UltraPse : A universal and extensible software platform for representing biological sequences

A User Manual

Pu-Feng Du PufengDu@gmail.com

School of Computer Science and Technology, Tianjin University, Tianjin 300350, China

Date	Version
Sep.19th, 2017	1.0

Contents

1 Introduction

A number of biological questions can be formulated as a pattern classification problem. For example, predicting protein subcellular localization, predicting protein post-translational modification sites and predicting recombination hotspots are all pattern classification problems. Since all these problems requires biological sequences as input, they are usually called biological sequence classification problems. The paradigms for classifying biological sequences usually has a very important step: representing every biological sequence with a fixed-length numerical vector. This is because most of the machine learning algorithm would require this kind of data as their input, and most of these problems can be solved by using machine learning algorithms.

There are a number of existing software for this task. For example, the PseAAC-Builder, PseAAC-General, PseKNC, PseKNC-General and Pse-In-One. These software can convert protein sequences or nucleotide sequences to numerical forms. As the direct decedent of PseAAC-General, the most important feature of UltraPse is its extensibility. The types of representations can be extended by the users. The users can use

Lua scripts and binary extension modules to configure and extend UltraPse by means of new type of biological sequence, new type of physicochemical properties and new algorithms of representations.

UltraPse is not a new competitor in this playground, which has already been filled with many other software. UltraPse aims to become a universal platform for representing biological sequences, where the users can develop their own sequence representation algorithms.

2 Installation

2.1 Linux/UNIX

To install the Linux/UNIX version of UltraPse, you need to ensure that you have all the following dependencies installed before you can compile the source codes.

- lua-devel
- GNU g++ (supporting --std=c++14 or --std=c++1y)

Follow the instructions on GitHub pages to prepare your Lua environment first!!

After you have prepared all these dependencies, you should just type the following command:

```
make
```

If there is no compiling error, you should have your UltraPse executables compiled.

2.2 Windows

Installation on Windows is pretty simple. UltraPse has no dependency on Windows. No compilation process is needed. You only need to download the `.zip` package and unpack it in your favored location. You can then use Windows command line environment to execute the Windows version UltraPse. Although the GUI of UltraPse is being developed, there is currently **NO GUI** for Windows version.

2.3 Others

There is **NO guarantee** that the source code of UltraPse can be compiled on other systems. However, most part of the UltraPse are written by standard C++ following ISO C++ 14 standard syntax with standard library. As long as you can solve the dependency problems, it should not be a problem to let UltraPse run on other systems.

3 Usage

3.1 Command line

Like its predecessor, the PseAAC-General, UltraPse has a command line interface, which follows GNU standard. There are many command line options for UltraPse.

The basic syntax for UltraPse is as the following:

```
Usage: upse [OPTION...]
UltraPse -- An ultra-fast extensible software platform for biological sequence
representations.

  -a, --arguments=LUA-EXPR   Lua expression to provide extra module-specific
                             parameters. This option can appear for multiple
                             times.
  -f, --format={svm|tsv|csv} Output format.
  -i, --in=FILE              A FASTA format file for input. The comment line of
                             a sequence in this file should be unique. If you
                             do not specify an in-file, the program will try to
                             read it from keyboard.
  -k, --k-tuple=K            Use K-tuples in computation. This option can have
                             user defined purpose.
  -l, --lambda=L             An integer for maximal lag. This option can have
                             user defined purpose.
  -m, --mode={comp|pse|dpc|cov|lua|pseb3|user,LIB_OBJECT,MOD_NAME}
                             Representation module choices. This option can
                             appear for multiple times.
  -n,                              --note={stdprot|didna|dirna|tridna}
Sequence definition types. There are four internal
                             types, if the user want to use their own sequence
                             type, they have to use a task definition file.
  -o, --out=FILE             A file for storing the results. If you do not
                             specify an out-file, the results will be written
                             on screen.
  -p, --property=PROP-ID     Activate the property PROP-ID for current task.
                             This option can appear for multiple times.
  -q, --query={prop|mode|note|all}
                             Query status of everything.
  -t, --type=T               Type parameter in pseknc, pseaac. This option can
                             have user defined purpose.
  -u, --user-task=TASK-FILE  Task description file.
  -v, --validate             Validate sequences automatically. Sequence with
                             invalid notations will be dropped automatically.
  -w, --omega=W              A decimal number for pseknc and pseaac mode,
                             parameter w. This option can have user defined
                             purpose.
  -?, --help                 give this help list
      --usage                give a short usage message
  -V, --version              print program version

Mandatory or optional arguments to long options are also mandatory or optional
for any corresponding short options.

Report bugs to Dr. Pu-Feng Du: <PufengDu@gmail.com>.
```

