PORTAL_ENDPOINT_ACTION_ADD, PORTAL_ENDPOINT_ACTION_LIST, PORTAL_ENDPOINT_ACTION_REMOVE, PORTAL_ENDPOINT_ACTION_REPLACE, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION, PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD, PORTAL_ENDPOINT_ERROR_MESSAGES_FLAG, PORTAL_ENDPOINT_ERROR_REFERENCE, PORTAL_ENDPOINT_ERROR_SYSTEM_FLAG, portal_endpoint_response(), portal_message_flags_array(), portal_parse_flags(), portal_validate_request(), serial_get(), serial_increment(), sess_trigger(), st_char_get(), st_cmp_ci_eq(), st_length_get(), meta_message_t::status, multi_t::u64, and multi_t::val.

void portal_endpoint_messages_list (connection_t * con)

Retrieve a list of the user's messages in response to a json-rpc "messages.list" portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

LOW: Add the ability to track the recipient email address for a message, even if its not provided in the To field.

LOW: Add snippet support.

Definition at line 905 of file endpoint.c.

References ar_field_st(), ar_length_get(), count, meta_message_t::created, meta_message_t::foldernum, connection_t::http, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), json_array_append_new_d, json_array_d, json_decref_d, json_pack_ex_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_string_d, json_unpack_ex_d, log_pedantic, mail_header_fetch_cleaned(), mail_load_header(), meta_message_t::messagenum, PLACER, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_MESSAGES_LIST, portal_endpoint_response(), portal_message_flags_array(), portal_validate_request(), meta_message_t::size, st_char_get(), st_cleanup(), st_free(), st_import(), st_length_get(), and meta_message_t::tags.

void portal_endpoint_messages_load (connection_t * con)

Definition at line 1712 of file endpoint.c.

References data, connection_t::http, inx_find(), json_decref_d, json_object_d, json_object_set_new_d, json_object_size_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic, M_TYPE_UINT64, mail_destroy(), mail_load_message(), mail_mime_update(), portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION, PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD, PORTAL_ENDPOINT_ERROR_MESSAGES_LOAD, PORTAL_ENDPOINT_ERROR_READ, PORTAL_ENDPOINT_ERROR_REFERENCE, PORTAL_ENDPOINT_MESSAGE_ATTACHMENTS, PORTAL_ENDPOINT_MESSAGE_BODY, PORTAL_ENDPOINT_MESSAGE_HEADER, PORTAL_ENDPOINT_MESSAGE_INFO, PORTAL_ENDPOINT_MESSAGE_META, PORTAL_ENDPOINT_MESSAGE_SECURITY, PORTAL_ENDPOINT_MESSAGE_SERVER, PORTAL_ENDPOINT_MESSAGE_SOURCE, portal_endpoint_response(), portal_message_attachments(), portal_message_body(), portal_message_header(), portal_message_info(), portal_message_meta(), portal_message_security(), portal_message_server(), portal_message_source(), portal_parse_sections(), portal_validate_request(), connection_t::server, st_char_get(), st_length_get(), multi_t::u64, and multi_t::val.

void portal_endpoint_messages_move (connection_t * con)

HIGH: If a multiple message copy fails in the middle, go back and remove any messages that may have been copied before the error.

Definition at line 1279 of file endpoint.c.

References count, meta_message_t::foldernum, connection_t::http, inx_find(), json_array_get_d, json_array_size_d, json_integer_value_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic, M_TYPE_UINT64, meta_folders_by_number(), META_LOCKED, meta_messages_mover(), meta_messages_update_sequences(), meta_user_unlock(), meta_user_wlock(), OBJECT_MESSAGES, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION, PORTAL_ENDPOINT_ERROR_MESSAGES_MOVE, PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), portal_validate_request(), serial_get(), serial_increment(), sess_trigger(), st_char_get(), st_length_get(), multi_t::u64, and multi_t::val.

void portal_endpoint_messages_remove (connection_t * con)

Remove a user's mail message in response to a "messages.remove" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 1375 of file endpoint.c.

References count, meta_message_t::foldernum, connection_t::http, inx_delete(), inx_find(), json_array_get_d, json_array_size_d, json_integer_value_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic, M_TYPE_UINT64, mail_remove_message(), meta_message_t::messagenum, meta_folders_by_number(), meta_messages_update_sequences(), meta_user_unlock(), meta_user_wlock(), OBJECT_MESSAGES,

portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_MESSAGES_REMOVE,
PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), portal_validate_request(),
serial_get(), serial_increment(), meta_message_t::server, sess_trigger(), meta_message_t::size, st_char_get(),
st_length_get(), multi_t::u64, and multi_t::val.

void portal_endpoint_messages_send (connection_t * con)

Send a composed message in response to a "messages.send" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2012 of file endpoint.c.

References composition_t::attachments, connection_t::http, inx_delete(), inx_find(), inx_free(),
json_object_size_d, JSON_RPC_2_ERROR_SERVER_INTERNAL,
JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_error, log_pedantic,
M_TYPE_UINT64, mutex_lock(), mutex_unlock(), NULLER, portal_endpoint_error(),
PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION,
PORTAL_ENDPOINT_ERROR_MESSAGES_SEND, PORTAL_ENDPOINT_ERROR_REFERENCE,
portal_endpoint_response(), portal_outbound_checks(), portal_parse_json_str_array(),
portal_smtp_create_data(), portal_smtp_relay_message(), portal_validate_request(), st_char_get(), st_free(),
st_length_get(), multi_t::u64, and multi_t::val.

void portal_endpoint_messages_tag (connection_t * con)

Perform a tag operation on a user's message, in response to a "messages.tag" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

LOW: We don't need to add the flag to every message. Just the ones that don't already have the flag.

LOW: Like the line, were not checking whether the message even needs to have the flag removed.
Were also not handling remove requests that result in a message having no tags.

If this is going supposed to be a replace operation, we truncate all of the message tags so we can
insert the replacements below.

TODO: This is ugly. Were rebuilding the tags array from the database on each iteration because its
easier than trying to figure out which slot needs to be removed.

TODO: If the update is triggered before the tagged flag is added to the database the refresh might not
pull the data for who recently got tagged! We need to need to look for this type of sequence bug
elsewhere too.

TODO: Were holding onto the user lock while were waiting on the response to be sent over the wire;
and this function could probably use a rethink.

Definition at line 1515 of file endpoint.c.

References `ar_field_st()`, `ar_free()`, `ar_length_get()`, `meta_message_t::foldernum`, `connection_t::http`, `inx_alloc()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_find()`, `inx_free()`, `inx_insert()`, `json_array_append_new_d`, `json_array_d`, `json_array_get_d`, `json_array_size_d`, `json_decref_d`, `json_integer_value_d`, `json_object_size_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_string_d`, `json_string_value_d`, `json_unpack_ex_d`, `log_pedantic`, `M_INX_LINKED`, `M_TYPE_UINT64`, `MAIL_STATUS_TAGGED`, `meta_message_t::messagenum`, `meta_data_delete_tag()`, `meta_data_fetch_message_tags()`, `meta_data_flags_add()`, `meta_data_flags_remove()`, `meta_data_insert_tag()`, `meta_data_truncate_tags()`, `meta_folders_by_number()`, `meta_user_rlock()`, `meta_user_serial_check()`, `meta_user_unlock()`, `meta_user_wlock()`, `NULLER`, `OBJECT_MESSAGES`, `PLACER`, `PORTAL_ENDPOINT_ACTION_ADD`, `PORTAL_ENDPOINT_ACTION_LIST`, `PORTAL_ENDPOINT_ACTION_REMOVE`, `PORTAL_ENDPOINT_ACTION_REPLACE`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION`, `PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD`, `PORTAL_ENDPOINT_ERROR_MESSAGES_TAG`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `portal_validate_request()`, `st_char_get()`, `st_cmp_ci_eq()`, `st_length_get()`, `meta_message_t::tags`, `multi_t::u64`, and `multi_t::val`.

void portal_endpoint_messages_tags (connection_t * con)

Get all of the message tags used by a specified user in response to a "messages.tags" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 1464 of file endpoint.c.

References `connection_t::http`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_free()`, `json_array_append_new_d`, `json_array_d`, `json_decref_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `json_string_d`, `log_error`, `log_pedantic`, `meta_data_fetch_all_tags()`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_MESSAGES_TAGS`, `portal_endpoint_response()`, `portal_validate_request()`, and `st_char_get()`.

void portal_endpoint_response (connection_t * con, chr_t * format, ...)

Generate a json-rpc 2.0 response to a portal request.

See also:

`json_vpack_ex()`

Note:

This function indents the json response if specified in the configuration, and also automatically decreases the reference count of any json object that was packed for the reply.

Parameters:

con a pointer to the connection object across which the portal response will be sent.

format a pointer to a format string specifying the construction of the json-rpc response.

... a variable arguments style list of parameters to be passed to the json packing function.

Returns:

This function returns no value.

Definition at line 168 of file endpoint.c.

References `con_write_st()`, `connection_t::http`, `HTTP_ERROR_500`, `http_response_header()`, `json_decref_d`, `json_dumps_d`, `json_vpack_ex_d`, `log_pedantic`, `magma`, `ns_append()`, `ns_free()`, `ns_length_get()`, `NULLER`, `PLACER`, `magma_t::portal`, and `magma_t::web`.

Referenced by `portal_endpoint_alert_acknowledge()`, `portal_endpoint_alert_list()`, `portal_endpoint_aliases()`, `portal_endpoint_attachments_add()`, `portal_endpoint_attachments_remove()`, `portal_endpoint_auth()`, `portal_endpoint_config_edit()`, `portal_endpoint_config_load()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_contacts_edit()`, `portal_endpoint_contacts_list()`, `portal_endpoint_contacts_load()`, `portal_endpoint_contacts_move()`, `portal_endpoint_contacts_remove()`, `portal_endpoint_cookies()`, `portal_endpoint_folders_list()`, `portal_endpoint_folders_rename()`, `portal_endpoint_folders_tags()`, `portal_endpoint_logout()`, `portal_endpoint_messages_compose()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_load()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_send()`, `portal_endpoint_messages_tag()`, `portal_endpoint_messages_tags()`, `portal_folder_contacts_add()`, `portal_folder_contacts_remove()`, `portal_folder_mail_add()`, `portal_folder_mail_remove()`, `portal_meta()`, `portal_settings_changepass()`, and `portal_settings_identity()`.

void portal_endpoint_scrape (connection_t * con)**Note:**

This function is not implemented and may be removed entirely in the future.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2812 of file endpoint.c.

References `JSON_RPC_2_ERROR_SERVER_METHOD_UNAVAIL`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_AD`, and `portal_validate_request()`.

void portal_endpoint_scrape_add (connection_t * con)**Note:**

This function is not implemented and may be removed entirely in the future.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2796 of file endpoint.c.

References `JSON_RPC_2_ERROR_SERVER_METHOD_UNAVAIL`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_AD`, and `portal_validate_request()`.

void portal_endpoint_search (connection_t * con)

Definition at line 2774 of file endpoint.c.

References `con_write_st()`, `connection_t::http`, `HTTP_ERROR_500`, `http_response_header()`, `PLACER`, `PORTAL_ENDPOINT_ERROR_SEARCH`, `st_aprint()`, `st_free()`, and `st_length_get()`.

void portal_endpoint_sort (void)

Sort the web portal JSON command table to be ready for binary searches.

TODO: We use session locks to synchronize access to the user context. Should we also be using mailbox cluster locks? Cluster level locking might be appropriate for operations that delete data likes `folders.remove` and `messages.remove`.

Returns:

This function returns no value.

Definition at line 26 of file `endpoint.c`.

References `portal_endpoint_compare()`, and `portal_methods`.

Referenced by `protocol_init()`.

attachment_t* portal_get_upload_attachment (connection_t * con)

Get a user's attachment to a message composition uploaded via multipart form data.

Note:

Parameters:

path a pointer to a managed string containing the full /attach path specified in the client's http POST request.

Definition at line 3053 of file `endpoint.c`.

References `composition_t::attachments`, `attachment_t::filedata`, `connection_t::http`, `inx_find()`, `log_pedantic`, `M_TYPE_UINT64`, `mutex_lock()`, `mutex_unlock()`, `placer_t`, `tok_get_count_st()`, `tok_get_st()`, `multi_t::u64`, `uint64_conv_st()`, and `multi_t::val`.

Referenced by `portal_upload()`.

void portal_meta (connection_t * con)

Return information for a portal "meta" json-rpc request.

Note:

This function returns various pieces of information about the user such as their plan type, quota, etc.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2879 of file `endpoint.c`.

References `con_addr_presentation()`, `connection_t::http`, `json_object_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `log_pedantic`, `MAGMA_PORTAL_VERSION`, `MANAGEDBUF`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_META`, `portal_endpoint_response()`, `portal_validate_request()`, and `st_char_get()`.

void portal_settings_changepass (connection_t * con)

Change a user's password in response to a portal "settings.changepass" json-rpc request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2916 of file endpoint.c.

References connection_t::http, json_object_d, json_pack_ex_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic, MAGMA_PORTAL_VERSION, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_SETTINGS_CHANGEPASS, portal_endpoint_response(), portal_validate_request(), st_char_get(), and st_length_get().

void portal_settings_identity (connection_t * con)

Return information for a portal "settings.identity" json-rpc request.

Note:

This function returns the full name, first name, last name, and website of the requested user.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2846 of file endpoint.c.

References connection_t::http, json_object_d, json_pack_ex_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, log_pedantic, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_SETTINGS_IDENTITY, portal_endpoint_response(), portal_validate_request(), and st_char_get().

void portal_upload (connection_t * con)

Process uploaded attachments for messages composed in conjunction with the portal interface.

Definition at line 3126 of file endpoint.c.

References con_addr_word(), con_secure(), data, attachment_t::filedata, get_header_opt(), connection_t::http, http_data_free(), http_data_header_parse_line(), HTTP_ERROR_401, HTTP_ERROR_403, HTTP_ERROR_404, HTTP_ERROR_405, HTTP_ERROR_500, HTTP_MERGED, HTTP_METHOD_POST, http_parse_context(), HTTP_PORTAL, http_print_301(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), log_error, log_pedantic, magma, multipart_get_boundary(), http_data_t::name, NULLER, pl_char_get(), pl_empty(), pl_length_get(), pl_null(), pl_shrink_before_characters(), pl_skip_characters(), pl_skip_to_characters(), PLACER, placer_t, magma_t::portal, portal_get_upload_attachment(), st_char_get(), st_cmp_ci_eq(), st_import(), st_length_get(), str_tok_get_bl(), str_tok_get_count_bl(), http_data_t::value, and magma_t::web.

Referenced by http_response().

bool_t portal_validate_request (connection_t * con, int err_mask, chr_t * method, bool_t has_params, size_t nparams)

Verify that a session underlying a portal request has been authenticated.

Note:

If the session is not authenticated, an error message will be sent to the client, so the caller only needs to return from its function if this function returns false.

Parameters:

con a pointer to the connection object across which the portal request was made.

err_mask an optional error bitmask to be combined with any portal authentication error code.

method a pointer to a null-terminated string that will be used in an optional error logging message if it is not NULL.

has_params if true, return an error if the authenticated portal session supplied no method parameters; if false, return an error if the authenticated portal supplies params and they're not empty.

nparams if non-zero and *has_params* is also set, return an error if the portal session's parameters count does not match this value.

Returns:

Definition at line 70 of file endpoint.c.

References `connection_t::http`, `json_object_size_d`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `log_pedantic`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_MODE`, `SESSION_STATE_AUTHENTICATED`, `st_char_get()`, and `st_length_get()`.

Referenced by `portal_endpoint_ad()`, `portal_endpoint_alert_acknowledge()`, `portal_endpoint_alert_list()`, `portal_endpoint_aliases()`, `portal_endpoint_attachments_add()`, `portal_endpoint_attachments_remove()`, `portal_endpoint_config_edit()`, `portal_endpoint_config_load()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_contacts_edit()`, `portal_endpoint_contacts_list()`, `portal_endpoint_contacts_load()`, `portal_endpoint_contacts_move()`, `portal_endpoint_contacts_remove()`, `portal_endpoint_cookies()`, `portal_endpoint_folders_add()`, `portal_endpoint_folders_list()`, `portal_endpoint_folders_remove()`, `portal_endpoint_folders_rename()`, `portal_endpoint_folders_tags()`, `portal_endpoint_logout()`, `portal_endpoint_messages_compose()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_load()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_send()`, `portal_endpoint_messages_tag()`, `portal_endpoint_messages_tags()`, `portal_endpoint_scrape()`, `portal_endpoint_scrape_add()`, `portal_meta()`, `portal_settings_changepass()`, and `portal_settings_identity()`.

magma/web/portal/mail.c File Reference

Functions for various smtp-level functionality for use in conjunction with the portal.

```
#include "magma.h"
```

Functions

- **bool_t portal_outbound_checks** (**credential_t** *cred, **uint64_t** usernum, **stringer_t** *from, **size_t** num_recipients, **stringer_t** *body_plain, **stringer_t** *body_html, **chr_t** **errmsg)
- *Perform a series of security-related checks on an outbound email message in a transport-independent manner.* **bool_t portal_smtp_relay_message** (**stringer_t** *from, **inx_t** *to, **stringer_t** *data, **size_t** send_size, **chr_t** **errmsg)
- *Send (relay) a message composed by a user via a portal session.* **stringer_t** * **portal_smtp_create_data** (**inx_t** *attachments, **stringer_t** *from, **inx_t** *to, **inx_t** *cc, **inx_t** *bcc, **stringer_t** *subject, **stringer_t** *body_plain, **stringer_t** *body_html)
- *Create the data of an outbound smtp message that will be specified with the smtp DATA command.* **stringer_t** * **portal_smtp_merge_headers** (**inx_t** *headers, **stringer_t** *leading, **stringer_t** *trailing)

Merge a list of smtp message headers into a single string, preceded by the leading text and followed by the trailing text.

Detailed Description

Functions for various smtp-level functionality for use in conjunction with the portal.

Definition in file **mail.c**.

Function Documentation

bool_t portal_outbound_checks (**credential_t** * cred, **uint64_t** usernum, **stringer_t** * from, **size_t** num_recipients, **stringer_t** * body_plain, **stringer_t** * body_html, **chr_t** ** errmsg)

Perform a series of security-related checks on an outbound email message in a transport-independent manner.

mail.c

See also:

smtp_data_outbound()

Note:

The checks include: Pattern matching for junk mail, authorization for the user to use the email address or domain specified in the From address, and finally, a virus check.

Parameters:

cred a credential structure obtained for the authenticated user's session.

usenum the numerical id of the user attempting to send the message.

from a managed string containing the email address specified as the From address.

nrecipients the number of recipients that will receive the sent message.

body_plain a managed string containing the plain text body of the message.

body_html a managed string containing the html body of the message.

errmsg the address of a pointer to a null-terminated string that will be set to a descriptive error message on failure.