The interpretation of all options are here:

- `-i, --in=FILE`

This option is to designate `FILE` as the input of UltraPse. `FILE` should be a FASTA format file. Unlike the PseAAC-General, UltraPse has no additional requirement to the FASTA format. Moreover, UltraPse can recognize the FASTA file format, and can automatically parse the comment line, as long as your FASTA format file follows the specification of one of the following five major databases: ***UniProt***, ***GenBank***, ***EMBL***, ***DDBJ*** and ***RefSeq***. For example, if your FASTA file contain the following contents:

```
>sp|P04637|P53_HUMAN Cellular tumor antigen p53 OS=Homo sapiens GN=TP53 PE=1 SV=4
MEEPQSDPSVEPPLSQETFSDLWKLLPENNVLSPLPSQAMDDLMLSPDDIEQWFTEDPGP
DEAPRMPEAAPPVAPAPAAPTPAAPAPAPSWPLSSSVPSQKTYQGSYGFRLGFLHSGTAK
SVTCTYSPALNKMFCQLAKTCPVQLWVDSTPPPGTRVRAMAIYKQSQHMTEVVRRCPHHE
RCSDSDGLAPPQHLIRVEGNLRVEYLDDRNTFRHSVVVPYEPPEVGSDCTTIHYNYMCNS
SCMGGMNRRPILTIITLEDSSGNLLGRNSFEVRVCACPGRDRRTEEENLRKKGEPHHELP
PGSTKRALPNNTSSSPQPKKKPLDGEYFTLQIRGRERFEMFRELNEALELKDAQAGKEPG
GSRAHSSHLKSKKGQSTSRHKKLMFKTEGPDSD
>sp|Q96EB6|SIR1_HUMAN NAD-dependent protein deacetylase sirtuin-1 OS=Homo sapiens GN=SIRT1
PE=1 SV=2
MADEAALALQPGGSPSAAGADREAASSPAGEPLRKRPRRDGPGLERSPGEPGGAAPEREV
PAAARGCPGAAAAALWREAEAEAAAAGGEQEAQATAAAGEGDNGPGLQGPSREPPLADNL
YDEDDDDEGEEEEEAAAAAIGYRDNLLFGDEIITNGFHSCESDEEDRASHASSSDWTPRP
RIGPYTFVQQHLMIGTDPRTILKDLLPETIPPPELDDMTLWQIVINILSEPPKRKKRKDI
NTIEDAVKLLQECKKIIVLTGAGVSVSCGIPDFRSRDGIYARLAVDFPDLPDPQAMFDIE
YFRKDPRPFFKFAKEIYPGQFQPSLCHKFIALSDKEGKLLRNYTQNIDTLEQVAGIQRII
QCHGSFATASCLICKYKVDCEAVRGDIFNQVVPRCPRCPADEPLAIMKPEIVFFGENLPE
QFHRAMKYDKDEVDLLIVIGSSLKVRPVALIPSSIPHEVPQILINREPLPHLHFDVELLG
DCDVIINELCHRLGGEYAKLCCNPVKLSEITEKPPRTQKELAYLSELPPTPLHVSEDSSS
PERTSPPDSSVIVTLLDQAAKSNDDLDVSESKGCMEEKPQEVQTSRNVESIAEQMENPDL
KNVGSSTGEKNERTSVAGTVRKCWPNRVAKEQISRRLDGNQYLFLPPNRYIFHGAEVYSD
SEDDVLSSSSCGSNSDSGTCQSPSLEEPMEDESEIEEFYNGLEDEPDVPERAGGAGFGTD
GDDQEAINEAISVKQEVTDMNYPSNKS
>sp|O60260|PRKN_HUMAN E3 ubiquitin-protein ligase parkin OS=Homo sapiens GN=PRKN PE=1 SV=2
MIVFVRFNSSHGFPVEVDSDTSIFQLKEVVAKRQGVPADQLRVIFAGKELRNDWTVQNCD
LDQQSIVHIVQRPWRKGQEMNATGGDDPRNAAGGCEREPQSLTRVDLSSSVLPGDSVGLA
VILHTDSRKDSPPAGSPAGRSIYNSFYVYCKGPCQRVQPGKLRVQCSTCRQATLTLTQGP
SCWDDVLIPNRMSGECQSPHCPGTSAEFFFKCGAHPTSDKETSVALHLIATNSRNITCIT
CTDVRSPVLVFQCNSRHVICLDCFHLYCVTRLNDRQFVHDPQLGYSLPCVAGCPNSLIKE
LHHFRILGEEQYNRYQQYGAEECVLQMGGVLCPRPGCGAGLLPEPDQRKVTCEGGNGLGC
GFAFCRECKEAYHEGECSAVFEASGTTTQAYRVDERAAEQARWEAASKETIKKTTKPCPR
CHVPVEKNGGCMHMKCPQPQCRLEWCWNCGCEWNRVCMGDHWFDV
```

UltraPse can automatically recognize these three sequence and identify them as **`sp|P04637`**, **`sp|Q96EB6`** and **`sp|O60260`**. You do NOT need to write the sequence on a single line. The downloaded format can be directly recognized.

If there is no `-in` option on the command line, UltraPse will try to find it in the Task Definition File. If the input file can not be find again, UltraPse will use **Standard Input** as its data source. This feature is particularly useful in writing shell scripts.

If there are multiple `-in` options on the command line, only the last one (from left to right) is effective.

- `-o,--out=FILE`

This option specify the output file location. The `FILE` must be a valid file name on the platform. For example, on Linux, it could be `~/data/test.pseaac`, while on Windows platform, it could be `D:\mydata\test.pseaac`.

Please note that, for data security reasons, UltraPse will automatically **ABORT ALL OPERATIONS**, if the specified output file has already existed.

If there are multiple `-o` options on the command line, only the last one will be effective.

This option equals to `SetOutputFile` in a TDF.

If there is no `-o` option on the command line, nor the TDF, UltraPse will use **Standard Output** as the output file. This is particularly useful in scripting.

UltraPse support three different output format, which can be configured using the `-f` option.

- `-f, --format={svm|tsv|csv}`

This option specifies the output file format. UltraPse, like most of the existing programs, support three different output formats, libSVM format (`-f svm`), TSV format (`-f tsv`) and CSV format (`-f csv`).

The libSVM format follows the libSVM data file specification, which can be found here: https://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html#_TOP .

In the libSVM format, **the class label is always zero** in UltraPse. This is different to PseAAC-General.

The TSV format is called Table Separated Vectors. The data are separated by `\\t` character, whose ASCII code equals `0x09`.

The CSV format is called Comma Separated Vector. The data are separated by ',' character.

Both CSV and TSV format can be loaded in spreadsheets program, like Microsoft Excel or LibreOffice Calc.

- `-u, --user-task=TASK-FILE`

This option is used to load a Task Definition File (TDF), which is specified by `TASK-FILE`. A TDF is a Lua script. It can access UltraPse internal data structure. It can also be used to define new sequence representation modes, new sequence types and new physicochemical properties.

The format of a TDF follows syntax of Lua language, which can be found here: http://www.lua.org/manual/5.3/

There are many UltraPse specific functions when writing a TDF. The details of these functions, as well as how to define new representation modes can be found in the Task Definition File section of this document.

- `-m, --mode={comp|pse|dpc|cov|lua|pseb3|user,LIB_OBJECT,MOD_NAME}`

This option add computation modules into current computation task. This option can appear multiple times on the command line. The computation module will work according to the order they are specified on the command line. The UltraPse host program supports five different computational modules. The computation modules can be defined using Binary Shared Objects (BSOs). It also provide compatibility to Lua extensions of PseAAC-General.

This option equals to the `AddModule` function in a TDF.

The `comp` module compute the notation compositions of a sequence, accodring to the value of `-n` option. That is to say, if `-n` option specify a sequence type like "Dinucleotide", the `comp` module will compute the 16-D dinucleotide compositions. If `-n` option specify a sequence type called "DNA", the same `comp` module will compute the 4-D nucleotide compositions.

The `pse` module compute the PseAAC or PseKNC general form descriptors. Its behaviour can be defined using a TDF. When `-n` option specify that the sequence is a protein sequence, the module `pse` will compute PseAAC. When `-n` option specify the sequecne is a dinucleotide sequence, the module `pse` will compute the PseDNC. When a user-defined sequence type is defined, as long as compatible physicochemical properties are defined also, the module `pse` will compute PsexxC according to the user definition of sequence types and physichochemical properties.