Returns:

true if all security checks were passed or false otherwise.

Definition at line 29 of file mail.c.

References `pattern_check()`, `smtp_check_authorized_from()`, `smtp_check_transmit_quota()`, `smtp_fetch_authorization()`, and `virus_check()`.

Referenced by `portal_endpoint_messages_send()`.

stringer_t* portal_smtp_create_data (inx_t * *attachments*, stringer_t * *from*, inx_t * *to*, inx_t * *cc*, inx_t * *bcc*, stringer_t * *subject*, stringer_t * *body_plain*, stringer_t * *body_html*)

Create the data of an outbound smtp message that will be specified with the smtp DATA command.

Parameters:

attachments an optional inx holder containing a list of attachments to be included in the message.

from a managed string containing the email address sending the message.

to an inx holder containing a list of To: email recipients as managed strings.

cc an inx holder containing a list of 0 or more CC: recipients as managed strings.

bcc an inx holder containing a list of 0 or more BCC: recipients as managed strings.

subject a managed string containing the subject of the message.

body_plain an optional managed string containing the plain text body of the message.

body_html an optional managed string containing the html-formatted body of the message.

Returns:

NULL on failure or a managed string containing the packaged outbound smtp message data on success.

Definition at line 182 of file mail.c.

References `ar_append()`, `ar_free()`, `ar_length_get()`, `ARRAY_TYPE_POINTER`, `attachment_t::filedata`, `attachment_t::filename`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_reset()`, `inx_cursor_value_next()`, `log_pedantic`, `mail_mime_encode_part()`, `mail_mime_generate_boundary()`, `mail_mime_get_smtp_envelope()`, `st_free()`, and `st_merge`.

Referenced by `portal_endpoint_messages_send()`.

stringer_t* portal_smtp_merge_headers (inx_t * *headers*, stringer_t * *leading*, stringer_t * *trailing*)

Merge a list of smtp message headers into a single string, preceded by the leading text and followed by the trailing text.

Parameters:

headers an inx holder containing a collection of header string data to be merged together.

leading a managed string containing text that will lead each header line.

trailing a managed string containing text that will trail each header line.

Returns:

NULL on failure or a managed string containing the merged headers on success.

Definition at line 306 of file mail.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `st_alloc`, `st_cleanup()`, and `st_merge`.

Referenced by `mail_mime_get_smtp_envelope()`.

bool_t portal_smtp_relay_message (stringer_t * *from*, inx_t * *to*, stringer_t * *data*, size_t *send_size*, chr_t ** *errmsg*)

Send (relay) a message composed by a user via a portal session.

See also:

smtp_relay_message() - a lot of logic borrowed from here.

Parameters:

from a pointer to a managed string containing the email address specified as the From address.

to a pointer to a managed string containing the destination email address of the message.

data a pointer to a managed string containing the raw data of the mail message.

send_size if greater than 0, specify the optional SIZE parameter to the MAIL FROM command.

errmsg the address of a pointer to a null-terminated string that will be set to a descriptive error message on failure.

Returns:

true if the mail message was sent successfully, or false otherwise.

Definition at line 91 of file mail.c.

References inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), smtp_client_close(),
smtp_client_connect(), smtp_client_send_data(), smtp_client_send_helo(), smtp_client_send_mailfrom(), and
smtp_client_send_rcptto().

Referenced by portal_endpoint_messages_send().

magma/web/portal/methods.h File Reference

Definitions for all supported web portal methods and their calling interfaces.

Variables

- `command_t portal_methods []`
-

Detailed Description

Definitions for all supported web portal methods and their calling interfaces.

Definition in file **methods.h**.

Variable Documentation

`command_t portal_methods[]`

Definition at line 27 of file `methods.h`.

Referenced by `portal_endpoint()`, and `portal_endpoint_sort()`.

magma/web/portal/portal.c File Reference

The portal web application.
`#include "magma.h"`

Functions

- void **portal_print_login** (**connection_t** *con, **chr_t** *message)
 - *Display the http portal login page.* void **portal_process** (**connection_t** *con)
Process a connection to the web portal.
-

Detailed Description

The portal web application.

Definition in file **portal.c**.

Function Documentation

void portal_print_login (connection_t * con, chr_t * message)

Display the http portal login page.

portal.c

Note:

The portal login page can be found in 'portal/login.template'

Parameters:

con a pointer to the connection object of the client requesting the portal login page.

message a pointer to a null-terminated string that will be displayed to the user in the login page as the contents of the node with the id of "message" in the portal login template.

Returns:

This function returns no value.

Definition at line 23 of file portal.c.

References `con_write_st()`, `http_page_t::content`, `http_page_t::doc_obj`, `http_page_free()`, `http_page_get()`, `http_print_500()`, `http_response_header()`, `st_free()`, `st_length_get()`, `http_content_t::type`, `xml_dump_doc()`, `xml_node_set_content()`, `xml_xpath_eval()`, and `http_page_t::xpath_ctx`.

Referenced by `portal_process()`.

void portal_process (connection_t * con)

Process a connection to the web portal.

Note:

If `magma.web.portal.safeguard` is set, the user will be redirected to a secure login.

Parameters:

con a pointer to the connection of the http client requesting the portal.

Returns:

This function returns no value.

Definition at line 62 of file portal.c.

References `con_addr_word()`, `con_secure()`, `http_parse_context()`, `http_print_301()`, `magma`, `PLACER`, `magma_t::portal`, `portal_print_login()`, and `magma_t::web`.

Referenced by `http_response()`.

magma/web/portal/portal.h File Reference

The portal web application.

Defines

- `#define MAGMA_PORTAL_VERSION "1.02"`

Enumerations

- `enum { JSON_RPC_2_ERROR_PARSE_MALFORMED = -32700, JSON_RPC_2_ERROR_PARSE_ENCODING = -32701, JSON_RPC_2_ERROR_PARSE_CHAR = -32702, JSON_RPC_2_ERROR_SERVER_REQUEST = -32600, JSON_RPC_2_ERROR_SERVER_METHOD_UNAVAIL = -32601, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS = -32602, JSON_RPC_2_ERROR_SERVER_INTERNAL = -32603, JSON_RPC_2_ERROR_APPLICATION = -32500, JSON_RPC_2_ERROR_SYSTEM = -32400, JSON_RPC_2_ERROR_TRANSPORT = -32300 }`
- `enum { PORTAL_ENDPOINT_ACTION_INVALID = -1, PORTAL_ENDPOINT_ACTION_ADD = 1, PORTAL_ENDPOINT_ACTION_REMOVE = 2, PORTAL_ENDPOINT_ACTION_REPLACE = 3, PORTAL_ENDPOINT_ACTION_LIST = 4 }`
- `enum { PORTAL_ENDPOINT_CONTEXT_INVALID = -1, PORTAL_ENDPOINT_CONTEXT_MAIL = 1, PORTAL_ENDPOINT_CONTEXT_CONTACTS = 2, PORTAL_ENDPOINT_CONTEXT_SETTINGS = 3, PORTAL_ENDPOINT_CONTEXT_HELP = 4 }`
- `enum { PORTAL_ENDPOINT_MESSAGE_META = 1, PORTAL_ENDPOINT_MESSAGE_SOURCE = 2, PORTAL_ENDPOINT_MESSAGE_SECURITY = 4, PORTAL_ENDPOINT_MESSAGE_SERVER = 8, PORTAL_ENDPOINT_MESSAGE_HEADER = 16, PORTAL_ENDPOINT_MESSAGE_BODY = 32, PORTAL_ENDPOINT_MESSAGE_ATTACHMENTS = 64, PORTAL_ENDPOINT_MESSAGE_INFO = 128 }`
- `enum { PORTAL_ENDPOINT_ERROR_MODE = 10000, PORTAL_ENDPOINT_ERROR_REFERENCE = 10100, PORTAL_ENDPOINT_ERROR_CONSTRAINT_VIOLATION = 10200, PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD = 10300, PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION = 10300, PORTAL_ENDPOINT_ERROR_SYSTEM_FLAG = 10400, PORTAL_ENDPOINT_ERROR_READ = 10500 }`
- `enum { PORTAL_ENDPOINT_ERROR_NONE = 0, PORTAL_ENDPOINT_ERROR_AUTH, PORTAL_ENDPOINT_ERROR_COOKIES, PORTAL_ENDPOINT_ERROR_AD, PORTAL_ENDPOINT_ERROR_ALERT_LIST, PORTAL_ENDPOINT_ERROR_ALERT_ACKNOWLEDGE, PORTAL_ENDPOINT_ERROR_ALIASES, PORTAL_ENDPOINT_ERROR_SCRAPE_ADD, PORTAL_ENDPOINT_ERROR_SCRAPE, PORTAL_ENDPOINT_ERROR_ATTACHMENTS_ADD, PORTAL_ENDPOINT_ERROR_ATTACHMENTS_PROGRESS, PORTAL_ENDPOINT_ERROR_ATTACHMENTS_REMOVE, PORTAL_ENDPOINT_ERROR_CONFIG_LOAD, PORTAL_ENDPOINT_ERROR_CONFIG_EDIT, PORTAL_ENDPOINT_ERROR_CONTACTS_ADD, PORTAL_ENDPOINT_ERROR_CONTACTS_EDIT, PORTAL_ENDPOINT_ERROR_CONTACTS_LIST, PORTAL_ENDPOINT_ERROR_CONTACTS_LOAD, PORTAL_ENDPOINT_ERROR_CONTACTS_MOVE, PORTAL_ENDPOINT_ERROR_CONTACTS_COPY, PORTAL_ENDPOINT_ERROR_CONTACTS_REMOVE, PORTAL_ENDPOINT_ERROR_FOLDERS_ADD, PORTAL_ENDPOINT_ERROR_FOLDERS_LIST, PORTAL_ENDPOINT_ERROR_FOLDERS_TAGS,`

PORTAL_ENDPOINT_ERROR_FOLDERS_REMOVE,
PORTAL_ENDPOINT_ERROR_FOLDERS_RENAME,
PORTAL_ENDPOINT_ERROR_MESSAGES_COMPOSE,
PORTAL_ENDPOINT_ERROR_MESSAGES_COPY,
PORTAL_ENDPOINT_ERROR_MESSAGES_FLAG,
PORTAL_ENDPOINT_ERROR_MESSAGES_LIST,
PORTAL_ENDPOINT_ERROR_MESSAGES_LOAD,
PORTAL_ENDPOINT_ERROR_MESSAGES_MOVE,
PORTAL_ENDPOINT_ERROR_MESSAGES_REMOVE,
PORTAL_ENDPOINT_ERROR_MESSAGES_SEND,
PORTAL_ENDPOINT_ERROR_MESSAGES_TAG,
PORTAL_ENDPOINT_ERROR_MESSAGES_TAGS, **PORTAL_ENDPOINT_ERROR_META,**
PORTAL_ENDPOINT_ERROR_SEARCH, **PORTAL_ENDPOINT_ERROR_SETTINGS_IDENTITY,**
PORTAL_ENDPOINT_ERROR_SETTINGS_CHANGEPASS,
PORTAL_ENDPOINT_ERROR_LOGOUT }

Functions

- `json_t * portal_config_collection (user_config_t *collection)`
- `config.c json_t * portal_config_entry (user_config_entry_t *entry)`
- `json_t * portal_config_entry_flags (user_config_entry_t *entry)`
- *Get a user config entry's flags as a json object.* `json_t * portal_contact_detail_flags (contact_detail_t *detail)`
- `contacts.c json_t * portal_contact_details (contact_t *contact)`
- *Retrieve all the details about a contact entry as a json object.* `void portal_endpoint (connection_t *con)`
- `endpoint.c void portal_endpoint_ad (connection_t *con)`
- *Return advertising information for a portal "ad" json-rpc request.* `void portal_endpoint_alert_acknowledge (connection_t *con)`
- *Process user alert message acknowledgements in response to an "alert.acknowledge" json-rpc portal request.* `void portal_endpoint_alert_list (connection_t *con)`
- *Get the list of a user's alert messages in response to an "alert.list" json-rpc portal request.* `void portal_endpoint_aliases (connection_t *con)`
- *Return the list of a user account's aliases in response to an "aliases" json-rpc portal request.* `void portal_endpoint_attachments_add (connection_t *con)`
- *Add an attachment to a message being composed in response to an "attachments.add" json-rpc portal request.* `void portal_endpoint_attachments_progress (connection_t *con)`
- `void portal_endpoint_attachments_remove (connection_t *con)`
- *Remove an attachment from a message being composed in response to an "attachments.remove" json-rpc portal request.* `void portal_endpoint_auth (connection_t *con)`
- *Obtain credentials and log a user into portal session in response to an "auth" json-rpc portal request.* `int_t portal_endpoint_compare (const void *compare, const void *command)`
- *Internal bsearch() comparison function for dispatching the proper portal handler.* `void portal_endpoint_config_edit (connection_t *con)`
- *Retrieve all of a user's config options and return them to the remote client in response to a portal "contacts.load" json-rpc request.* `void portal_endpoint_config_load (connection_t *con)`
- *Retrieve all of a user's config options and return them to the remote client in response to a portal "contacts.load" json-rpc request.* `void portal_endpoint_contacts_add (connection_t *con)`
- *Add a contact in response to a "contacts.add" json-rpc portal request.* `void portal_endpoint_contacts_copy (connection_t *con)`
- `void portal_endpoint_contacts_edit (connection_t *con)`
- `void portal_endpoint_contacts_list (connection_t *con)`
- *List a user's contacts in response to a "contacts.list" json-rpc portal request.* `void portal_endpoint_contacts_load (connection_t *con)`
- *Return information for a portal "contacts.load" json-rpc request.* `void portal_endpoint_contacts_move (connection_t *con)`

- Move a user's contact entry to another contacts folder in response to a "contacts.move" json-rpc portal request. void **portal_endpoint_contacts_remove** (connection_t *con)
- Remove a user's contact entry in response to a "contacts.remove" json-rpc portal request. void **portal_endpoint_cookies** (connection_t *con)
- Return whether or not a user is using cookies, in response to a "cookies" json-rpc portal request. void **portal_endpoint_error** (connection_t *con, int_t http_code, int_t error_code, chr_t *message)
- Return a json-rpc error response to the remote client. void **portal_endpoint_folders_add** (connection_t *con)
- Create a new folder in response to a "folders.add" json-rpc portal request. void **portal_endpoint_folders_list** (connection_t *con)
- Return a list of a user's folders in response to a "folders.list" json-rpc portal request. void **portal_endpoint_folders_remove** (connection_t *con)
- Remove a user's folder in response to a json-rpc "folders.remove" portal request. void **portal_endpoint_folders_rename** (connection_t *con)
- Rename a user's folder in response to a json-rpc "folders.rename" portal request. void **portal_endpoint_folders_tags** (connection_t *con)
- void **portal_endpoint_logout** (connection_t *con)
- Log a user out of a portal session in response to a "logout" json-rpc portal request. void **portal_endpoint_messages_compose** (connection_t *con)
- Compose a new message in response to a "messages.compose" json-rpc portal request. void **portal_endpoint_messages_copy** (connection_t *con)
- void **portal_endpoint_messages_flag** (connection_t *con)
- void **portal_endpoint_messages_list** (connection_t *con)
- Retrieve a list of the user's messages in response to a json-rpc "messages.list" portal request. void **portal_endpoint_messages_load** (connection_t *con)
- void **portal_endpoint_messages_move** (connection_t *con)
- void **portal_endpoint_messages_remove** (connection_t *con)
- Remove a user's mail message in response to a "messages.remove" json-rpc portal request. void **portal_endpoint_messages_tags** (connection_t *con)
- Get all of the message tags used by a specified user in response to a "messages.tags" json-rpc portal request. void **portal_endpoint_messages_tag** (connection_t *con)
- Perform a tag operation on a user's message, in response to a "messages.tag" json-rpc portal request. void **portal_endpoint_response** (connection_t *con, chr_t *format,...)
- Generate a json-rpc 2.0 response to a portal request. void **portal_endpoint_scrape** (connection_t *con)
- void **portal_endpoint_scrape_add** (connection_t *con)
- void **portal_endpoint_search** (connection_t *con)
- void **portal_settings_identity** (connection_t *con)
- Return information for a portal "settings.identity" json-rpc request. void **portal_meta** (connection_t *con)
- Return information for a portal "meta" json-rpc request. void **portal_endpoint_sort** (void)
- Sort the web portal JSON command table to be ready for binary searches. void **portal_upload** (connection_t *con)
- Process uploaded attachments for messages composed in conjunction with the portal interface. void **portal_endpoint_messages_send** (connection_t *con)
- Send a composed message in response to a "messages.send" json-rpc portal request. void **portal_debug** (connection_t *con)
- A portal debug function that will be disabled and/or deleted completely in production. bool_t **portal_outbound_checks** (credential_t *cred, uint64_t usernum, stringer_t *from, size_t num_recipients, stringer_t *body_plain, stringer_t *body_html, chr_t **errmsg)
- *mail.c* bool_t **portal_smtp_relay_message** (stringer_t *from, inx_t *to, stringer_t *data, size_t send_size, chr_t **errmsg)
- Send (relay) a message composed by a user via a portal session. stringer_t * **portal_smtp_create_data** (inx_t *attachments, stringer_t *from, inx_t *to, inx_t *cc, inx_t *bcc, stringer_t *subject, stringer_t *body_plain, stringer_t *body_html)
- Create the data of an outbound smtp message that will be specified with the smtp DATA command. stringer_t * **portal_smtp_merge_headers** (inx_t *headers, stringer_t *leading, stringer_t *trailing)

- *Merge a list of smtp message headers into a single string, preceded by the leading text and followed by the trailing text.* json_t * **portal_message_flags_array** (meta_message_t *meta)
 - *flags.c* json_t * **portal_message_tags_array** (meta_message_t *meta)
 - **int_t portal_parse_flags** (json_t *array, uint32_t *flags)
 - void **portal_folder_contacts_add** (connection_t *con, stringer_t *name, uint64_t parent)
 - *folders.c* void **portal_folder_contacts_remove** (connection_t *con, uint64_t foldernum)
 - *Remove a user's contacts folder.* void **portal_folder_mail_add** (connection_t *con, stringer_t *name, uint64_t parent)
 - *Add a new mail folder for a user.* void **portal_folder_mail_remove** (connection_t *con, uint64_t foldernum)
 - *Remove a user's mail folder.* json_t * **portal_message_attachments** (meta_message_t *meta, mail_message_t *data)
 - *messages.c* json_t * **portal_message_body** (meta_message_t *meta, mail_message_t *data)
 - json_t * **portal_message_header** (meta_message_t *meta, mail_message_t *data)
 - *Build the "header" section response to a rpc-json "messages.load" request.* json_t * **portal_message_meta** (meta_message_t *meta)
 - json_t * **portal_message_security** (meta_message_t *meta)
 - json_t * **portal_message_server** (meta_message_t *meta)
 - json_t * **portal_message_source** (meta_message_t *meta)
 - json_t * **portal_message_info** (meta_message_t *meta)
 - *Build the "info" section response to a rpc-json "messages.load" request.* inx_t * **portal_parse_json_str_array** (json_t *json, size_t *nout)
 - *parse.c* **int_t portal_parse_action** (stringer_t *action)
 - **int_t portal_parse_context** (stringer_t *context)
 - *Parse the context of a requested folder.* **int_t portal_parse_sections** (json_t *array, uint32_t *sections)
 - *Parse the json-rpc messages.load parameter "section" from an array of strings into a bitmask of section flags.* void **portal_print_login** (connection_t *con, chr_t *message)
 - *portal.c* void **portal_process** (connection_t *con)
- Process a connection to the web portal.*
-

Detailed Description

The portal web application.

Definition in file **portal.h**.

Define Documentation

#define MAGMA_PORTAL_VERSION "1.02"

Definition at line 16 of file portal.h.

Referenced by portal_meta(), and portal_settings_changepass().

Enumeration Type Documentation

anonymous enum

Enumerator:

JSON_RPC_2_ERROR_PARSE_MALFORMED
JSON_RPC_2_ERROR_PARSE_ENCODING
JSON_RPC_2_ERROR_PARSE_CHAR
JSON_RPC_2_ERROR_SERVER_REQUEST
JSON_RPC_2_ERROR_SERVER_METHOD_UNAVAIL
JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS
JSON_RPC_2_ERROR_SERVER_INTERNAL
JSON_RPC_2_ERROR_APPLICATION
JSON_RPC_2_ERROR_SYSTEM
JSON_RPC_2_ERROR_TRANSPORT

Definition at line 19 of file portal.h.

anonymous enum

Enumerator:

PORTAL_ENDPOINT_ACTION_INVALID
PORTAL_ENDPOINT_ACTION_ADD
PORTAL_ENDPOINT_ACTION_REMOVE
PORTAL_ENDPOINT_ACTION_REPLACE
PORTAL_ENDPOINT_ACTION_LIST

Definition at line 34 of file portal.h.

anonymous enum

Enumerator:

PORTAL_ENDPOINT_CONTEXT_INVALID
PORTAL_ENDPOINT_CONTEXT_MAIL
PORTAL_ENDPOINT_CONTEXT_CONTACTS
PORTAL_ENDPOINT_CONTEXT_SETTINGS
PORTAL_ENDPOINT_CONTEXT_HELP

Definition at line 42 of file portal.h.

anonymous enum

Enumerator:

PORTAL_ENDPOINT_MESSAGE_META
PORTAL_ENDPOINT_MESSAGE_SOURCE
PORTAL_ENDPOINT_MESSAGE_SECURITY
PORTAL_ENDPOINT_MESSAGE_SERVER
PORTAL_ENDPOINT_MESSAGE_HEADER
PORTAL_ENDPOINT_MESSAGE_BODY
PORTAL_ENDPOINT_MESSAGE_ATTACHMENTS
PORTAL_ENDPOINT_MESSAGE_INFO

Definition at line 50 of file portal.h.

anonymous enum

Enumerator:

PORTAL_ENDPOINT_ERROR_MODE
PORTAL_ENDPOINT_ERROR_REFERENCE
PORTAL_ENDPOINT_ERROR_CONSTRAINT_VIOLATION
PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD
PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION
PORTAL_ENDPOINT_ERROR_SYSTEM_FLAG
PORTAL_ENDPOINT_ERROR_READ

Definition at line 61 of file portal.h.

anonymous enum

Enumerator:

PORTAL_ENDPOINT_ERROR_NONE
PORTAL_ENDPOINT_ERROR_AUTH
PORTAL_ENDPOINT_ERROR_COOKIES
PORTAL_ENDPOINT_ERROR_AD
PORTAL_ENDPOINT_ERROR_ALERT_LIST
PORTAL_ENDPOINT_ERROR_ALERT_ACKNOWLEDGE
PORTAL_ENDPOINT_ERROR_ALIASES
PORTAL_ENDPOINT_ERROR_SCRAPE_ADD
PORTAL_ENDPOINT_ERROR_SCRAPE
PORTAL_ENDPOINT_ERROR_ATTACHMENTS_ADD
PORTAL_ENDPOINT_ERROR_ATTACHMENTS_PROGRESS
PORTAL_ENDPOINT_ERROR_ATTACHMENTS_REMOVE
PORTAL_ENDPOINT_ERROR_CONFIG_LOAD
PORTAL_ENDPOINT_ERROR_CONFIG_EDIT
PORTAL_ENDPOINT_ERROR_CONTACTS_ADD
PORTAL_ENDPOINT_ERROR_CONTACTS_EDIT
PORTAL_ENDPOINT_ERROR_CONTACTS_LIST
PORTAL_ENDPOINT_ERROR_CONTACTS_LOAD
PORTAL_ENDPOINT_ERROR_CONTACTS_MOVE
PORTAL_ENDPOINT_ERROR_CONTACTS_COPY
PORTAL_ENDPOINT_ERROR_CONTACTS_REMOVE
PORTAL_ENDPOINT_ERROR_FOLDERS_ADD
PORTAL_ENDPOINT_ERROR_FOLDERS_LIST
PORTAL_ENDPOINT_ERROR_FOLDERS_TAGS
PORTAL_ENDPOINT_ERROR_FOLDERS_REMOVE
PORTAL_ENDPOINT_ERROR_FOLDERS_RENAME
PORTAL_ENDPOINT_ERROR_MESSAGES_COMPOSE
PORTAL_ENDPOINT_ERROR_MESSAGES_COPY
PORTAL_ENDPOINT_ERROR_MESSAGES_FLAG
PORTAL_ENDPOINT_ERROR_MESSAGES_LIST
PORTAL_ENDPOINT_ERROR_MESSAGES_LOAD
PORTAL_ENDPOINT_ERROR_MESSAGES_MOVE
PORTAL_ENDPOINT_ERROR_MESSAGES_REMOVE
PORTAL_ENDPOINT_ERROR_MESSAGES_SEND
PORTAL_ENDPOINT_ERROR_MESSAGES_TAG
PORTAL_ENDPOINT_ERROR_MESSAGES_TAGS
PORTAL_ENDPOINT_ERROR_META

PORTAL_ENDPOINT_ERROR_SEARCH
PORTAL_ENDPOINT_ERROR_SETTINGS_IDENTITY
PORTAL_ENDPOINT_ERROR_SETTINGS_CHANGEPASS
PORTAL_ENDPOINT_ERROR_LOGOUT

Definition at line 72 of file portal.h.

Function Documentation

json_t* portal_config_collection (user_config_t * collection)

config.c

config.c

Parameters:

collection a pointer to a user config object that contains all the config options pairs to be serialized.

Returns:

NULL on failure, or a pointer to a json object containing the collection of user config options on success.

Definition at line 67 of file config.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `json_decref_d`, `json_object_d`, `json_object_set_new_d`, `log_error`, `log_options`, `M_LOG_CRITICAL`, `portal_config_entry()`, `st_char_get()`, `USER_CONF_STATUS_CRITICAL`, and `user_config_entry_t`.

Referenced by `portal_endpoint_config_load()`.

json_t* portal_config_entry (user_config_entry_t * entry)

Definition at line 40 of file config.c.

References `json_decref_d`, `json_pack_ex_d`, `log_pedantic`, `portal_config_entry_flags()`, `st_char_get()`, and `st_length_int()`.

Referenced by `portal_config_collection()`.

json_t* portal_config_entry_flags (user_config_entry_t * entry)

Get a user config entry's flags as a json object.

Parameters:

entry a pointer to the user config entry object to have its flags serialized.

Returns:

NULL on failure or a pointer to a json object storing the user's flags on success.

Definition at line 20 of file config.c.

References `json_array_append_new_d`, `json_array_d`, `json_decref_d`, `json_string_d`, and `USER_CONF_STATUS_CRITICAL`.

Referenced by `portal_config_entry()`.

json_t* portal_contact_detail_flags (contact_detail_t * *detail*)

contacts.c

contacts.c

Pack a json array representing the flags associated with a user contact entry detail.

Parameters:

detail a pointer to the user contact detail to have its flags queried.

Returns:

NULL on failure or a pointer to a json array containing strings describing the specified contact detail's flags.

Definition at line 21 of file contacts.c.

References CONTACT_DETAIL_FLAG_CRITICAL, json_array_append_new_d, json_array_d, and json_string_d.

Referenced by portal_contact_details().

json_t* portal_contact_details (contact_t * *contact*)

Retrieve all the details about a contact entry as a json object.

Parameters:

contact a pointer to the contact object whose details will be retrieved.

Returns:

a pointer to a json object containing the details of the specified contact entry.

Definition at line 40 of file contacts.c.

References contact_detail_t, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), json_array_d, json_decref_d, json_object_set_new_d, json_pack_ex_d, log_error, log_pedantic, portal_contact_detail_flags(), and st_char_get().

Referenced by portal_endpoint_contacts_load().

void portal_debug (connection_t * *con*)

A portal debug function that will be disabled and/or deleted completely in production.

Note:

The debug output is displayed LOCALLY in magmad's console.

Definition at line 3310 of file endpoint.c.

void portal_endpoint (connection_t * *con*)

endpoint.c

endpoint.c

Parameters:

con the connection object corresponding to the web client making the request.

Returns:

This function returns no value.

Definition at line 2958 of file endpoint.c.

References `command`, `con_addr_word()`, `con_secure()`, `command_t::function`, `connection_t::http`, `HTTP_ERROR_500`, `HTTP_MERGED`, `http_parse_context()`, `HTTP_PORTAL`, `http_print_301()`, `http_response_header()`, `json_integer_value_d`, `json_loads_d`, `json_object_get_d`, `JSON_RPC_2_ERROR_PARSE_MALFORMED`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_UNAVAIL`, `JSON_RPC_2_ERROR_SERVER_REQUEST`, `json_string_value_d`, `json_type_string_d`, `command_t::length`, `log_pedantic`, `magma`, `ns_length_get()`, `PLACER`, `magma_t::portal`, `portal_endpoint_compare()`, `portal_endpoint_error()`, `portal_methods`, `sess_create()`, `st_char_get()`, `command_t::string`, `uint64_conv_ns()`, and `magma_t::web`.

Referenced by `http_response()`.

void portal_endpoint_ad (connection_t * con)

Return advertising information for a portal "ad" json-rpc request.

^ finished / work area v ^ work area / stubs v

Note:

This function is not implemented and may be removed entirely in the future.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2763 of file endpoint.c.

References `JSON_RPC_2_ERROR_SERVER_METHOD_UNAVAIL`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_AD`, and `portal_validate_request()`.

void portal_endpoint_alert_acknowledge (connection_t * con)

Process user alert message acknowledgements in response to an "alert.acknowledge" json-rpc portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 360 of file endpoint.c.

References `count`, `connection_t::http`, `json_array_get_d`, `json_array_size_d`, `json_integer_value_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_pedantic`, `meta_data_acknowledge_alert()`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_ALERT_ACKNOWLEDGE`, `portal_endpoint_response()`, `portal_validate_request()`, `st_char_get()`, `st_length_get()`, `tran_commit()`, `tran_rollback()`, and `tran_start()`.

void portal_endpoint_alert_list (connection_t * con)

Get the list of a user's alert messages in response to an "alert.list" json-rpc portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 312 of file endpoint.c.

References `connection_t::http`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_free()`, `json_array_append_new_d`, `json_array_d`, `json_decref_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `log_pedantic`, `meta_data_fetch_alerts()`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_ALERT_LIST`, `portal_endpoint_response()`, `portal_validate_request()`, and `st_char_get()`.

void portal_endpoint_aliases (connection_t * con)

Return the list of a user account's aliases in response to an "aliases" json-rpc portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 429 of file endpoint.c.

References `connection_t::http`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `json_array_append_new_d`, `json_array_d`, `json_decref_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `json_true_d`, `log_pedantic`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_ALIASES`, `portal_endpoint_response()`, `portal_validate_request()`, and `st_char_get()`.

void portal_endpoint_attachments_add (connection_t * con)

Add an attachment to a message being composed in response to an "attachments.add" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 1878 of file endpoint.c.

References `attachment_t::attach_id`, `composition_t::attached`, `composition_t::attachments`, `attachment_t::filename`, `connection_t::http`, `inx_find()`, `inx_insert()`, `json_object_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_error`, `log_pedantic`, `M_TYPE_UINT64`, `mm_alloc()`, `mutex_lock()`, `mutex_unlock()`, `ns_length_get()`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_ATTACHMENTS_ADD`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `portal_validate_request()`, `sess_release_attachment()`, `st_char_get()`, `st_import()`, `st_length_get()`, `multi_t::u64`, and `multi_t::val`.

void portal_endpoint_attachments_progress (connection_t * con)

Definition at line 2823 of file endpoint.c.

References `con_write_st()`, `connection_t::http`, `HTTP_ERROR_500`, `http_response_header()`, `PLACER`, `PORTAL_ENDPOINT_ERROR_ATTACHMENTS_PROGRESS`, `st_aprint()`, `st_free()`, and `st_length_get()`.

void portal_endpoint_attachments_remove (connection_t * con)

Remove an attachment from a message being composed in response to an "attachments.remove" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 1956 of file endpoint.c.

References `composition_t::attachments`, `connection_t::http`, `inx_delete()`, `inx_find()`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_error`, `log_pedantic`, `M_TYPE_UINT64`, `mutex_lock()`, `mutex_unlock()`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_ATTACHMENTS_REMOVE`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `portal_validate_request()`, `st_char_get()`, `st_length_get()`, `multi_t::u64`, and `multi_t::val`.

void portal_endpoint_auth (connection_t * con)

Obtain credentials and log a user into portal session in response to an "auth" json-rpc portal request.

Parameters:

con a pointer to the connection object of the user attempting to log in.

Returns:

This function returns no value.

Definition at line 200 of file endpoint.c.

References `credential_t::auth`, `con_secure()`, `credential_alloc_auth()`, `meta_user_t::flags`, `connection_t::http`, `HTTP_COOKIE_SET`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `meta_user_t::lock_status`, `log_pedantic`, `meta_get()`, `META_GET_CONTACTS`, `META_GET_FOLDERS`, `META_GET_MESSAGES`, `META_PROT_WEB`, `meta_remove()`, `meta_user_ref_add()`, `META_USER_SSL`, `NULLER`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_AUTH`, `PORTAL_ENDPOINT_ERROR_MODE`, `portal_endpoint_response()`, `SESSION_STATE_AUTHENTICATED`, `SESSION_STATE_NEUTRAL`, `st_char_get()`, and `meta_user_t::username`.

int_t portal_endpoint_compare (const void * compare, const void * command)

Internal bsearch() comparison function for dispatching the proper portal handler.

Parameters:

compare a pointer to a **command_t** object with a portal method name for string comparison.

command a pointer to a **command_t** object with a portal method name for string comparison.

Returns:

Definition at line 42 of file endpoint.c.

References `command_t::length`, `PLACER`, `st_cmp_ci_eq()`, and `command_t::string`.

Referenced by `portal_endpoint()`, and `portal_endpoint_sort()`.

void portal_endpoint_config_edit (connection_t * con)

Retrieve all of a user's config options and return them to the remote client in response to a portal "contacts.load" json-rpc request.

Note:

Requires a user to be authenticated; failure will be communicated to the client via a json response.

Parameters:

con a pointer to a connection object over which the json response will be sent.

Returns:

This function returns no value.

HIGH: We aren't checking/validating the config entries! And we should probably do our own duplication checks to avoid placing unnecessary load on the database.

Definition at line 2614 of file endpoint.c.

References `content`, `connection_t::http`, `json_object_iter_d`, `json_object_iter_key_d`, `json_object_iter_next_d`, `json_object_iter_value_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `json_string_value_d`, `NULLER`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONFIG_EDIT`, `portal_endpoint_response()`, `portal_validate_request()`, `status`, `user_config_create()`, `user_config_edit()`, `user_config_free()`, and `user_config_t`.

void portal_endpoint_config_load (connection_t * con)

Retrieve all of a user's config options and return them to the remote client in response to a portal "contacts.load" json-rpc request.

Note:

Requires a user to be authenticated; failure will be communicated to the client via a json response.

Parameters:

con a pointer to a connection object over which the json response will be sent.

Returns:

This function returns no value.

Definition at line 2673 of file endpoint.c.

References `connection_t::http`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `portal_config_collection()`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONFIG_LOAD`, `portal_endpoint_response()`, `portal_validate_request()`, `user_config_create()`, `user_config_free()`, and `user_config_t`.

void portal_endpoint_contacts_add (connection_t * con)

Add a contact in response to a "contacts.add" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

HIGH: We aren't checking/validating the contact details! And we should probably do our own duplication checks to avoid placing unnecessary load on the database.

Definition at line 2422 of file endpoint.c.

References `contact_create()`, `contact_edit()`, `contact_free()`, `contact_t`, `contact_validate_detail()`, `contact_validate_name()`, `content`, `connection_t::http`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_insert()`, `json_object_get_d`, `json_object_iter_d`, `json_object_iter_key_d`, `json_object_iter_next_d`, `json_object_iter_value_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_string_value_d`, `json_type_string_d`, `json_unpack_ex_d`, `log_pedantic`, `M_TYPE_UINT64`, `magma_folder_find_number()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_user_wlock()`, `NULLER`, `OBJECT_CONTACTS`, `PLACER`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONSTRAINT_VIOLATION`, `PORTAL_ENDPOINT_ERROR_CONTACTS_ADD`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `portal_validate_request()`, `sess_serial_check()`, `st_char_get()`, `st_cmp_ci_eq()`, `st_cmp_cs_eq()`, `st_length_get()`, `status`, `multi_t::u64`, and `multi_t::val`.

void portal_endpoint_contacts_copy (connection_t * con)

Note:

If the source and target folder are equal, a copy will be made with a different name.

Definition at line 2128 of file endpoint.c.

References `contact_create()`, `contact_detail_t`, `contact_edit()`, `contact_find_number()`, `contact_free()`, `contact_t`, `connection_t::http`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_insert()`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_error`, `log_pedantic`, `M_TYPE_UINT64`, `magma_folder_find_number()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_CONTACTS`, `PLACER`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONSTRAINT_VIOLATION`, `PORTAL_ENDPOINT_ERROR_CONTACTS_COPY`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `portal_validate_request()`, `sess_serial_check()`, `st_char_get()`, `st_cleanup()`, `st_length_get()`, `st_merge`, `multi_t::u64`, and `multi_t::val`.

void portal_endpoint_contacts_edit (connection_t * con)

HIGH: We aren't checking/validating the contact details! And we should probably do our own duplication checks to avoid placing unnecessary load on the database.

Definition at line 2538 of file endpoint.c.