For a hypothetical example, a user can define a sequence type describing methylated cytidines, using five letters. Lets say that they are using ACGTM, where M indicate methylated cytidines. Lets also say that they intend to see the sequence as di-nucleotide sequences. If the physicochemical properties of all dinucleotide including M are properly defined, the module `pse` will produce pseudo-methylated-cytidine K-nucleotide compositions (PseMeCKNC). With parameter $\boldsymbol{\lambda = 3}$, a $\boldsymbol{25 + 3 = 28 - D}$ feature vector will be generated for Type-I PseMeCKNC descriptors. If there are 2 physicochemical properties defined, a $\boldsymbol{25 + 3 * 2 = 31 - D}$ feature vector will be generated for Type-II PseMeCKNC descriptors.

To select Type-I or Type-II descriptors, the `-t` option should be applied. `-t 1` is for Type-I descriptors, and `-t 2` is for Type-II descriptors.

To choose physicochemical properties, either use `-p` option on the command line, or use `AddProperty` function in a TDF. If both methods are used, the result will consider properties indicated by both methods together.

The `dpc` module is the only module that are restriced to protein sequence, which can only be used along with `-n stdprot`. It provides di-peptide compositions, which is compatible with PseAAC-General.

The `cov` module is to calculate covariance based descriptors, including the auto-correlation descriptor (`-t 1`), cross-correlation descriptor (`-t 2`) and auto-cross-correlation descriptors (`-t 3`). All these descriptors have been defined in Pse-In-One. To specify maximal lag parameter, the `-l` option should be applied. To choose different descriptors, the `-t` option should be applied.

The `lua` module is to interface the user-defined sequence representation mode in TDF form. The details of how UltraPse ultilize user-defined sequence representation mode in TDF can be found in Task Definition File section.

The `pseb3` module is to interface with PseAAC-General legacy modules. All PseAAC-General Lua extensions can be applied in UltraPse **with minor modifications**.

The `user` module is to interface with the BSO form user defined mode. The user module take two sub-options: LIB_OBJECT is a filename and path that specify a loadable library. On Windows platform, it is usually a DLL file. On Linux platform, it is usually a SO file. The MOD_NAME is the name of the computational module that should be loaded from the BSO. A BSO file may contain multiple computational modules.

- `-n, --note={stdprot|didna|dirna|tridna}`

Specify the seqeunce types, aka notation definitions. There are four public internal notation types. They are Protein (`-n stdprot`), Di-nucleotide (DNA) (`-n didna`), Di-nucleotide (RNA) (`-n dirna`), and Tri-nucleotide (`-n tridna`).

There are other internal notation types. They can be explored by using the `-q note` method. They can be applied if the users knows exactly what they indicates. However, the non-public internal notation types are not guaranteed to be supported in future versions of UltraPse. We **DO NOT** support the other internal notation types. No detail documentation will be provided for those non-public internal notation types.

This option equals to the `SetNotation` function in a TDF. If both command line option and TDF function are used, the TDF function override the command line option.

To define new sequence types, a TDF **MUST** be used.

- `-p, --property=PROP-ID`

This option add a physicochemical property into current computational task. In UltraPse, the set of available physicochemical properties is determined by the sequence types. For example, if you speified `-n stdprot`, only protein properties can be applied in `-p` option. If you specified `-n didna`, only Di-nucleotide (DNA) properties can be applied. To find out integrated property types, use `-q prop` option with a corresponding setting on `-n` option.

This option equals `AddProperty` function in a TDF.

To define a new physichchemical property, a TDF **MUST** be used.

For those properties that are defined in a TDF, they can also be specified on command line, as long as you load the right TDF.

If both TDF and command line specify physicochemical properties, the result will use both.

- `-v, --validate`

This option let UltraPse to automatically filter out those sequences with invalide notations, according to the requirement of `-n` option.

If this option is not specified, UltraPse will automatically abort operation with **segmentation fault** message when a sequence with invalide notation is encountered. If this options is specified, UltraPse will skip the seqeunce and report that when all the processing are completed.

If you are not sure whether your input data completely conform to your sequence type specification, **ALWAYS** use this option.

If you are very sure that your data comform to the sequence type specification, losing this option will accelarate the processing a little.