References `contact_edit()`, `contact_find_number()`, `contact_t`, `content`, `connection_t::http`, `json_object_iter_d`, `json_object_iter_key_d`, `json_object_iter_next_d`, `json_object_iter_value_d`, `json_object_size_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_string_value_d`, `json_unpack_ex_d`, `log_pedantic`, `magma_folder_find_number()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_user_wlock()`, `NULLER`, `OBJECT_CONTACTS`, `portal_endpoint_error()`,

PORTAL_ENDPOINT_ERROR_CONTACTS_EDIT,
PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION,
PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), portal_validate_request(),
sess_serial_check(), st_char_get(), st_length_get(), and status.

void portal_endpoint_contacts_list (connection_t * con)

List a user's contacts in response to a "contacts.list" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2353 of file endpoint.c.

References contact_detail_t, contact_t, connection_t::http, inx_cursor_alloc(), inx_cursor_free(),
inx_cursor_value_next(), json_array_append_new_d, json_array_d, json_decref_d, json_object_set_new_d,
json_pack_ex_d, JSON_RPC_2_ERROR_SERVER_INTERNAL,
JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_string_d, json_unpack_ex_d, log_pedantic,
magma_folder_find_number(), portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_CONTACTS_LIST,
PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), portal_validate_request(),
st_char_get(), and st_length_get().

void portal_endpoint_contacts_load (connection_t * con)

Return information for a portal "contacts.load" json-rpc request.

Note:

This function returns the all the stored details about a specified contact entry.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2709 of file endpoint.c.

References contact_t, connection_t::http, inx_find(), JSON_RPC_2_ERROR_SERVER_INTERNAL,
JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic,
M_TYPE_UINT64, magma_folder_find_number(), portal_contact_details(), portal_endpoint_error(),
PORTAL_ENDPOINT_ERROR_CONTACTS_LOAD, PORTAL_ENDPOINT_ERROR_REFERENCE,
portal_endpoint_response(), portal_validate_request(), st_char_get(), st_length_get(), multi_t::u64, and
multi_t::val.

void portal_endpoint_contacts_move (connection_t * con)

Move a user's contact entry to another contacts folder in response to a "contacts.move" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2235 of file endpoint.c.

References `contact_find_number()`, `contact_move()`, `contact_t`, `connection_t::http`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_pedantic`, `magma_folder_find_number()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_CONTACTS`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONTACTS_MOVE`, `PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `portal_validate_request()`, `sess_serial_check()`, `st_char_get()`, `st_length_get()`, and `status`.

void portal_endpoint_contacts_remove (connection_t * con)

Remove a user's contact entry in response to a "contacts.remove" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2298 of file endpoint.c.

References `contact_delete()`, `contact_find_number()`, `contact_t`, `connection_t::http`, `inx_delete()`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_pedantic`, `M_TYPE_UINT64`, `magma_folder_find_number()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_CONTACTS`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONTACTS_REMOVE`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `portal_validate_request()`, `sess_serial_check()`, `st_char_get()`, `st_length_get()`, `status`, `multi_t::u64`, and `multi_t::val`.

void portal_endpoint_cookies (connection_t * con)

Return whether or not a user is using cookies, in response to a "cookies" json-rpc portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 479 of file endpoint.c.

References `connection_t::http`, `PLACER`, `PORTAL_ENDPOINT_ERROR_COOKIES`, `portal_endpoint_response()`, `portal_validate_request()`, and `st_cmp_ci_starts()`.

void portal_endpoint_error (connection_t * con, int_t http_code, int_t error_code, chr_t * message)

Return a json-rpc error response to the remote client.

Parameters:

con a pointer to the connection object across which the response will be sent.
http_code the http response code to be sent to the remote client in the response header.
error_code the numerical error code to be encoded in the json-rpc error message.
message a descriptive error string to be encoded in the json-rpc error message.

Returns:

This function returns no value.

Definition at line 130 of file endpoint.c.

References `con_write_st()`, `connection_t::http`, `HTTP_ERROR_500`, `http_response_header()`, `json_decref_d`, `json_dumps_d`, `json_pack_ex_d`, `log_options`, `log_pedantic`, `M_LOG_CRITICAL`, `M_LOG_STACK_TRACE`, `magma`, `ns_append()`, `ns_free()`, `ns_length_get()`, `NULLER`, `PLACER`, `magma_t::portal`, and `magma_t::web`.

Referenced by `portal_endpoint()`, `portal_endpoint_ad()`, `portal_endpoint_alert_acknowledge()`, `portal_endpoint_alert_list()`, `portal_endpoint_aliases()`, `portal_endpoint_attachments_add()`, `portal_endpoint_attachments_remove()`, `portal_endpoint_auth()`, `portal_endpoint_config_edit()`, `portal_endpoint_config_load()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_contacts_edit()`, `portal_endpoint_contacts_list()`, `portal_endpoint_contacts_load()`, `portal_endpoint_contacts_move()`, `portal_endpoint_contacts_remove()`, `portal_endpoint_folders_add()`, `portal_endpoint_folders_list()`, `portal_endpoint_folders_remove()`, `portal_endpoint_folders_rename()`, `portal_endpoint_folders_tags()`, `portal_endpoint_messages_compose()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_load()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_send()`, `portal_endpoint_messages_tag()`, `portal_endpoint_messages_tags()`, `portal_endpoint_scrape()`, `portal_endpoint_scrape_add()`, `portal_folder_contacts_add()`, `portal_folder_contacts_remove()`, `portal_folder_mail_add()`, `portal_folder_mail_remove()`, `portal_meta()`, `portal_settings_changepass()`, `portal_settings_identity()`, and `portal_validate_request()`.

void portal_endpoint_folders_add (connection_t * con)

Create a new folder in response to a "folders.add" json-rpc portal request.

Note:

Currently only contexts for new mail folders and contacts folders are supported.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 502 of file endpoint.c.

References `count`, `connection_t::http`, `json_object_size_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_pedantic`, `ns_length_get()`, `NULLER`, `PORTAL_ENDPOINT_CONTEXT_CONTACTS`, `PORTAL_ENDPOINT_CONTEXT_INVALID`, `PORTAL_ENDPOINT_CONTEXT_MAIL`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_FOLDERS_ADD`, `PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD`, `portal_folder_contacts_add()`, `portal_folder_mail_add()`, `portal_parse_context()`, `portal_validate_request()`, `st_char_get()`, `st_free()`, `st_import()`, and `st_length_get()`.

void portal_endpoint_folders_list (connection_t * con)

Return a list of a user's folders in response to a "folders.list" json-rpc portal request.

Note:

Currently only contexts for new mail folders and contacts folders are supported.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

LOW: User created folder names `_could_` have a NULL byte. How should we handle that?

LOW: User created folder names `_could_` have a NULL byte. How should we handle that?

Definition at line 563 of file endpoint.c.

References `meta_folder_t::foldernum`, `connection_t::http`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `json_array_append_new_d`, `json_array_d`, `json_decref_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_pedantic`, `magma_folder_t`, `meta_folder_t::name`, `NULLER`, `meta_folder_t::parent`, `PLACER`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_FOLDERS_LIST`, `PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD`, `portal_endpoint_response()`, `portal_validate_request()`, `st_char_get()`, `st_cmp_ci_eq()`, and `st_length_get()`.

void portal_endpoint_folders_remove (connection_t * con)

Remove a user's folder in response to a json-rpc "folders.remove" portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 646 of file endpoint.c.

References `connection_t::http`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_pedantic`, `NULLER`, `PORTAL_ENDPOINT_CONTEXT_CONTACTS`, `PORTAL_ENDPOINT_CONTEXT_INVALID`, `PORTAL_ENDPOINT_CONTEXT_MAIL`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_FOLDERS_REMOVE`, `PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD`, `portal_folder_contacts_remove()`, `portal_folder_mail_remove()`, `portal_parse_context()`, `portal_validate_request()`, `st_char_get()`, and `st_length_get()`.

void portal_endpoint_folders_rename (connection_t * con)

Rename a user's folder in response to a json-rpc "folders.rename" portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

HIGH: Use the context and act appropriately!

Definition at line 687 of file endpoint.c.

References contact_folder_rename(), connection_t::http, imap_folder_rename(),
JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS,
json_unpack_ex_d, log_pedantic, magma_folder_find_number(), magma_folder_name(), magma_folder_t,
meta_folders_by_number(), meta_folders_name(), meta_user_unlock(), meta_user_wlock(), ns_length_get(),
NULLER, OBJECT_CONTACTS, OBJECT_FOLDERS, PORTAL_ENDPOINT_CONTEXT_CONTACTS,
PORTAL_ENDPOINT_CONTEXT_INVALID, PORTAL_ENDPOINT_CONTEXT_MAIL,
portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_CONSTRAINT_VIOLATION,
PORTAL_ENDPOINT_ERROR_FOLDERS_RENAME,
PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD, PORTAL_ENDPOINT_ERROR_REFERENCE,
portal_endpoint_response(), portal_parse_context(), portal_validate_request(), serial_get(), serial_increment(),
sess_trigger(), st_char_get(), st_free(), st_import(), and st_length_get().

void portal_endpoint_folders_tags (connection_t * con)

HIGH: Right now we only have indexes with the mail folders so other context types generate an empty array result.

Definition at line 832 of file endpoint.c.

References meta_stats_tag_t::count, connection_t::http, inx_cursor_alloc(), inx_cursor_free(),
inx_cursor_value_next(), inx_free(), json_decref_d, json_integer_d, json_object_d, json_object_set_new_d,
JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS,
json_unpack_ex_d, log_pedantic, meta_folders_by_number(), meta_folders_stats_tags(), meta_user_rlock(),
meta_user_unlock(), NULLER, PORTAL_ENDPOINT_CONTEXT_MAIL, portal_endpoint_error(),
PORTAL_ENDPOINT_ERROR_FOLDERS_TAGS, PORTAL_ENDPOINT_ERROR_REFERENCE,
portal_endpoint_response(), portal_parse_context(), portal_validate_request(), st_char_get(), st_length_get(),
stats, and meta_stats_tag_t::tag.

void portal_endpoint_logout (connection_t * con)

Log a user out of a portal session in response to a "logout" json-rpc portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

Definition at line 290 of file endpoint.c.

References connection_t::http, HTTP_COOKIE_DELETE, META_PROT_WEB, meta_user_ref_dec(),
PORTAL_ENDPOINT_ERROR_LOGOUT, portal_endpoint_response(), portal_validate_request(), and
SESSION_STATE_TERMINATED.

void portal_endpoint_messages_compose (connection_t * con)

Compose a new message in response to a "messages.compose" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 1818 of file endpoint.c.

References `composition_t::attachments`, `composition_t::comp_id`, `connection_t::http`, `inx_alloc()`, `inx_find()`, `inx_insert()`, `json_object_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `log_error`, `M_INX_LINKED`, `M_TYPE_UINT64`, `mm_alloc()`, `mutex_lock()`, `mutex_unlock()`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_MESSAGES_COMPOSE`, `portal_endpoint_response()`, `portal_validate_request()`, `sess_release_attachment()`, `sess_release_composition()`, `multi_t::u64`, and `multi_t::val`.

void portal_endpoint_messages_copy (connection_t * con)

HIGH: If a multiple message copy fails in the middle, go back and remove any messages that may have been copied before the error.

Definition at line 990 of file endpoint.c.

References `count`, `meta_message_t::foldernum`, `connection_t::http`, `inx_find()`, `json_array_append_new_d`, `json_array_d`, `json_array_get_d`, `json_array_size_d`, `json_decref_d`, `json_integer_value_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_pedantic`, `M_TYPE_UINT64`, `meta_message_t::messagenum`, `meta_folders_by_number()`, `META_LOCKED`, `meta_messages_copier()`, `meta_messages_update_sequences()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_MESSAGES`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_MESSAGES_COPY`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `portal_validate_request()`, `serial_get()`, `serial_increment()`, `sess_trigger()`, `st_char_get()`, `st_length_get()`, `multi_t::u64`, and `multi_t::val`.

void portal_endpoint_messages_flag (connection_t * con)

HIGH: This function needs restructuring pretty badly. We are setting collection to NULL so we can add a cleanup call below. But if the function works as intended the reference is stolen by the response. but if an error occurs and the we don't return a response the collection ends up as a memory leak without the cleanup call below.

TODO: Were holding onto the user lock while waiting on the response to be sent over the wire; and this function could probably use a rethink.

HIGH: See the sister comment a few lines back!

Definition at line 1079 of file endpoint.c.

References `count`, `meta_message_t::foldernum`, `connection_t::http`, `inx_alloc()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_find()`, `inx_free()`, `inx_insert()`, `json_array_append_new_d`, `json_array_d`, `json_array_get_d`, `json_array_size_d`, `json_decref_d`, `json_integer_value_d`, `json_object_size_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS`, `json_unpack_ex_d`, `log_pedantic`, `M_INX_LINKED`, `M_TYPE_UINT64`, `MAIL_STATUS_SYSTEM_FLAGS`, `MAIL_STATUS_USER_FLAGS`, `meta_message_t::messagenum`, `meta_data_flags_add()`, `meta_data_flags_remove()`, `meta_data_flags_replace()`, `meta_folders_by_number()`, `meta_user_rlock()`, `meta_user_unlock()`, `meta_user_wlock()`, `NULLER`, `OBJECT_MESSAGES`, `PLACER`,

PORTAL_ENDPOINT_ACTION_ADD, PORTAL_ENDPOINT_ACTION_LIST,
 PORTAL_ENDPOINT_ACTION_REMOVE, PORTAL_ENDPOINT_ACTION_REPLACE,
 portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION,
 PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD,
 PORTAL_ENDPOINT_ERROR_MESSAGES_FLAG, PORTAL_ENDPOINT_ERROR_REFERENCE,
 PORTAL_ENDPOINT_ERROR_SYSTEM_FLAG, portal_endpoint_response(), portal_message_flags_array(),
 portal_parse_flags(), portal_validate_request(), serial_get(), serial_increment(), sess_trigger(), st_char_get(),
 st_cmp_ci_eq(), st_length_get(), meta_message_t::status, multi_t::u64, and multi_t::val.

void portal_endpoint_messages_list (connection_t * con)

Retrieve a list of the user's messages in response to a json-rpc "messages.list" portal request.

Parameters:

con a pointer to the connection object of the requesting user.

Returns:

This function returns no value.

LOW: Add the ability to track the recipient email address for a message, even if its not provided in the To field.

LOW: Add snippet support.

Definition at line 905 of file endpoint.c.

References ar_field_st(), ar_length_get(), count, meta_message_t::created, meta_message_t::foldernum, connection_t::http, inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), json_array_append_new_d, json_array_d, json_decref_d, json_pack_ex_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_string_d, json_unpack_ex_d, log_pedantic, mail_header_fetch_cleaned(), mail_load_header(), meta_message_t::messagenum, PLACER, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_MESSAGES_LIST, portal_endpoint_response(), portal_message_flags_array(), portal_validate_request(), meta_message_t::size, st_char_get(), st_cleanup(), st_free(), st_import(), st_length_get(), and meta_message_t::tags.

void portal_endpoint_messages_load (connection_t * con)

Definition at line 1712 of file endpoint.c.

References data, connection_t::http, inx_find(), json_decref_d, json_object_d, json_object_set_new_d, json_object_size_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic, M_TYPE_UINT64, mail_destroy(), mail_load_message(), mail_mime_update(), portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION, PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD, PORTAL_ENDPOINT_ERROR_MESSAGES_LOAD, PORTAL_ENDPOINT_ERROR_READ, PORTAL_ENDPOINT_ERROR_REFERENCE, PORTAL_ENDPOINT_MESSAGE_ATTACHMENTS, PORTAL_ENDPOINT_MESSAGE_BODY, PORTAL_ENDPOINT_MESSAGE_HEADER, PORTAL_ENDPOINT_MESSAGE_INFO, PORTAL_ENDPOINT_MESSAGE_META, PORTAL_ENDPOINT_MESSAGE_SECURITY, PORTAL_ENDPOINT_MESSAGE_SERVER, PORTAL_ENDPOINT_MESSAGE_SOURCE, portal_endpoint_response(), portal_message_attachments(), portal_message_body(), portal_message_header(), portal_message_info(), portal_message_meta(), portal_message_security(), portal_message_server(), portal_message_source(), portal_parse_sections(), portal_validate_request(), connection_t::server, st_char_get(), st_length_get(), multi_t::u64, and multi_t::val.

void portal_endpoint_messages_move (connection_t * con)

HIGH: If a multiple message copy fails in the middle, go back and remove any messages that may have been copied before the error.

Definition at line 1279 of file endpoint.c.

References count, meta_message_t::foldernum, connection_t::http, inx_find(), json_array_get_d, json_array_size_d, json_integer_value_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic, M_TYPE_UINT64, meta_folders_by_number(), META_LOCKED, meta_messages_mover(), meta_messages_update_sequences(), meta_user_unlock(), meta_user_wlock(), OBJECT_MESSAGES, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION, PORTAL_ENDPOINT_ERROR_MESSAGES_MOVE, PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), portal_validate_request(), serial_get(), serial_increment(), sess_trigger(), st_char_get(), st_length_get(), multi_t::u64, and multi_t::val.

void portal_endpoint_messages_remove (connection_t * con)

Remove a user's mail message in response to a "messages.remove" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 1375 of file endpoint.c.

References count, meta_message_t::foldernum, connection_t::http, inx_delete(), inx_find(), json_array_get_d, json_array_size_d, json_integer_value_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_pedantic, M_TYPE_UINT64, mail_remove_message(), meta_message_t::messagenum, meta_folders_by_number(), meta_messages_update_sequences(), meta_user_unlock(), meta_user_wlock(), OBJECT_MESSAGES, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_MESSAGES_REMOVE, PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), portal_validate_request(), serial_get(), serial_increment(), meta_message_t::server, sess_trigger(), meta_message_t::size, st_char_get(), st_length_get(), multi_t::u64, and multi_t::val.

void portal_endpoint_messages_send (connection_t * con)

Send a composed message in response to a "messages.send" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2012 of file endpoint.c.

References composition_t::attachments, connection_t::http, inx_delete(), inx_find(), inx_free(), json_object_size_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_unpack_ex_d, log_error, log_pedantic,

M_TYPE_UINT64, mutex_lock(), mutex_unlock(), NULLER, portal_endpoint_error(),
 PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION,
 PORTAL_ENDPOINT_ERROR_MESSAGES_SEND, PORTAL_ENDPOINT_ERROR_REFERENCE,
 portal_endpoint_response(), portal_outbound_checks(), portal_parse_json_str_array(),
 portal_smtp_create_data(), portal_smtp_relay_message(), portal_validate_request(), st_char_get(), st_free(),
 st_length_get(), multi_t::u64, and multi_t::val.

void portal_endpoint_messages_tag (connection_t * con)

Perform a tag operation on a user's message, in response to a "messages.tag" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

LOW: We don't need to add the flag to every message. Just the ones that don't already have the flag.