- `-l, --lambda=L`
- `-w, --omega=W`
- `-t, --type=T`
- `-k, --k-tuple=K`
- `-a, --arguments=LUA-EXPR`

The above five options are called "Module specific paramter options". The are used to provide parameters for computational modules on the command line. The meanings of `-l, -w, -t, and -k` are different in different computational modules. They can be read in a TDF, using function `GetOption`.

The `-a` option provide additional ways to give parameters to a TDF. The `LUA-EXPR` is a piece of single-line Lua script. This piece of script will be executed in the same scope and space as the TDF, but before TDF. Therefore, TDF can read the values that are specified in this `LUA-EXPR`.

3.2 Task Definition File

3.2.1 TDF general instructions

A TDF in UltraPse is a Lua script. UltraPse includes a fully functional Lua interpreter inside. The TDF is executed as a Lua script after UltraPse parsed all command line options, and before UltraPse create any computational facilities. Therefore, a TDF can alter all sort of definitions, options and parameters. It can configure computational engine, physicochemical properties, sequence notation definitions. But, it can not access instanced computational facilities, because when the the TDF is executed, those computational facilities have not been created yet. A TDF can provide functions, so that some computational modules can call these functions to implement user-defined representation modes.

Since TDF is a Lua script, all Lua language elements are valid. You can use Lua controling statements in a TDF.

In the following part, we will introduce the **UltraPse TDF specific *functions***, which are defined in UltraPse and can be called in a TDF. We will also introduce the **UltraPse TDF specific *interfaces***, which are defined in a TDF and will be called by the UltraPse.

3.2.2 TDF specific functions

- **LoadModule**

Syntax : `LoadModule (BSO_File_Name)`

Parameter Type: BSO_File_Name : `LUA_TSTRING`. A double quoted string or equivalent in Lua.

Returned Value: None

Function: Load a BSO file. Make the module in the BSO ready to be chosen.

- **AddModule**

Syntax : `AddModule (Module_Name)`

Parameter Type: Module_Name : `LUA_TSTRING`. A double quoted string or equivalent in Lua.

Returned Value: None

Function: Add a module into the computational engine. The module name can be internal modules or those in BSO files. If a module is defined in a BSO file, this BSO file must be loaded before, using `LoadModule` function.

- **GetAllModules**

Syntax : `GetAllModules ()`

Parameter Type: None.

Returned Value: `LUA_TTABLE`. A Lua table object. Indices are 1-based subscripts. Values are the names of all available modules. These names are provided as `LUA_TSTRING` type values.

Function: Returns all available module names.

- **GetActiveModules**

Syntax : `GetActiveModules ()`

Parameter Type: None.

Returned Value: `LUA_TTABLE`. A Lua table object, Indices are 1-based subscripts. Values are the names of all modules that have already been added into the computational engine. These names are provided as `LUA_TSTRING` type values.

Function: Returns all activated module names.

- **DefineNotation**

Syntax : `DefineNotation (Notation_Object)`

Parameter Type: Notation_Object : `LUA_TTABLE`. A Lua table object that defines a new sequence type.

The Notation_Object **MUST** follow the following format:

```
Notation_Object = {
    Name = "...";
    Base = "...";
    Length = ...;
    ReduceMap = "...";
};
```

Notation_Object can be any valid variable name in Lua. *Name*, *Base*, *Length* and *ReduceMap* are four mandatory member in this table. *Name* is a double quoted string. It can be any valid string in Lua. It must be unique among all sequence types, including internal types and user-defined types. *Base* is a double quoted string. It contains all the letters that will appear in a sequence. *Length* is an integer. It defines how many letter in the sequences should be regarded as a unit. For example, for di-nucleotide sequence type, you write Base = "ACGT"; Length = 2;, while for tri-nucleotide sequence type, you should write Base="ACGT"; Length = 3;. *ReduceMap* is a double quoted string. It is the reducing rule when sequences are analyzed. For example, if you specify ReduceMap = "AAACAGAT" in a di-nucleotide sequence type, the AA, AC, AG and AT will be treated equally as AA.

Returned Value: None

Function: Add a user-defined seuqence type, make it ready to be chosen.

- **GetNotation**

Syntax `GetNotation()`

Parameter types : None.

Returned value: `LUA_TSTRING`. A double-quoted string in Lua or equavalent.

Function: Return the current setting of the sequence type, as the name of that type.

- **SetNotation**

Syntax: `SetNotation(Notation_Name)`

Parameter types : Notation_Name: `LUA_TSTRING`. A double-quoted string in Lua or equavalent.