LOW: Like the line, were not checking whether the message even needs to have the flag removed. Were also not handling remove requests that result in a message having no tags.

If this is going supposed to be a replace operation, we truncate all of the message tags so we can insert the replacements below.

TODO: This is ugly. Were rebuilding the tags array from the database on each iteration because its easier than trying to figure out which slot needs to be removed.

TODO: If the update is triggered before the tagged flag is added to the database the refresh might not pull the data for who recently got tagged! We need to need to look for this type of sequence bug elsewhere too.

TODO: Were holding onto the user lock while were waiting on the response to be sent over the wire; and this function could probably use a rethink.

Definition at line 1515 of file endpoint.c.

References ar_field_st(), ar_free(), ar_length_get(), meta_message_t::foldernum, connection_t::http,
 inx_alloc(), inx_cursor_alloc(), inx_cursor_free(), inx_cursor_value_next(), inx_find(), inx_free(), inx_insert(),
 json_array_append_new_d, json_array_d, json_array_get_d, json_array_size_d, json_decref_d,
 json_integer_value_d, json_object_size_d, json_pack_ex_d, JSON_RPC_2_ERROR_SERVER_INTERNAL,
 JSON_RPC_2_ERROR_SERVER_METHOD_PARAMS, json_string_d, json_string_value_d,
 json_unpack_ex_d, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, MAIL_STATUS_TAGGED,
 meta_message_t::messagenum, meta_data_delete_tag(), meta_data_fetch_message_tags(),
 meta_data_flags_add(), meta_data_flags_remove(), meta_data_insert_tag(), meta_data_truncate_tags(),
 meta_folders_by_number(), meta_user_rlock(), meta_user_serial_check(), meta_user_unlock(),
 meta_user_wlock(), NULLER, OBJECT_MESSAGES, PLACER, PORTAL_ENDPOINT_ACTION_ADD,
 PORTAL_ENDPOINT_ACTION_LIST, PORTAL_ENDPOINT_ACTION_REMOVE,
 PORTAL_ENDPOINT_ACTION_REPLACE, portal_endpoint_error(),
 PORTAL_ENDPOINT_ERROR_ILLEGAL_COMBINATION,
 PORTAL_ENDPOINT_ERROR_INVALID_KEYWORD,
 PORTAL_ENDPOINT_ERROR_MESSAGES_TAG, PORTAL_ENDPOINT_ERROR_REFERENCE,
 portal_endpoint_response(), portal_validate_request(), st_char_get(), st_cmp_ci_eq(), st_length_get(),
 meta_message_t::tags, multi_t::u64, and multi_t::val.

void portal_endpoint_messages_tags (connection_t * con)

Get all of the message tags used by a specified user in response to a "messages.tags" json-rpc portal request.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 1464 of file endpoint.c.

References `connection_t::http`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `inx_free()`, `json_array_append_new_d`, `json_array_d`, `json_decref_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `json_string_d`, `log_error`, `log_pedantic`, `meta_data_fetch_all_tags()`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_MESSAGES_TAGS`, `portal_endpoint_response()`, `portal_validate_request()`, and `st_char_get()`.

void portal_endpoint_response (connection_t * con, chr_t * format, ...)

Generate a json-rpc 2.0 response to a portal request.

See also:

`json_vpack_ex()`

Note:

This function indents the json response if specified in the configuration, and also automatically decreases the reference count of any json object that was packed for the reply.

Parameters:

con a pointer to the connection object across which the portal response will be sent.

format a pointer to a format string specifying the construction of the json-rpc response.

... a variable arguments style list of parameters to be passed to the json packing function.

Returns:

This function returns no value.

Definition at line 168 of file endpoint.c.

References `con_write_st()`, `connection_t::http`, `HTTP_ERROR_500`, `http_response_header()`, `json_decref_d`, `json_dumps_d`, `json_vpack_ex_d`, `log_pedantic`, `magma`, `ns_append()`, `ns_free()`, `ns_length_get()`, `NULLER`, `PLACER`, `magma_t::portal`, and `magma_t::web`.

Referenced by `portal_endpoint_alert_acknowledge()`, `portal_endpoint_alert_list()`, `portal_endpoint_aliases()`, `portal_endpoint_attachments_add()`, `portal_endpoint_attachments_remove()`, `portal_endpoint_auth()`, `portal_endpoint_config_edit()`, `portal_endpoint_config_load()`, `portal_endpoint_contacts_add()`, `portal_endpoint_contacts_copy()`, `portal_endpoint_contacts_edit()`, `portal_endpoint_contacts_list()`, `portal_endpoint_contacts_load()`, `portal_endpoint_contacts_move()`, `portal_endpoint_contacts_remove()`, `portal_endpoint_cookies()`, `portal_endpoint_folders_list()`, `portal_endpoint_folders_rename()`, `portal_endpoint_folders_tags()`, `portal_endpoint_logout()`, `portal_endpoint_messages_compose()`, `portal_endpoint_messages_copy()`, `portal_endpoint_messages_flag()`, `portal_endpoint_messages_list()`, `portal_endpoint_messages_load()`, `portal_endpoint_messages_move()`, `portal_endpoint_messages_remove()`, `portal_endpoint_messages_send()`, `portal_endpoint_messages_tag()`, `portal_endpoint_messages_tags()`, `portal_folder_contacts_add()`, `portal_folder_contacts_remove()`, `portal_folder_mail_add()`, `portal_folder_mail_remove()`, `portal_meta()`, `portal_settings_changepass()`, and `portal_settings_identity()`.

void portal_endpoint_scape (connection_t * con)

Note:

This function is not implemented and may be removed entirely in the future.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2812 of file endpoint.c.

References JSON_RPC_2_ERROR_SERVER_METHOD_UNAVAIL, portal_endpoint_error(),
PORTAL_ENDPOINT_ERROR_AD, and portal_validate_request().

void portal_endpoint_scape_add (connection_t * con)

Note:

This function is not implemented and may be removed entirely in the future.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2796 of file endpoint.c.

References JSON_RPC_2_ERROR_SERVER_METHOD_UNAVAIL, portal_endpoint_error(),
PORTAL_ENDPOINT_ERROR_AD, and portal_validate_request().

void portal_endpoint_search (connection_t * con)

Definition at line 2774 of file endpoint.c.

References con_write_st(), connection_t::http, HTTP_ERROR_500, http_response_header(), PLACER,
PORTAL_ENDPOINT_ERROR_SEARCH, st_aprint(), st_free(), and st_length_get().

void portal_endpoint_sort (void)

Sort the web portal JSON command table to be ready for binary searches.

TODO: We use session locks to synchronize access to the user context. Should we also be using mailbox cluster locks? Cluster level locking might be appropriate for operations that delete data likes folders.remove and messages.remove.

Returns:

This function returns no value.

Definition at line 26 of file endpoint.c.

References portal_endpoint_compare(), and portal_methods.

Referenced by protocol_init().

void portal_folder_contacts_add (connection_t * con, stringer_t * name, uint64_t parent)

folders.c

folders.c

Note:

If parent is 0, then the new folder is assumed to be a root folder. This function returns no value, but returns the appropriate portal json-rpc response directly.

Parameters:

con a pointer to the connection object across which the add folder response will be sent.

name a managed string containing the name of the new folder.

parent the numerical id of the folder that will be the parent of the new folder (or 0 to specify a root folder).

Returns:

This function returns no value.

Definition at line 86 of file folders.c.

References `contact_folder_create()`, `connection_t::http`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `magma_folder_find_name()`, `magma_folder_find_number()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_CONTACTS`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_FOLDERS_ADD`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, `sess_serial_check()`, and `st_cleanup()`.

Referenced by `portal_endpoint_folders_add()`.

void portal_folder_contacts_remove (connection_t * con, uint64_t foldernum)

Remove a user's contacts folder.

Parameters:

con a pointer to the connection object across which the remove folder response will be sent.

foldernum the numerical id of the contacts folder that will be removed.

Returns:

This function returns no value.

Definition at line 205 of file folders.c.

References `contact_folder_remove()`, `connection_t::http`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `magma_folder_find_number()`, `meta_user_unlock()`, `meta_user_wlock()`, `OBJECT_CONTACTS`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_CONSTRAINT_VIOLATION`, `PORTAL_ENDPOINT_ERROR_FOLDERS_REMOVE`, `PORTAL_ENDPOINT_ERROR_REFERENCE`, `portal_endpoint_response()`, and `sess_serial_check()`.

Referenced by `portal_endpoint_folders_remove()`.

void portal_folder_mail_add (connection_t * con, stringer_t * name, uint64_t parent)

Add a new mail folder for a user.

Note:

If parent is 0, then the new folder is assumed to be a root folder. This function returns no value, but returns the appropriate portal json-rpc response directly.

Parameters:

con a pointer to the connection object across which the add folder response will be sent.

name a managed string containing the name of the new folder.

parent the numerical id of the folder that will be the parent of the new folder (or 0 to specify a root folder).

Returns:

This function returns no value.

Definition at line 25 of file folders.c.

References meta_folder_t::foldernum, connection_t::http, imap_folder_create(), JSON_RPC_2_ERROR_SERVER_INTERNAL, meta_folders_by_name(), meta_folders_by_number(), meta_folders_name(), meta_user_unlock(), meta_user_wlock(), OBJECT_FOLDERS, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_FOLDERS_ADD, PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), sess_serial_check(), st_cleanup(), and st_merge.

Referenced by portal_endpoint_folders_add().

void portal_folder_mail_remove (connection_t * con, uint64_t foldernum)

Remove a user's mail folder.

Parameters:

con a pointer to the connection object across which the remove folder response will be sent.

foldernum the numerical id of the mail folder that will be removed.

Returns:

This function returns no value.

Definition at line 152 of file folders.c.

References connection_t::http, imap_folder_remove(), JSON_RPC_2_ERROR_SERVER_INTERNAL, meta_folders_by_number(), meta_folders_name(), meta_user_unlock(), meta_user_wlock(), OBJECT_FOLDERS, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_CONSTRAINT_VIOLATION, PORTAL_ENDPOINT_ERROR_FOLDERS_REMOVE, PORTAL_ENDPOINT_ERROR_REFERENCE, portal_endpoint_response(), sess_serial_check(), and st_cleanup().

Referenced by portal_endpoint_folders_remove().

json_t* portal_message_attachments (meta_message_t * meta, mail_message_t * data)

messages.c

LOW: We also need to detect messages that have no readable content so the entire body is just a blob.

Create a function to search for the filename. Common locations are: Content-Type: image/png; name="webmail-php-download.png" Content-Disposition: attachment; filename="webmail-php-download.png"

Definition at line 169 of file messages.c.

References ar_field_ptr(), ar_length_get(), mail_mime_t::body, mail_mime_t::children, count, mail_mime_t::header, json_array_append_new_d, json_array_d, json_decref_d, json_pack_ex_d, log_pedantic, mail_mime_type_group(), mail_mime_type_sub(), MESSAGE_TYPE_HTML, MESSAGE_TYPE_MULTI_MIXED, MESSAGE_TYPE_PLAIN, mail_message_t::mime, PLACER, st_aprint(), st_char_get(), st_cleanup(), st_length_get(), st_merge, mail_mime_t::type, and type().

Referenced by portal_endpoint_messages_load().

json_t* portal_message_body (meta_message_t * meta, mail_message_t * data)

TODO: I consider it a miracle whenever the above logic actually manages to select the message content. But if it doesn't we fall through to this fixed error message, at least until we can improve the selection process.

HIGH: Because JSON wants NULLERS, and were using PLACEHOLDERS, its printing out the entire message, not just the section of interest.

Definition at line 113 of file messages.c.

References ar_field_ptr(), mail_mime_t::body, mail_mime_t::children, CONTIGUOUS, HEAP, json_pack_ex_d, log_pedantic, MAIL_MIME_RECURSION_LIMIT, MESSAGE_TYPE_HTML, MESSAGE_TYPE_MULTI_ALTERNATIVE, MESSAGE_TYPE_MULTI_MIXED, MESSAGE_TYPE_MULTI_RELATED, MESSAGE_TYPE_MULTI_UNKOWN, MESSAGE_TYPE_PLAIN, mail_message_t::mime, NULLER_T, pl_char_get(), pl_length_get(), PLACER, st_char_get(), st_cleanup(), st_dupe_opts(), st_nullify(), mail_mime_t::type, and type().

Referenced by portal_endpoint_messages_load().

json_t* portal_message_flags_array (meta_message_t * meta)

flags.c

flags.c

TODO: The messages.load method uses the flags/tags helper functions, but the messages.list and the messages.tags/flags methods still need to be updated.

Parameters:

meta a pointer to the meta message object of the mail message to have its flags parsed.

Returns:

NULL on failure, or a pointer to the json object of the specified mail message's flags.

Definition at line 23 of file flags.c.

References json_array_append_new_d, json_array_d, json_string_d, MAIL_MARK_BLACKHOLED, MAIL_MARK_INFECTED, MAIL_MARK_JUNK, MAIL_MARK_PHISHING, MAIL_MARK_SPOOFED, MAIL_STATUS_ANSWERED, MAIL_STATUS_APPENDED, MAIL_STATUS_DELETED, MAIL_STATUS_DRAFT, MAIL_STATUS_EMPTY, MAIL_STATUS_ENCRYPTED, MAIL_STATUS_FLAGGED, MAIL_STATUS_RECENT, MAIL_STATUS_SEEN, and meta_message_t::status.

Referenced by portal_endpoint_messages_flag(), portal_endpoint_messages_list(), and portal_message_meta().

json_t* portal_message_header (meta_message_t * meta, mail_message_t * data)

Build the "header" section response to a rpc-json "messages.load" request.

Note:

The following header fields will be returned: To, CC, BCC, From, Replyto, Sender, Return-Path, Subject, Date, and Size.

Parameters:

meta a pointer to the meta message object of the requested message.

data a pointer to the mail message object containing the requested message's data.

Returns:

a pointer to a json object containing the header fields of the requested message.

Definition at line 84 of file messages.c.

References mail_message_t::header_length, json_pack_ex_d, log_pedantic, mail_header_fetch_cleaned(), mm_wipe(), PLACER, meta_message_t::size, st_char_get(), st_cleanup(), and mail_message_t::text.

Referenced by portal_endpoint_messages_load().

json_t* portal_message_info (meta_message_t * *meta*)

Build the "info" section response to a rpc-json "messages.load" request.

Parameters:

meta a pointer to the meta message object of the requested message.

Returns:

a pointer to a json object containing the appropriate information about the requested message.

Definition at line 275 of file messages.c.

References json_pack_ex_d, and log_pedantic.

Referenced by portal_endpoint_messages_load().

json_t* portal_message_meta (meta_message_t * *meta*)

Definition at line 18 of file messages.c.

References meta_message_t::foldernum, json_pack_ex_d, log_pedantic, meta_message_t::messagenum, portal_message_flags_array(), and portal_message_tags_array().

Referenced by portal_endpoint_messages_load().

json_t* portal_message_security (meta_message_t * *meta*)

TODO: Replace hard coded values with actual data.

Definition at line 47 of file messages.c.

References json_pack_ex_d, and log_pedantic.

Referenced by portal_endpoint_messages_load().

json_t* portal_message_server (meta_message_t * *meta*)

TODO: Replace hard coded values with actual data.

Definition at line 61 of file messages.c.

References meta_message_t::created, json_pack_ex_d, and log_pedantic.

Referenced by portal_endpoint_messages_load().

json_t* portal_message_source (meta_message_t * meta)

TODO: Replace hard coded values with actual data.

Definition at line 32 of file messages.c.

References json_pack_ex_d, log_pedantic, and rand_get_int64().

Referenced by portal_endpoint_messages_load().

json_t* portal_message_tags_array (meta_message_t * meta)

Definition at line 91 of file flags.c.

References ar_field_st(), ar_length_get(), count, json_array_append_new_d, json_array_d, json_string_d, st_char_get(), and meta_message_t::tags.

Referenced by portal_message_meta().

void portal_meta (connection_t * con)

Return information for a portal "meta" json-rpc request.

Note:

This function returns various pieces of information about the user such as their plan type, quota, etc.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2879 of file endpoint.c.

References con_addr_presentation(), connection_t::http, json_object_d, json_pack_ex_d, JSON_RPC_2_ERROR_SERVER_INTERNAL, log_pedantic, MAGMA_PORTAL_VERSION, MANAGEDBUF, portal_endpoint_error(), PORTAL_ENDPOINT_ERROR_META, portal_endpoint_response(), portal_validate_request(), and st_char_get().

bool_t portal_outbound_checks (credential_t * cred, uint64_t usernum, stringer_t * from, size_t num_recipients, stringer_t * body_plain, stringer_t * body_html, chr_t ** errmsg)

mail.c

mail.c

See also:

smtp_data_outbound()

Note:

The checks include: Pattern matching for junk mail, authorization for the user to use the email address or domain specified in the From address, and finally, a virus check.

Parameters:

cred a credential structure obtained for the authenticated user's session.

usernum the numerical id of the user attempting to send the message.

from a managed string containing the email address specified as the From address.
nrecipients the number of recipients that will receive the sent message.
body_plain a managed string containing the plain text body of the message.
body_html a managed string containing the html body of the message.
errmsg the address of a pointer to a null-terminated string that will be set to a descriptive error message on failure.

Returns:

true if all security checks were passed or false otherwise.

Definition at line 29 of file mail.c.

References `pattern_check()`, `smtp_check_authorized_from()`, `smtp_check_transmit_quota()`,
`smtp_fetch_authorization()`, and `virus_check()`.

Referenced by `portal_endpoint_messages_send()`.

int_t portal_parse_action (stringer_t * action)

Note:

This function is currently not called anywhere in the code and may be subject to removal.

Definition at line 126 of file parse.c.

References `PLACER`, `PORTAL_ENDPOINT_ACTION_ADD`, `PORTAL_ENDPOINT_ACTION_INVALID`,
`PORTAL_ENDPOINT_ACTION_LIST`, `PORTAL_ENDPOINT_ACTION_REMOVE`,
`PORTAL_ENDPOINT_ACTION_REPLACE`, and `st_cmp_ci_eq()`.

int_t portal_parse_context (stringer_t * context)

Parse the context of a requested folder.

Note:

If no context is specified, the "mail" context is assumed (`PORTAL_ENDPOINT_CONTEXT_MAIL`).

Parameters:

context a managed string containing the context (supported values are "mail", "contacts", "settings", and "help").

Returns:

`PORTAL_ENDPOINT_CONTEXT_INVALID` on failure, or the flag of the determined context on success.

Definition at line 95 of file parse.c.