Returned value: None.

Function: Alter the current setting of the sequence type, using the name of a sequence type. The specified Notation_Name must be a name of internal sequence type or a type that has already been defined using `DefineNotation`.

- **GetInputFile**

Syntax: `GetInputFile()`

Parameter types: None.

Returned value: `LUA_TSTRING`. A double-quoted string in Lua.

Function: Get the current setting of the input file. If current input file is standard input, it will return "stdin".

- **SetInputFile**

Syntax: `SetInputFile(INPUT_FILE_NAME)`

Parameter types: INPUT_FILE_NAME: `LUA_TSTRING`. A double-quoted string in Lua or equavalent.

Returned value: None.

Function: Set the current setting of the input file.

- **GetOutputFile**

Syntax: `GetOutputFile()`

Parameter types: None.

Returned value: `LUA_TSTRING`. A double-quoted string in Lua.

Function: Get the current setting of the output file. If current output file is standard output, it will return "stdout".

- **SetOutputFile**

Syntax: `SetOutputFile(OUTPUT_FILE_NAME)`

Parameter types: OUTPUT_FILE_NAME: `LUA_TSTRING`. A double-quoted string in Lua.

Returned value: None.

Function: Set the current setting of the output file.

- **GetOption**

Syntax: `GetOption(OPTION_NAME)`

Parameter types: OPTION_NAME: `LUA_TSTRING`. A double-quoted string in Lua.

Returned value: `LUA_TSTRING`. The corresponding value of the command line options.

Function: Return a command line options value. There are four command line options that can be accessed using this function: `-l, -w, -t and -k`. To access them, the corresponding OPTION_NAME must be used according to the following table:

Option name	Command line option	Returned Type
`_cmd_lambda`	`-l`	`LUA_TSTRING`
`_cmd_omega`	`-w`	`LUA_TSTRING`
`_cmd_subtype`	`-t`	`LUA_TSTRING`
`_cmd_ktuple`	`-k`	`LUA_TSTRING`

For example, if `GetOption("_cmd_lambda")` is used, it will return whatever is set on the command line for the `-w` option.

- **DefineProperty**

Syntax: `DefineProperty(Property_Object)`

Parameter types: Property_Object: `LUA_TTABLE`. A Lua table object describing a type of physicochemical property. This table object **MUST** follow the following format:

```
Property_Object =
{
    Template = "...";
    ID = "...";
    Values =
    {
        AA = 1.0;
        AC = 2.0;
        AG = 3.0;
        AT = 3.3;
        ...
    };
    Comment = "...";
}
```

Property_Object can be any valid variable name in Lua. This table has four mandatory members: *Template*, *ID*, *Values* and *Comment*. *Template* is identifier of the existing physicochemical properties that should be used as the template for this new physicochemical properties. Besides the values altered in the *Values* section, all the other values of the new physicochemical properties will be the same as the template. If *Template* is "", *Values* section must specify values for all notations in current sequence types. *ID* is a string. It is the unique identifier of the property. *Values* is a table object. It defines all physicochemical property values. The member name in *Values* table must conform to the sequence type definition. Neither extra nor missing member is allowed. *Comment* is a string. It explains what this property actually offer. If you do not need that, just leave it as "".

Returned value: None.

Function: Define a physicochemical property and make it ready to be chosen.

- **AddProperty**

Syntax: `AddProperty(PROP_ID)`

Parameter types: PROP_ID: `LUA_TSTRING`. A double-quoted Lua string or equavalent.

Returned value: None.

Function: Add a physicochemical property into current computation task. The PROP_ID must be an ID of a physicochemical property that conforms to current setting of sequence types. The PROP_ID must exist as an internal physicochemical property or a user-defined property that has already been defined using `DefineProperty` function.

- **GetActiveProperties**

Syntax: `GetActiveProperties()`

Parameter types: None.

Returned value: `LUA_TTABLE`. A Lua table object, Indices are 1-based subscripts. Values are the IDs of all physicochemical properties that have already been added into current task. These IDs are provided as `LUA_TSTRING` type values.

Function: Get all the physicochemical properties that will be used in this task.

PLEASE NOTE

There are other TDF specific functions, which we have not documented here. Users can read the source code of `LuaVMExt.cpp` for a comprehensive understanding. Those undocumented functions can be used in a TDF. However, we do not guarantee that undocumented functions can be used in future versions of UltraPse.