References `PLACER`, `PORTAL_ENDPOINT_CONTEXT_CONTACTS`,
`PORTAL_ENDPOINT_CONTEXT_HELP`, `PORTAL_ENDPOINT_CONTEXT_INVALID`,
`PORTAL_ENDPOINT_CONTEXT_MAIL`, `PORTAL_ENDPOINT_CONTEXT_SETTINGS`, and
`st_cmp_ci_eq()`.

Referenced by `portal_endpoint_folders_add()`, `portal_endpoint_folders_remove()`,
`portal_endpoint_folders_rename()`, and `portal_endpoint_folders_tags()`.

int_t portal_parse_flags (json_t * array, uint32_t * flags)

Definition at line 51 of file flags.c.

References `count`, `json_array_get_d`, `json_array_size_d`, `json_string_value_d`, `MAIL_MARK_BLACKHOLED`,
`MAIL_MARK_INFECTED`, `MAIL_MARK_JUNK`, `MAIL_MARK_PHISHING`, `MAIL_MARK_SPOOFED`,
`MAIL_STATUS_ANSWERED`, `MAIL_STATUS_APPENDED`, `MAIL_STATUS_DELETED`,

MAIL_STATUS_DRAFT, MAIL_STATUS_FLAGGED, MAIL_STATUS_RECENT,
MAIL_STATUS_SEEN, NULLER, PLACER, and st_cmp_ci_eq().

Referenced by portal_endpoint_messages_flag().

inx_t* portal_parse_json_str_array (json_t * json, size_t * nout)

parse.c

parse.c

Parameters:

json a json object containing an array of strings.

nout if not NULL, an optional pointer to a size_t to receive the item count of the json string array.

Returns:

NULL on failure, or a pointer to an inx holder containing the specified json array contents as a collection of managed strings.

Definition at line 21 of file parse.c.

References count, inx_alloc(), inx_free(), inx_insert(), json_array_get_d, json_array_size_d, json_string_value_d, log_error, log_pedantic, M_INX_LINKED, M_TYPE_UINT64, ns_length_get(), st_free(), st_import(), multi_t::u64, and multi_t::val.

Referenced by portal_endpoint_messages_send().

int_t portal_parse_sections (json_t * array, uint32_t * sections)

Parse the json-rpc messages.load parameter "section" from an array of strings into a bitmask of section flags.

Parameters:

array a pointer to the json object representing the "section" string array of the json request message.

sections a pointer to an unsigned 32 bit integer that will receive the translated section flags on success.

Returns:

< 0 on error (-3 for an internal server error, -2 if an unknown flag was received, or -1 on syntax error), 0 for an empty sections array, or 1 on success.

Definition at line 162 of file parse.c.

References count, json_array_get_d, json_array_size_d, json_string_value_d, NULLER, PLACER, PORTAL_ENDPOINT_MESSAGE_ATTACHMENTS, PORTAL_ENDPOINT_MESSAGE_BODY, PORTAL_ENDPOINT_MESSAGE_HEADER, PORTAL_ENDPOINT_MESSAGE_INFO, PORTAL_ENDPOINT_MESSAGE_META, PORTAL_ENDPOINT_MESSAGE_SECURITY, PORTAL_ENDPOINT_MESSAGE_SERVER, PORTAL_ENDPOINT_MESSAGE_SOURCE, and st_cmp_ci_eq().

Referenced by portal_endpoint_messages_load().

void portal_print_login (connection_t * con, chr_t * message)

portal.c

portal.c

Note:

The portal login page can be found in 'portal/login.template'

Parameters:

con a pointer to the connection object of the client requesting the portal login page.
message a pointer to a null-terminated string that will be displayed to the user in the login page as the contents of the node with the id of "message" in the portal login template.

Returns:

This function returns no value.

Definition at line 23 of file portal.c.

References `con_write_st()`, `http_page_t::content`, `http_page_t::doc_obj`, `http_page_free()`, `http_page_get()`, `http_print_500()`, `http_response_header()`, `st_free()`, `st_length_get()`, `http_content_t::type`, `xml_dump_doc()`, `xml_node_set_content()`, `xml_xpath_eval()`, and `http_page_t::xpath_ctx`.

Referenced by `portal_process()`.

void portal_process (connection_t * con)

Process a connection to the web portal.

Note:

If `magma.web.portal.safeguard` is set, the user will be redirected to a secure login.

Parameters:

con a pointer to the connection of the http client requesting the portal.

Returns:

This function returns no value.

Definition at line 62 of file portal.c.

References `con_addr_word()`, `con_secure()`, `http_parse_context()`, `http_print_301()`, `magma`, `PLACER`, `magma_t::portal`, `portal_print_login()`, and `magma_t::web`.

Referenced by `http_response()`.

void portal_settings_identity (connection_t * con)

Return information for a portal "settings.identity" json-rpc request.

Note:

This function returns the full name, first name, last name, and website of the requested user.

Parameters:

con a pointer to the connection object across which the json-rpc response will be sent.

Returns:

This function returns no value.

Definition at line 2846 of file endpoint.c.

References `connection_t::http`, `json_object_d`, `json_pack_ex_d`, `JSON_RPC_2_ERROR_SERVER_INTERNAL`, `log_pedantic`, `portal_endpoint_error()`, `PORTAL_ENDPOINT_ERROR_SETTINGS_IDENTITY`, `portal_endpoint_response()`, `portal_validate_request()`, and `st_char_get()`.

stringer_t* portal_smtp_create_data (inx_t * *attachments*, stringer_t * *from*, inx_t * *to*, inx_t * *cc*, inx_t * *bcc*, stringer_t * *subject*, stringer_t * *body_plain*, stringer_t * *body_html*)

Create the data of an outbound smtp message that will be specified with the smtp DATA command.

Parameters:

attachments an optional inx holder containing a list of attachments to be included in the message.
from a managed string containing the email address sending the message.
to an inx holder containing a list of To: email recipients as managed strings.
cc an inx holder containing a list of 0 or more CC: recipients as managed strings.
bcc an inx holder containing a list of 0 or more BCC: recipients as managed strings.
subject a managed string containing the subject of the message.
body_plain an optional managed string containing the plain text body of the message.
body_html an optional managed string containing the html-formatted body of the message.

Returns:

NULL on failure or a managed string containing the packaged outbound smtp message data on success.

Definition at line 182 of file mail.c.

References `ar_append()`, `ar_free()`, `ar_length_get()`, `ARRAY_TYPE_POINTER`, `attachment_t::filedata`, `attachment_t::filename`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_reset()`, `inx_cursor_value_next()`, `log_pedantic`, `mail_mime_encode_part()`, `mail_mime_generate_boundary()`, `mail_mime_get_smtp_envelope()`, `st_free()`, and `st_merge`.

Referenced by `portal_endpoint_messages_send()`.

stringer_t* portal_smtp_merge_headers (inx_t * *headers*, stringer_t * *leading*, stringer_t * *trailing*)

Merge a list of smtp message headers into a single string, preceded by the leading text and followed by the trailing text.

Parameters:

headers an inx holder containing a collection of header string data to be merged together.
leading a managed string containing text that will lead each header line.
trailing a managed string containing text that will trail each header line.

Returns:

NULL on failure or a managed string containing the merged headers on success.

Definition at line 306 of file mail.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `st_alloc`, `st_cleanup()`, and `st_merge`.

Referenced by `mail_mime_get_smtp_envelope()`.

bool_t portal_smtp_relay_message (stringer_t * *from*, inx_t * *to*, stringer_t * *data*, size_t *send_size*, chr_t ** *errmsg*)

Send (relay) a message composed by a user via a portal session.

See also:

`smtp_relay_message()` - a lot of logic borrowed from here.

Parameters:

from a pointer to a managed string containing the email address specified as the From address.

to a pointer to a managed string containing the destination email address of the message.

data a pointer to a managed string containing the raw data of the mail message.

send_size if greater than 0, specify the optional SIZE parameter to the MAIL FROM command.

errmsg the address of a pointer to a null-terminated string that will be set to a descriptive error message on failure.

Returns:

true if the mail message was sent successfully, or false otherwise.

Definition at line 91 of file mail.c.

References `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `smtp_client_close()`, `smtp_client_connect()`, `smtp_client_send_data()`, `smtp_client_send_helo()`, `smtp_client_send_mailfrom()`, and `smtp_client_send_rcptto()`.

Referenced by `portal_endpoint_messages_send()`.

void portal_upload (connection_t * con)

Process uploaded attachments for messages composed in conjunction with the portal interface.

Definition at line 3126 of file endpoint.c.

References `con_addr_word()`, `con_secure()`, `data`, `attachment_t::filedata`, `get_header_opt()`, `connection_t::http`, `http_data_free()`, `http_data_header_parse_line()`, `HTTP_ERROR_401`, `HTTP_ERROR_403`, `HTTP_ERROR_404`, `HTTP_ERROR_405`, `HTTP_ERROR_500`, `HTTP_MERGED`, `HTTP_METHOD_POST`, `http_parse_context()`, `HTTP_PORTAL`, `http_print_301()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `log_error`, `log_pedantic`, `magma`, `multipart_get_boundary()`, `http_data_t::name`, `NULLER`, `pl_char_get()`, `pl_empty()`, `pl_length_get()`, `pl_null()`, `pl_shrink_before_characters()`, `pl_skip_characters()`, `pl_skip_to_characters()`, `PLACER`, `placer_t`, `magma_t::portal`, `portal_get_upload_attachment()`, `st_char_get()`, `st_cmp_ci_eq()`, `st_import()`, `st_length_get()`, `str_tok_get_bl()`, `str_tok_get_count_bl()`, `http_data_t::value`, and `magma_t::web`.

Referenced by `http_response()`.

magma/web/register/captcha.c File Reference

The captcha interface for the registration process.

```
#include "magma.h"
```

Functions

- **stringer_t * register_captcha_random_font** (void)
- *LOW: We shouldn't have to actually scan the fonts directory to find a valid file. Instead we could cache a list of valid fonts and then pick from it randomly.* void **register_captcha_write_noise** (gdImagePtr image, **int_t** x, **int_t** y)
- *Fill an image's background partially with pixelated noise to make it more difficult to read.* **stringer_t * register_captcha_generate** (**stringer_t** *value)

Generate a captcha image for a specified character string.

Detailed Description

The captcha interface for the registration process.

Definition in file **captcha.c**.

Function Documentation

stringer_t* register_captcha_generate (stringer_t * value)

Generate a captcha image for a specified character string.

captcha.c

Parameters:

value a managed string containing the text that is to become the basis of the captcha challenge.

Returns:

NULL on failure, or a managed string containing the path to the image file containing the captcha graphic on success.

Definition at line 111 of file captcha.c.

References `gdFree_d`, `gdImageColorResolve_d`, `gdImageCreate_d`, `gdImageDestroy_d`, `gdImageGifPtr_d`, `gdImageStringFT_d`, `log_pedantic`, `mm_wipe()`, `rand_get_uint32()`, `register_captcha_random_font()`, `register_captcha_write_noise()`, `st_char_get()`, `st_import()`, and `st_length_get()`.

Referenced by `register_print_captcha()`.

stringer_t* register_captcha_random_font (void)

LOW: We shouldn't have to actually scan the fonts directory to find a valid file. Instead we could cache a list of valid fonts and then pick from it randomly.

Select a random truetype font from the directory specified in **magma.http.fonts**.

Returns:

NULL on failure, or a managed string containing the pathname of the randomly selected font file on success.

Definition at line 20 of file captcha.c.

References count, magma_t::http, log_pedantic, magma, NULLER, PLACER, rand_get_uint32(), st_aprint(), and st_cmp_ci_ends().

Referenced by register_captcha_generate().

void register_captcha_write_noise (gdImagePtr *image*, int_t *x*, int_t *y*)

Fill an image's background partially with pixelated noise to make it more difficult to read.

Parameters:

image a pointer to the gd image to be modified.

x the height, in pixels, of the image region to be filled.

y the width, in pixels, of the image region to be filled.

Returns:

This function returns no value.

Definition at line 82 of file captcha.c.

References gdImageColorResolve_d, gdImageSetPixel_d, and rand_get_uint32().

Referenced by register_captcha_generate().

magma/web/register/register.c File Reference

Functions for handling the registration process.

#include "magma.h"

Functions

- void **register_print_message** (**connection_t** *con, **chr_t** *message)
- *Display a custom message to the remote client using the register/message template.* void **register_print_captcha** (**connection_t** *con, **register_session_t** *reg)
- *Print out a captcha challenge for a specified registration process.* void **register_print_step1** (**connection_t** *con, **register_session_t** *reg, **chr_t** *message)
- *Display step 1 of the registration process (collect username, password, and captcha challenge).* void **register_print_step2** (**connection_t** *con, **register_session_t** *reg, **chr_t** *message)
- *Display step 2 of the registration process.* void **register_print_step3** (**connection_t** *con)
- *Display the registration step 3 template for a connection that has successfully created a new user.* void **register_process** (**connection_t** *con)

Process a user registration request (/register).

Detailed Description

Functions for handling the registration process.

Definition in file **register.c**.

Function Documentation

void register_print_captcha (connection_t * con, register_session_t * reg)

Print out a captcha challenge for a specified registration process.

register.c

Parameters:

con the underlying client connection.

reg the pending registration session of the remote client.

Returns:

This function returns no value.

Definition at line 51 of file register.c.

References `con_print()`, `con_write_st()`, `http_print_500()`, `register_session_t::hvf_value`, `log_info`, `rand_choices()`, `register_captcha_generate()`, `st_char_get()`, `st_free()`, and `st_length_get()`.

Referenced by `register_process()`.

void register_print_message (connection_t * con, chr_t * message)

Display a custom message to the remote client using the register/message template.

Parameters:

con the client connection to receive the message.

message a pointer to a null-terminated string containing the custom message to be displayed inside the template sent to the remote client.

Returns:

This function returns no value.

Definition at line 22 of file register.c.

References `con_write_st()`, `content`, `http_get_template()`, `http_print_500()`, `http_response_header()`, `NULLER`, `PLACER`, `http_content_t::resource`, `st_dupe()`, `st_free()`, `st_length_get()`, `st_replace()`, and `http_content_t::type`.

Referenced by `register_abuse_checks()`.

void register_print_step1 (connection_t * con, register_session_t * reg, chr_t * message)

Display step 1 of the registration process (collect username, password, and captcha challenge).

Parameters:

con a pointer to the connection object of the remote client making the registration request.

reg a pointer to the pending registration session of the remote client.

message if not NULL, an optional error message to be displayed inside the registration step 1 template.

Returns:

This function returns no value.

Definition at line 83 of file register.c.

References `con_write_st()`, `content`, `http_get_template()`, `http_print_500_log()`, `http_response_header()`, `register_session_t::name`, `ns_length_get()`, `PLACER`, `http_content_t::resource`, `st_dupe()`, `st_free()`, `st_length_get()`, `st_replace()`, `http_content_t::type`, and `register_session_t::username`.

Referenced by `register_process()`.

void register_print_step2 (connection_t * con, register_session_t * reg, chr_t * message)

Display step 2 of the registration process.

Note:

Previously, this step used to obtain information like the desired user plan and billing info. At the moment, however, this step effectively does nothing but display a page to be clicked-through.

Parameters:

con a pointer to the connection object of the remote client making the registration request.

reg a pointer to the pending registration session of the remote client.

message if not NULL, an optional error message to be displayed inside the registration step 2 template.

Returns:

This function returns no value.

Definition at line 130 of file register.c.

References `con_write_st()`, `content`, `http_get_template()`, `http_print_500_log()`, `http_response_header()`, `register_session_t::name`, `ns_length_get()`, `PLACER`, `http_content_t::resource`, `st_dupe()`, `st_free()`, `st_length_get()`, `st_replace()`, and `http_content_t::type`.

Referenced by `register_process()`.

void register_print_step3 (connection_t * con)

Display the registration step 3 template for a connection that has successfully created a new user.

Parameters:

the remote connection making the registration request.

Returns:

This function returns no value.

Definition at line 169 of file register.c.

References `con_write_st()`, `content`, `http_get_template()`, `http_print_500()`, `http_response_header()`, `http_content_t::resource`, `st_dupe()`, `st_free()`, `st_length_get()`, and `http_content_t::type`.

Referenced by `register_process()`.

void register_process (connection_t * con)

Process a user registration request (/register).

Note:

All requests will be redirected to a secure location. All new registration requests will also need to pass a test to ensure that they haven't been the product of abusive behavior. If an existing registration session can't be located, a new one will be generated. Depending on the user supplied http POST data, one of the three registration steps will be shown, or a captcha challenge will be presented to the user.

Returns:

This function returns no value.

Definition at line 195 of file register.c.

References `con_secure()`, `connection_t::http`, `HTTP_COMPLETE`, `HTTP_DATA_GET`, `http_data_get()`, `HTTP_DATA_POST`, `HTTP_ERROR_500`, `http_print_301()`, `log_pedantic`, `register_abuse_checks()`, `register_business_step1()`, `register_business_step2()`, `register_print_captcha()`, `register_print_step1()`, `register_print_step2()`, `register_print_step3()`, `register_session_cache()`, `register_session_free()`, `register_session_generate()`, `register_session_get()`, `register_session_t::usernum`, and `http_data_t::value`.

Referenced by `http_response()`.

magma/web/register/register.h File Reference

Functions for handling the registration process.

Data Structures

- struct **register_session_t**

Defines

- #define **REGISTER_PASSWORD_MIN_LENGTH** 5
- #define **REGISTER_PASSWORD_MAX_LENGTH** 200
- #define **REGISTER_USERNAME_MIN_LENGTH** 1
- #define **REGISTER_USERNAME_MAX_LENGTH** 200

Functions

- **bool_t register_data_check_username** (stringer_t *username)
- *datatier.c* **inx_t * register_data_fetch_blocklist** (void)
- *Fetch the blocklist for new user registration from the database.* **bool_t register_data_insert_user** (**connection_t *con**, **register_session_t *reg**, **int_t transaction**, **uint64_t *outuser**)
- *Insert a newly registered user into the database using information gathered by registration step #2.* **bool_t register_abuse_check_blocklist** (**connection_t *con**)
- *abuse.c* **bool_t register_abuse_checks** (**connection_t *con**)
- *Check to see if a registration request is allowed by a remote host; if not, display a banner.* **void register_abuse_increment_history** (**connection_t *con**)
- *Increment the registration abuse counter for the requesting IP address.* **void register_blocklist_free** (void)
- *Free a registration blocked list.* **void register_blocklist_update** (void)
- *Update the registration blocklist from the database.* **stringer_t * register_captcha_generate** (**stringer_t *value**)
- *captcha.c* **stringer_t * register_captcha_random_font** (void)
- *LOW: We shouldn't have to actually scan the fonts directory to find a valid file. Instead we could cache a list of valid fonts and then pick from it randomly.* **void register_captcha_write_noise** (**gdImagePtr image**, **int_t x**, **int_t y**)
- *Fill an image's background partially with pixelated noise to make it more difficult to read.* **bool_t register_session_cache** (**connection_t *con**, **register_session_t *session**)
- *sessions.c* **void register_session_free** (**register_session_t *session**)
- *Destroy a registration session and all its associated data.* **register_session_t * register_session_generate** (void)
- *Generate a new registration session.* **register_session_t * register_session_get** (**connection_t *con**, **stringer_t *name**)
- *Retrieve a registration session by a data key supplied in the user's http POST request.* **chr_t * register_business_step1** (**connection_t *con**, **register_session_t *reg**)
- *business.c* **chr_t * register_business_step2** (**connection_t *con**, **register_session_t *reg**)
- *Perform verification checking on all step 2 completed user fields, and display a welcome banner on success.* **bool_t register_business_validate_password** (**stringer_t *password**)
- *Determine whether a registered password is valid.* **int_t register_business_validate_username** (**stringer_t *username**)
- *Determine whether a registered username is valid.* **void register_print_captcha** (**connection_t *con**, **register_session_t *reg**)
- *register.c* **void register_print_message** (**connection_t *con**, **chr_t *message**)
- *Display a custom message to the remote client using the register/message template.* **void register_print_step1** (**connection_t *con**, **register_session_t *reg**, **chr_t *message**)

- *Display step 1 of the registration process (collect username, password, and captcha challenge).* void **register_print_step2** (**connection_t** *con, **register_session_t** *reg, **chr_t** *message)
- *Display step 2 of the registration process.* void **register_print_step3** (**connection_t** *con)
- *Display the registration step 3 template for a connection that has successfully created a new user.* void **register_process** (**connection_t** *con)

Process a user registration request (/register).