3.2.3 TDF specific interfaces

- **UltraPseVMConfig**

Prototype: `UltraPseVMConfig()`

Function: This is the global configuration interface. This function will be called after TDF execution, just before actual computation starts. It is desinged to perform global scale specific configuration for the Lua runtime environment itself.

- **UPseConfig**

Prototype: `UPseConfig()`

Function: This is the configuration interface for user-defined sequence representation mode. This function will be called by the internal module `lua`. This function will be called when the computational engine is instantiated. This function should return an integer, which is the number of descriptors that will be generated in the user-defined representation mode.

- **UPseCompute**

Prototype: `UPseCompute(Table_Indices)`

Function: This is the computation interface for user-defined sequence representation mode. This function will be called by the internal moduel `lua`. This function will be called on every sequence.

The Table_Indices is a table object. It contains internal indices of a sequence. The internal indice of a sequence is the zero-based dictionary order number of every notation on that sequence. For example, on a protein sequence, the internal index is for every letter on the sequence. It is the dictionary order of that letter, from 0 to 19. For another example, on a DNA sequence, the internal index is for every 2 or 3 letters according to your choice of seqeunce type. It is also the dictionary order for 2 or 4 letters.

The Table_Indices table contain a specific field, called "Length", indicating the number of indices in the table.

This function should also return a table object, which contain all the descriptors generated, and a "Length" field, indicating the number of descriptors.

- **pseb3_seq_proc**

Prototype: `pseb3_seq_proc(id, seq)`

Function: This is the computational interface for PseAAC-General legacy extension modules. All PseAAC-General Lua extension module should have this interface. There is no need to change this interface in the Lua script. The parameter and the returned result are identical to the PseAAC-General. For more details, please refer to documents of PseAAC-General.

- **pseb3_length**

Prototype: `pseb3_length()`

Function: This is the compatitble interface for PseAAC-General legacy extension modules. All PseAAC-General Lua extension module **MUST** add this interface to be executed by UltraPse. This interface should return an integer to the UltraPse host program, which indicate the number of descriptors that will be generated by the `pseb3_seq_proc` procedure.

4 Synopsis

4.1 Amino acid compositions

The following command compute only amino acid compositons for all sequences in tiny.fas

```
./upse -n stdprot -m comp -f svm -i test/tiny.fas
```

4.1 Pseudo-amino acid compositions

The following command use a pre-defined TDF to compute Classic Type-I PseAAC, with $\lambda = 10, \omega = 0.05$.

```
./upse -i test/tiny.fas -u tdfs/classic-pseaac.lua -t 1 -l 10 -w 0.05 -f svm
```

The following command use a pre-defined TDF to compute Classic Type-II PseAAC, with $\lambda = 10, \omega = 0.05$.

```
./upse -i test/tiny.fas -u tdfs/classic-pseaac.lua -t 2 -l 10 -w 0.05 -f svm
```

4.2 Pseudo-Dinucleotide compositions

The following command use a pre-defined TDF to compute Type-I PseDNC, with $\lambda = 3, \omega = 0.5$.

```
./upse -i test/tiny-dna.fas -u tdfs/psednc.lua -t 1 -l 3 -w 0.5 -f svm
```

5 Other Information

5.1 Authors correspondence

- Correspondence

Please send all correspondence regarding the UltraPse software to Dr. Pu-Feng Du, via PufengDu@gmail.com .If you find any bugs or find UltraPse can not be compiled in your environment, you should contact Dr. Pu-Feng Du via PufengDu@gmail.com. We will try our best to help you.

- Specialized Version Requests

We can produce specific versions of UltraPse for your speicial purpose. The details of this kind of request can be discussed. Please send your request to PufengDu@gmail.com.

5.2 Submitting your add-ons

If you want to contribute plugins for UltrPse, just join the project on GitHub. The address of UltraPse is: https://github.com/pufengdu/UltraPse

5.3 Resources

The following resources may be useful when you use UltraPse.

- Lua official site

http://www.lua.org/

- C++14 standard

https://isocpp.org/std/the-standard

- MinGW64 / MSYS2

http://www.msys2.org/

- PseAAC

http://www.csbio.sjtu.edu.cn/bioinf/PseAAC/

- PseAAC-General

https://github.com/pufengdu/PseAAC-General

- Pse-In-One

http://bioinformatics.hitsz.edu.cn/Pse-in-One/home/