Detailed Description

Functions for handling the registration process.

Definition in file **register.h**.

Define Documentation

#define REGISTER_PASSWORD_MAX_LENGTH 200

Definition at line 17 of file register.h.

Referenced by register_business_step1(), and register_business_validate_password().

#define REGISTER_PASSWORD_MIN_LENGTH 5

Definition at line 16 of file register.h.

Referenced by register_business_step1().

#define REGISTER_USERNAME_MAX_LENGTH 200

Definition at line 19 of file register.h.

Referenced by register_business_validate_username().

#define REGISTER_USERNAME_MIN_LENGTH 1

Definition at line 18 of file register.h.

Function Documentation

bool_t register_abuse_check_blocklist (connection_t * con)

abuse.c

abuse.c

Parameters:

con the client connection to be checked.

Returns:

false if the check was passed, or true if the connection was made from an IP on the blocklist.

Definition at line 63 of file abuse.c.

References `con_addr_standard()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `MANAGEDBUF`, `register_blocklist_lock`, `rwlock_lock_read()`, `rwlock_unlock()`, `st_char_get()`, `st_cmp_ci_starts()`, `st_length_get()`, and `stats_increment_by_name()`.

Referenced by `register_abuse_checks()`.

bool_t register_abuse_checks (connection_t * con)

Check to see if a registration request is allowed by a remote host; if not, display a banner.

Parameters:

con the remote connection to be checked.

Returns:

false if registration is allowed or true if not (remote host is on blocklist or registration has been throttled).

Definition at line 117 of file abuse.c.

References `cache_get_u64()`, `con_addr_presentation()`, `con_addr_standard()`, `MANAGEDBUF`, `NULLER`, `register_abuse_check_blocklist()`, `register_print_message()`, and `st_char_get()`.

Referenced by `register_process()`.

void register_abuse_increment_history (connection_t * con)

Increment the registration abuse counter for the requesting IP address.

Parameters:

con a pointer to the connection object of the client making the registration request.

Returns:

This function returns no value.

Definition at line 100 of file abuse.c.

References `cache_increment()`, `con_addr_standard()`, `MANAGEDBUF`, `NULLER`, and `st_char_get()`.

Referenced by `register_business_step2()`.

void register_blocklist_free (void)

Free a registration blocked list.

Returns:

This function returns no value.

Definition at line 23 of file abuse.c.

References `inx_free()`.

void register_blocklist_update (void)

Update the registration blocklist from the database.

Returns:

This function returns no value.

Definition at line 37 of file abuse.c.

References `inx_free()`, `register_blocklist_lock`, `register_data_fetch_blocklist()`, `rwlock_lock_write()`, and `rwlock_unlock()`.

chr_t* register_business_step1 (connection_t * con, register_session_t * reg)

business.c

business.c

Note:

Checks include captcha verification, username validation, and password reentry verification and validation.

Parameters:

con the underlying client connection.

reg the underlying registration session.

Returns:

NULL on success, or a descriptive error string on failure.

Definition at line 111 of file business.c.

References `data`, `http_data_get()`, `HTTP_DATA_POST`, `register_session_t::hvf_input`, `register_session_t::hvf_value`, `log_pedantic`, `register_session_t::password`, `register_business_validate_password()`, `register_business_validate_username()`, `register_data_check_username()`, `REGISTER_PASSWORD_MAX_LENGTH`, `REGISTER_PASSWORD_MIN_LENGTH`, `st_char_get()`, `st_cmp_ci_eq()`, `st_dupe()`, `st_length_int()`, `register_session_t::username`, and `http_data_t::value`.

Referenced by `register_process()`.

chr_t* register_business_step2 (connection_t * con, register_session_t * reg)

Perform verification checking on all step 2 completed user fields, and display a welcome banner on success.

Note:

Checks include plan type validation, and billing information processing. After step 2, the user's supplied information will be persisted into the database.

Parameters:

con the underlying client connection.

reg the underlying registration session.

Returns:

NULL on success, or a descriptive error string on failure.

Definition at line 179 of file business.c.

References `magma_t::admin`, `magma_t::contact`, `data`, `magma_t::domain`, `http_data_get()`, `HTTP_DATA_POST`, `log_pedantic`, `magma`, `PLACER`, `register_session_t::plan`, `register_abuse_increment_history()`, `register_data_insert_user()`, `smtp_send_message()`, `st_cleanup()`, `st_cmp_cs_eq()`, `st_dupe()`, `st_merge`, `magma_t::system`, `tran_commit()`, `tran_rollback()`, `tran_start()`, `register_session_t::username`, `register_session_t::usernum`, and `http_data_t::value`.

Referenced by register_process().

bool_t register_business_validate_password (stringer_t * password)

Determine whether a registered password is valid.

Note:

Each password must be between REGISTER_PASSWORD_MIN_LENGTH (5) and REGISTER_PASSWORD_MAX_LENGTH (200) characters long.

Parameters:

password the user's password to be evaluated.

Returns:

false on failure (too long, too short, or bad characters) or true on success.

Definition at line 21 of file business.c.

References length, REGISTER_PASSWORD_MAX_LENGTH, st_char_get(), and st_length_get().

Referenced by register_business_step1().

int_t register_business_validate_username (stringer_t * username)

Determine whether a registered username is valid.

Parameters:

username a managed string containing the proposed username to be evaluated.

Returns:

-1 or 0 on failure (too long, too short, or bad characters) or 1 on success.

Definition at line 53 of file business.c.

References length, REGISTER_USERNAME_MAX_LENGTH, st_char_get(), and st_length_get().

Referenced by register_business_step1().

stringer_t* register_captcha_generate (stringer_t * value)

captcha.c

captcha.c

Parameters:

value a managed string containing the text that is to become the basis of the captcha challenge.

Returns:

NULL on failure, or a managed string containing the path to the image file containing the captcha graphic on success.

Definition at line 111 of file captcha.c.

References gdFree_d, gdImageColorResolve_d, gdImageCreate_d, gdImageDestroy_d, gdImageGifPtr_d, gdImageStringFT_d, log_pedantic, mm_wipe(), rand_get_uint32(), register_captcha_random_font(), register_captcha_write_noise(), st_char_get(), st_import(), and st_length_get().

Referenced by register_print_captcha().

stringer_t* register_captcha_random_font (void)

LOW: We shouldn't have to actually scan the fonts directory to find a valid file. Instead we could cache a list of valid fonts and then pick from it randomly.

Select a random truetype font from the directory specified in **magma.http.fonts**.

Returns:

NULL on failure, or a managed string containing the pathname of the randomly selected font file on success.

Definition at line 20 of file captcha.c.

References count, magma_t::http, log_pedantic, magma, NULLER, PLACER, rand_get_uint32(), st_aprint(), and st_cmp_ci_ends().

Referenced by register_captcha_generate().

void register_captcha_write_noise (gdImagePtr *image*, int_t *x*, int_t *y*)

Fill an image's background partially with pixelated noise to make it more difficult to read.

Parameters:

image a pointer to the gd image to be modified.

x the height, in pixels, of the image region to be filled.

y the width, in pixels, of the image region to be filled.

Returns:

This function returns no value.

Definition at line 82 of file captcha.c.

References gdImageColorResolve_d, gdImageSetPixel_d, and rand_get_uint32().

Referenced by register_captcha_generate().

bool_t register_data_check_username (stringer_t * *username*)

datatier.c

datatier.c

Parameters:

username the username to be checked against the database.

Returns:

true if the username is taken or false if it is not.

Definition at line 68 of file datatier.c.

References mm_wipe(), res_row_next(), res_table_free(), st_char_get(), st_length_get(), and stmt_get_result().

Referenced by register_business_step1().

inx_t* register_data_fetch_blocklist (void)

Fetch the blocklist for new user registration from the database.

HIGH: The prepared statements being used aren't valid. The queries need to be copied over and created. register_fetch_blocklist still needs to be defined.

Returns:

an `inx` holder containing the registration blocklist as a collection of managed strings.

Definition at line 21 of file `datatier.c`.

References `inx_alloc()`, `inx_free()`, `inx_insert()`, `log_pedantic`, `M_INX_LINKED`, `M_TYPE_UINT64`, `res_field_string()`, `res_row_next()`, `res_table_free()`, `st_free()`, `stmt_get_result()`, `multi_t::u64`, and `multi_t::val`.

Referenced by `register_blocklist_update()`.

`bool_t register_data_insert_user (connection_t * con, register_session_t * reg, int_t transaction, uint64_t * outuser)`

Insert a newly registered user into the database using information gathered by registration step #2.

Note:

The following steps occur: 1. Insert a new user into the Users table, supplying username, hashed password, plan info, and quota. 2. Insert a blank entry into the Profile table for the user. 3. Insert an entry into the Folders table for the user's "Inbox" folder. 4. Insert a new entry into the Log table containing the username and IP address of the client request. 5. Insert a new entry into the Dispatch table for the user, configuring the spam folder, inbox, send/receive/daily send/daily receive limits, etc. 6. Insert a new entry into the Mailboxes table for the user.

Parameters:

con a pointer to the connection object of the client making the registration request.

reg the current registration session of the user to be added.

transaction a mysql transaction id for all database operations, since they all need to be committed atomically or rolled back.

outuser a pointer to a numerical id to receive the newly generated and inserted user id.

Returns:

true if the new user account was successfully created, or false on failure.

Definition at line 111 of file `datatier.c`.

References `credential_t::auth`, `con_addr_presentation()`, `CONTIGUOUS`, `credential_alloc_auth()`, `credential_free()`, `magma_t::domain`, `log_pedantic`, `magma`, `MANAGED_T`, `MANAGEDBUF`, `meta_data_user_build_storage_keys()`, `mm_wipe()`, `ns_length_get()`, `register_session_t::password`, `register_session_t::plan`, `SECURE`, `st_alloc_opts()`, `st_char_get()`, `st_cleanup()`, `st_data_get()`, `st_free()`, `st_length_get()`, `st_merge`, `stmt_exec_conn()`, `stmt_insert_conn()`, `magma_t::system`, and `register_session_t::username`.

Referenced by `register_business_step2()`.

`void register_print_captcha (connection_t * con, register_session_t * reg)`

`register.c`

`register.c`

Parameters:

con the underlying client connection.

reg the pending registration session of the remote client.

Returns:

This function returns no value.

Definition at line 51 of file `register.c`.

References `con_print()`, `con_write_st()`, `http_print_500()`, `register_session_t::hvf_value`, `log_info`, `rand_choices()`, `register_captcha_generate()`, `st_char_get()`, `st_free()`, and `st_length_get()`.

Referenced by `register_process()`.

void register_print_message (connection_t * con, chr_t * message)

Display a custom message to the remote client using the register/message template.

Parameters:

con the client connection to receive the message.

message a pointer to a null-terminated string containing the custom message to be displayed inside the template sent to the remote client.

Returns:

This function returns no value.

Definition at line 22 of file `register.c`.

References `con_write_st()`, `content`, `http_get_template()`, `http_print_500()`, `http_response_header()`, `NULLER`, `PLACER`, `http_content_t::resource`, `st_dupe()`, `st_free()`, `st_length_get()`, `st_replace()`, and `http_content_t::type`.

Referenced by `register_abuse_checks()`.

void register_print_step1 (connection_t * con, register_session_t * reg, chr_t * message)

Display step 1 of the registration process (collect username, password, and captcha challenge).

Parameters:

con a pointer to the connection object of the remote client making the registration request.

reg a pointer to the pending registration session of the remote client.

message if not NULL, an optional error message to be displayed inside the registration step 1 template.

Returns:

This function returns no value.

Definition at line 83 of file `register.c`.

References `con_write_st()`, `content`, `http_get_template()`, `http_print_500_log()`, `http_response_header()`, `register_session_t::name`, `ns_length_get()`, `PLACER`, `http_content_t::resource`, `st_dupe()`, `st_free()`, `st_length_get()`, `st_replace()`, `http_content_t::type`, and `register_session_t::username`.

Referenced by `register_process()`.

void register_print_step2 (connection_t * con, register_session_t * reg, chr_t * message)

Display step 2 of the registration process.

Note:

Previously, this step used to obtain information like the desired user plan and billing info. At the moment, however, this step effectively does nothing but display a page to be clicked-through.

Parameters:

con a pointer to the connection object of the remote client making the registration request.

reg a pointer to the pending registration session of the remote client.

message if not NULL, an optional error message to be displayed inside the registration step 2 template.

Returns:

This function returns no value.

Definition at line 130 of file register.c.

References `con_write_st()`, `content`, `http_get_template()`, `http_print_500_log()`, `http_response_header()`, `register_session_t::name`, `ns_length_get()`, `PLACER`, `http_content_t::resource`, `st_dupes()`, `st_free()`, `st_length_get()`, `st_replace()`, and `http_content_t::type`.

Referenced by `register_process()`.

void register_print_step3 (connection_t * con)

Display the registration step 3 template for a connection that has successfully created a new user.

Parameters:

the remote connection making the registration request.

Returns:

This function returns no value.

Definition at line 169 of file register.c.

References `con_write_st()`, `content`, `http_get_template()`, `http_print_500()`, `http_response_header()`, `http_content_t::resource`, `st_dupes()`, `st_free()`, `st_length_get()`, and `http_content_t::type`.

Referenced by `register_process()`.

void register_process (connection_t * con)

Process a user registration request (/register).

Note:

All requests will be redirected to a secure location. All new registration requests will also need to pass a test to ensure that they haven't been the product of abusive behavior. If an existing registration session can't be located, a new one will be generated. Depending on the user supplied http POST data, one of the three registration steps will be shown, or a captcha challenge will be presented to the user.

Returns:

This function returns no value.

Definition at line 195 of file register.c.

References `con_secure()`, `connection_t::http`, `HTTP_COMPLETE`, `HTTP_DATA_GET`, `http_data_get()`, `HTTP_DATA_POST`, `HTTP_ERROR_500`, `http_print_301()`, `log_pedantic`, `register_abuse_checks()`, `register_business_step1()`, `register_business_step2()`, `register_print_captcha()`, `register_print_step1()`, `register_print_step2()`, `register_print_step3()`, `register_session_cache()`, `register_session_free()`, `register_session_generate()`, `register_session_get()`, `register_session_t::usernum`, and `http_data_t::value`.

Referenced by `http_response()`.

bool_t register_session_cache (connection_t * con, register_session_t * session)

sessions.c

sessions.c

Note:

The session is stored under the parent key "lavad.register.session."

Parameters:

con a pointer to the client connection underlying the user session.
session a pointer to the registration session to be persisted.

Returns:

false on failure or true if the session was cached successfully.

Definition at line 111 of file sessions.c.

References `cache_set()`, `con_addr_presentation()`, `data`, `register_session_t::hvf_input`, `register_session_t::hvf_value`, `log_pedantic`, `MANAGEDBUF`, `register_session_t::name`, `NULLER`, `register_session_t::password`, `register_session_t::plan`, `serialize_st()`, `serialize_uint16()`, `serialize_uint64()`, `st_char_get()`, `st_cleanup()`, `st_free()`, `st_length_int()`, `register_session_t::username`, and `register_session_t::usernum`.

Referenced by `register_process()`.

void register_session_free (register_session_t * session)

Destroy a registration session and all its associated data.

Returns:

This function returns no value.

Definition at line 19 of file sessions.c.

References `register_session_t::hvf_input`, `register_session_t::hvf_value`, `mm_free()`, `register_session_t::name`, `register_session_t::password`, `st_cleanup()`, and `register_session_t::username`.

Referenced by `register_process()`, `register_session_generate()`, and `register_session_get()`.

register_session_t* register_session_generate (void)

Generate a new registration session.

Returns:

NULL on failure, or a pointer to a new randomly named session on success.

Definition at line 40 of file sessions.c.

References `log_pedantic`, `mm_alloc()`, `register_session_t::name`, `rand_choices()`, and `register_session_free()`.

Referenced by `register_process()`.

register_session_t* register_session_get (connection_t * con, stringer_t * name)

Retrieve a registration session by a data key supplied in the user's http POST request.

Note:

The session is stored under the parent key "lavad.register.session."

Parameters:

con the client connection underlying the user request.

name a managed string containing the registration session identifier.

Returns:

NULL on failure, or a pointer to the user's registration session on success.

Definition at line 64 of file sessions.c.

References `cache_get()`, `con_addr_presentation()`, `serialization_t::data`, `data`, `deserialize_st()`, `deserialize_uint16()`, `deserialize_uint64()`, `register_session_t::hvf_input`, `register_session_t::hvf_value`, `log_pedantic`, `MANAGEDBUF`, `mm_alloc()`, `mm_wipe()`, `register_session_t::name`, `NULLER`, `register_session_t::password`, `register_session_t::plan`, `register_session_free()`, `st_char_get()`, `st_dupe()`, `st_free()`, `st_length_int()`, `register_session_t::username`, and `register_session_t::usernum`.

Referenced by `register_process()`.

magma/web/statistics/statistics.h File Reference

Code for dynamically generating the portal statistics page.

Data Structures

- `struct statistics_vp_t`

Enumerations

- `enum { portal_stat_total_users = 0, portal_stat_users_checked_email_today = 1, portal_stat_users_checked_email_week = 2, portal_stat_users_sent_email_today = 3, portal_stat_users_sent_email_week = 4, portal_stat_emails_received_today = 5, portal_stat_emails_received_week = 6, portal_stat_emails_sent_today = 7, portal_stat_emails_sent_week = 8, portal_stat_users_registered_today = 9, portal_stat_users_registered_week = 10, portal_stat_users_num_statements = 11 }`

Functions

- `void statistics_process(connection_t *con)`
- *statistics.c* `void statistics_init(void)`
- *datatier.c* `bool_t statistics_refresh(void)`

Refresh all portal statistics from the database, if they haven't been updated recently.

Detailed Description

Code for dynamically generating the portal statistics page.

Definition in file `statistics.h`.

Enumeration Type Documentation

anonymous enum

Enumerator:

portal_stat_total_users
portal_stat_users_checked_email_today
portal_stat_users_checked_email_week
portal_stat_users_sent_email_today
portal_stat_users_sent_email_week
portal_stat_emails_received_today
portal_stat_emails_received_week
portal_stat_emails_sent_today
portal_stat_emails_sent_week
portal_stat_users_registered_today
portal_stat_users_registered_week
portal_stat_users_num_statements

Definition at line 16 of file `statistics.h`.

Function Documentation

void statistics_init (void)

datatier.c

datatier.c

Returns:

This function returns no value.

Definition at line 29 of file datatier.c.

References mm_wipe(), portal_stat_emails_received_today, portal_stat_emails_received_week, portal_stat_emails_sent_today, portal_stat_emails_sent_week, portal_stat_total_users, portal_stat_users_checked_email_today, portal_stat_users_checked_email_week, portal_stat_users_num_statements, portal_stat_users_registered_today, portal_stat_users_registered_week, portal_stat_users_sent_email_today, portal_stat_users_sent_email_week, and statistics_vp_t::stmt.

Referenced by statistics_refresh().

void statistics_process (connection_t * con)

statistics.c

statistics.c

Parameters:

con a pointer to the connection object across which the server statistics will be transmitted.

Returns:

This function returns no value.

Definition at line 23 of file statistics.c.

References con_write_st(), http_page_free(), http_page_get(), http_print_500(), http_response_header(), log_pedantic, portal_stat_emails_received_today, portal_stat_emails_received_week, portal_stat_emails_sent_today, portal_stat_emails_sent_week, portal_stat_total_users, portal_stat_users_checked_email_today, portal_stat_users_checked_email_week, portal_stat_users_registered_today, portal_stat_users_registered_week, portal_stat_users_sent_email_today, portal_stat_users_sent_email_week, st_free(), st_length_get(), statistics_refresh(), statistics_vp_t::val, xml_dump_doc(), xml_set_xpath_ns(), and xml_set_xpath_uint64().

Referenced by http_response().

bool_t statistics_refresh (void)

Refresh all portal statistics from the database, if they haven't been updated recently.

Returns:

true if all statistics were refreshed successfully, or false if they were not.

Definition at line 51 of file datatier.c.

References log_pedantic, mutex_lock(), mutex_unlock(), portal_statistics_mutex, PORTAL_STATISTICS_TIMEOUT, res_field_uint64(), res_row_next(), res_table_free(), statistics_init(), statistics_last_updated, statistics_vp_t::stmt, stmt_get_result(), and statistics_vp_t::val.

Referenced by statistics_process().

magma/web/teacher/teacher.c File Reference

Functions to allow users to train the statistical mail filter.

```
#include "magma.h"
```

Functions

- void **teacher_print_message** (**connection_t** *con, **chr_t** *message)
- *Display a custom message to the remote client using the teacher/message template.* void **teacher_print_form** (**connection_t** *con, **http_page_t** *page, **teacher_data_t** *teach)
- *Display a custom message to the remote client using the teacher/message template.* **http_page_t** * **teacher_add_error** (**chr_t** *xpath, **chr_t** *id, **chr_t** *message)
- *Get the teacher/teacher template and add an error message to it.* void **teacher_add_cookie** (**connection_t** *con, **teacher_data_t** *teach)
- *Create a cookie for a successfully password-authenticated teacher request.* void **teacher_process** (**connection_t** *con)

The main entry point for the /teacher web application.

Detailed Description

Functions to allow users to train the statistical mail filter.

Definition in file **teacher.c**.

Function Documentation

void teacher_add_cookie (connection_t * con, teacher_data_t * teach)

Create a cookie for a successfully password-authenticated teacher request.

teacher.c

Parameters:

con the connection of the web client accessing the teacher facility.

teach the spam signature associated with the cookie request; it supplies the identifying password stored in the cookie.

Returns:

This function returns no value.

Definition at line 151 of file teacher.c.

References `connection_t::http`, `inx_alloc()`, `inx_insert()`, `log_pedantic`, `M_INX_LINKED`, `M_TYPE_STRINGER`, `ns_length_get()`, `teacher_data_t::password`, `multi_t::st`, `st_char_get()`, `st_free()`, `st_import()`, and `multi_t::val`.

Referenced by `teacher_process()`.

http_page_t* teacher_add_error (chr_t * xpath, chr_t * id, chr_t * message)

Get the teacher/teacher template and add an error message to it.

Parameters:

xpath a null-terminated string containing the xpath of the node in the template to be marked with the error message.

id a null-terminated string containing the id of the error message node to be added as the sibling of the node in the specified xpath.

message a null-terminated string containing the error message to be displayed to the user.

NULL on failure, or a pointer to the modified teacher/teacher template page on success.

Definition at line 112 of file teacher.c.

References `http_page_get()`, `xml_node_add_sibling()`, `xml_node_free()`, `xml_node_new()`, `xml_node_set_content()`, `xml_node_set_property()`, `xml_xpath_eval()`, and `http_page_t::xpath_ctx`.

Referenced by `teacher_process()`.

void teacher_print_form (connection_t * con, http_page_t * page, teacher_data_t * teach)

Display a custom message to the remote client using the teacher/message template.

Parameters:

con the client connection to receive the message.

message a null-terminated string containing the custom message to be displayed inside the template sent to the remote client.

Returns:

This function returns no value.

Definition at line 68 of file teacher.c.

References `con_write_st()`, `http_page_t::content`, `teacher_data_t::disposition`, `http_page_t::doc_obj`, `http_page_free()`, `http_page_get()`, `http_print_500()`, `http_response_header()`, `teacher_data_t::keynum`, `teacher_data_t::signum`, `st_free()`, `st_length_get()`, `teacher_print_message()`, `http_content_t::type`, `xml_dump_doc()`, `xml_set_xpath_ns()`, `xml_set_xpath_property()`, and `http_page_t::xpath_ctx`.

Referenced by `teacher_process()`.

void teacher_print_message (connection_t * con, chr_t * message)

Display a custom message to the remote client using the teacher/message template.

Parameters:

con the client connection to receive the message.

message a null-terminated string containing the custom message to be displayed inside the template sent to the remote client.

Returns:

This function returns no value.

Definition at line 21 of file teacher.c.

References `con_print()`, `con_write_st()`, `http_page_t::content`, `http_page_t::doc_obj`, `connection_t::http`, `http_page_free()`, `http_page_get()`, `http_print_500()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `st_char_get()`, `st_free()`, `st_length_get()`, `http_content_t::type`, `xml_dump_doc()`, `xml_set_xpath_ns()`, and `http_page_t::xpath_ctx`.

Referenced by `teacher_print_form()`, and `teacher_process()`.

void teacher_process (connection_t * con)

The main entry point for the /teacher web application.

Note:

Each /teacher request must contain a spam signature and key specified by the user. If magma was able to find the signature, the signature's key must match the user-specified key value. The signature must also not have been previously trained. On successful training, the signature will be freed in memory and in the database, but kept for a couple of hours in distributed cache. User password authentication is necessary for training, and this routine manages cookies for subsequent training requests.

Parameters:

con the connection object underlying the client /teacher request.

Returns:

This function returns no value.

Definition at line 224 of file teacher.c.

References credential_t::auth, teacher_data_t::completed, con_secure(), credential_alloc_auth(), credential_free(), teacher_data_t::disposition, dspam_train(), connection_t::http, HTTP_COMPLETE, HTTP_DATA_ANY, http_data_get(), HTTP_DATA_HEADER, HTTP_DATA_POST, http_print_301(), teacher_data_t::keynum, teacher_data_t::password, PLACER, teacher_data_t::signature, st_char_get(), st_cleanup(), st_cmp_cs_eq(), st_cmp_cs_starts(), st_dupe(), st_length_get(), st_replace(), teacher_add_cookie(), teacher_add_error(), teacher_data_delete(), teacher_data_free(), teacher_data_get(), teacher_data_save(), teacher_print_form(), teacher_print_message(), uint64_conv_st(), teacher_data_t::username, teacher_data_t::usernum, and http_data_t::value.

Referenced by http_response().

magma/web/teacher/teacher.h File Reference

A facility for allowing users to train the statistical mail filter.

Data Structures

- struct **teacher_data_t**

Functions

- void **teacher_data_delete** (**teacher_data_t** **teach*)
- *datatier.c* **teacher_data_t** * **teacher_data_fetch** (uint64_t *signum*)
- *Fetch information about a spam signature from the database.* void **teacher_data_free** (**teacher_data_t** **teach*)
- *Free a spam signature object.* **teacher_data_t** * **teacher_data_get** (uint64_t *signum*)
- *Get information about a spam signature from the cache, or fall back to the database.* void **teacher_data_save** (**teacher_data_t** **teach*)
- *Save spam signature information to the cache.* void **teacher_add_cookie** (**connection_t** **con*, **teacher_data_t** **teach*)
- *teacher.c* **http_page_t** * **teacher_add_error** (**chr_t** **xpath*, **chr_t** **id*, **chr_t** **message*)
- *Get the teacher/teacher template and add an error message to it.* void **teacher_print_form** (**connection_t** **con*, **http_page_t** **page*, **teacher_data_t** **teach*)
- *Display a custom message to the remote client using the teacher/message template.* void **teacher_print_message** (**connection_t** **con*, **chr_t** **message*)
- *Display a custom message to the remote client using the teacher/message template.* void **teacher_process** (**connection_t** **con*)

The main entry point for the /teacher web application.

Detailed Description

A facility for allowing users to train the statistical mail filter.

Definition in file **teacher.h**.

Function Documentation

void teacher_add_cookie (**connection_t** * *con*, **teacher_data_t** * *teach*)

teacher.c

teacher.c

Parameters:

con the connection of the web client accessing the teacher facility.

teach the spam signature associated with the cookie request; it supplies the identifying password stored in the cookie.

Returns:

This function returns no value.

Definition at line 151 of file **teacher.c**.

References `connection_t::http`, `inx_alloc()`, `inx_insert()`, `log_pedantic`, `M_INX_LINKED`, `M_TYPE_STRINGER`, `ns_length_get()`, `teacher_data_t::password`, `multi_t::st`, `st_char_get()`, `st_free()`, `st_import()`, and `multi_t::val`.

Referenced by `teacher_process()`.

`http_page_t* teacher_add_error (chr_t * xpath, chr_t * id, chr_t * message)`

Get the teacher/teacher template and add an error message to it.

Parameters:

xpath a null-terminated string containing the xpath of the node in the template to be marked with the error message.

id a null-terminated string containing the id of the error message node to be added as the sibling of the node in the specified xpath.

message a null-terminated string containing the error message to be displayed to the user.

NULL on failure, or a pointer to the modified teacher/teacher template page on success.

Definition at line 112 of file `teacher.c`.

References `http_page_get()`, `xml_node_add_sibling()`, `xml_node_free()`, `xml_node_new()`, `xml_node_set_content()`, `xml_node_set_property()`, `xml_xpath_eval()`, and `http_page_t::xpath_ctx`.

Referenced by `teacher_process()`.

`void teacher_data_delete (teacher_data_t * teach)`

`datatier.c`

`datatier.c`

HIGH: After training a signature, we should search the messages table for references to the signature being trained and update the message status flags. The `UPDATE_SIGNATURE_FLAGS_ADD/UPDATE_SIGNATURE_FLAGS_REMOVE` queries were created for that purpose but aren't being used right now. Note to self: retroactively brand messages as junk accordingly.

Note:

If the signature matched junk, all matching messages in the database belonging to the user will have their junk flag cleared. But if the signature didn't match junk, all matching messages in the database belonging to the user will have the junk flag added.

Parameters:

teach the spam signature to be removed from the database.

Returns:

This function returns no value.

Definition at line 43 of file `datatier.c`.

References `teacher_data_t::completed`, `teacher_data_t::disposition`, `log_pedantic`, `MAIL_MARK_JUNK`, `mm_wipe()`, `teacher_data_t::password`, `teacher_data_t::signature`, `teacher_data_t::signum`, `st_cleanup()`, `stmt_exec()`, `teacher_data_t::username`, and `teacher_data_t::usernum`.

Referenced by `teacher_process()`.

`teacher_data_t* teacher_data_fetch (uint64_t signum)`

Fetch information about a spam signature from the database.

Parameters:

signum the numerical id of the spam signature to be retrieved.

NULL on failure, or a pointer to a newly allocated signature teacher object on success.

Definition at line 142 of file `datatier.c`.

References `teacher_data_t::disposition`, `teacher_data_t::keynum`, `mm_alloc()`, `mm_wipe()`, `teacher_data_t::password`, `res_field_int8()`, `res_field_string()`, `res_field_uint64()`, `res_row_next()`, `res_table_free()`, `teacher_data_t::signature`, `teacher_data_t::signum`, `stmt_get_result()`, `teacher_data_free()`, `teacher_data_t::username`, and `teacher_data_t::usernum`.

Referenced by `teacher_data_get()`.

void teacher_data_free (teacher_data_t * teach)

Free a spam signature object.

Parameters:

teach the spam signature object to be freed.

Returns:

This function returns no value.

Definition at line 20 of file `datatier.c`.

References `mm_free()`, `teacher_data_t::password`, `teacher_data_t::signature`, `st_cleanup()`, and `teacher_data_t::username`.

Referenced by `teacher_data_fetch()`, `teacher_data_get()`, and `teacher_process()`.

teacher_data_t* teacher_data_get (uint64_t signum)

Get information about a spam signature from the cache, or fall back to the database.

Parameters:

signum the numerical id of the spam signature to be retrieved.

NULL on failure, or a pointer to a newly allocated signature teacher object on success.

Definition at line 234 of file `datatier.c`.

References `cache_get()`, `teacher_data_t::completed`, `serialization_t::data`, `data`, `deserialize_int32()`, `deserialize_st()`, `deserialize_uint64()`, `teacher_data_t::disposition`, `teacher_data_t::keynum`, `log_pedantic`, `mm_alloc()`, `mm_wipe()`, `teacher_data_t::password`, `PLACER`, `teacher_data_t::signature`, `teacher_data_t::signum`, `st_free()`, `teacher_data_fetch()`, `teacher_data_free()`, `teacher_data_t::username`, and `teacher_data_t::usernum`.

Referenced by `teacher_process()`.

void teacher_data_save (teacher_data_t * teach)

Save spam signature information to the cache.

Note:

The information will be cached for 2 hours.

Parameters:

teach the spam signature to be cached.

Returns:

This function returns no value.

Definition at line 194 of file `datatier.c`.

References `cache_set()`, `teacher_data_t::completed`, `data`, `teacher_data_t::disposition`, `teacher_data_t::keynum`, `log_pedantic`, `teacher_data_t::password`, `PLACER`, `serialize_int32()`, `serialize_st()`, `serialize_uint64()`, `teacher_data_t::signature`, `teacher_data_t::signum`, `st_free()`, `teacher_data_t::username`, and `teacher_data_t::usernum`.

Referenced by `teacher_process()`.

`void teacher_print_form (connection_t * con, http_page_t * page, teacher_data_t * teach)`

Display a custom message to the remote client using the `teacher/message` template.

Parameters:

con the client connection to receive the message.

message a null-terminated string containing the custom message to be displayed inside the template sent to the remote client.

Returns:

This function returns no value.

Definition at line 68 of file `teacher.c`.

References `con_write_st()`, `http_page_t::content`, `teacher_data_t::disposition`, `http_page_t::doc_obj`, `http_page_free()`, `http_page_get()`, `http_print_500()`, `http_response_header()`, `teacher_data_t::keynum`, `teacher_data_t::signum`, `st_free()`, `st_length_get()`, `teacher_print_message()`, `http_content_t::type`, `xml_dump_doc()`, `xml_set_xpath_ns()`, `xml_set_xpath_property()`, and `http_page_t::xpath_ctx`.

Referenced by `teacher_process()`.

`void teacher_print_message (connection_t * con, chr_t * message)`

Display a custom message to the remote client using the `teacher/message` template.

Parameters:

con the client connection to receive the message.

message a null-terminated string containing the custom message to be displayed inside the template sent to the remote client.

Returns:

This function returns no value.

Definition at line 21 of file `teacher.c`.

References `con_print()`, `con_write_st()`, `http_page_t::content`, `http_page_t::doc_obj`, `connection_t::http`, `http_page_free()`, `http_page_get()`, `http_print_500()`, `inx_cursor_alloc()`, `inx_cursor_free()`, `inx_cursor_value_next()`, `st_char_get()`, `st_free()`, `st_length_get()`, `http_content_t::type`, `xml_dump_doc()`, `xml_set_xpath_ns()`, and `http_page_t::xpath_ctx`.

Referenced by `teacher_print_form()`, and `teacher_process()`.

void teacher_process (connection_t * con)

The main entry point for the /teacher web application.

Note:

Each /teacher request must contain a spam signature and key specified by the user. If magma was able to find the signature, the signature's key must match the user-specified key value. The signature must also not have been previously trained. On successful training, the signature will be freed in memory and in the database, but kept for a couple of hours in distributed cache. User password authentication is necessary for training, and this routine manages cookies for subsequent training requests.

Parameters:

con the connection object underlying the client /teacher request.

Returns:

This function returns no value.

Definition at line 224 of file teacher.c.

References credential_t::auth, teacher_data_t::completed, con_secure(), credential_alloc_auth(), credential_free(), teacher_data_t::disposition, dspam_train(), connection_t::http, HTTP_COMPLETE, HTTP_DATA_ANY, http_data_get(), HTTP_DATA_HEADER, HTTP_DATA_POST, http_print_301(), teacher_data_t::keynum, teacher_data_t::password, PLACER, teacher_data_t::signature, st_char_get(), st_cleanup(), st_cmp_cs_eq(), st_cmp_cs_starts(), st_dupe(), st_length_get(), st_replace(), teacher_add_cookie(), teacher_add_error(), teacher_data_delete(), teacher_data_free(), teacher_data_get(), teacher_data_save(), teacher_print_form(), teacher_print_message(), uint64_conv_st(), teacher_data_t::username, teacher_data_t::usernum, and http_data_t::value.

Referenced by http_response().

magma/web/web.h File Reference

The web application modules.

```
#include "contact/contact.h"
#include "portal/portal.h"
#include "register/register.h"
#include "statistics/statistics.h"
#include "teacher/teacher.h"
```

Detailed Description

The web application modules.

Definition in file **web.h**.

