1 CCopasiObject

Many classes in Copasi are derived of the CCopasiObject class. They all inherit some common properties:

- All Objects have a name and a type
- The objects can be organized in a hierarchical tree
- There is a mechanism to reference objects (Common Names)
- Objects can have values that can be calculated and queried with a unified mechanism
- All output in Copasi is handled with mechanisms of the CCopasiObject class

The following is a more detailed description of those properties.

1.1 Name and Type

Every CCopasiObject instance has a name and a type. Both are STL strings.

The name can be set with setObjectName() and can be accessed with getObjectName(). The type is set in the constructor of the derived classes (so there is no need for a setObjectType() method) and can be accessed with the getObjectType() method.

1.2 Hierarchical tree

CCopasiObjects can be organized in a hierarchical tree. The parent of an object is retrieved with getObjectParent(). In most cases when an object is created the parent is passed to the constructor. Usually it should not be necessary to call setObjectParent() directly.

getObjectAncestor(std::string type) looks for an object with a given type among the ancestors (parent, parent of parent, etc.) of a given object.

An object can only have children if its class is derived from CCopasiContainer (which is itself derived from CCopasiObject). For information on how to access children of objects see the documentation of CCopasiContainer.

In the current Copasi implementation there is one single hierarchical tree of CCopasiObjects. Most CCopasiObjects (except some temporary ones) are part of this tree. The root of this tree is the static pointer CCopasiContainer::Root. There is also a define which allows to access the root container as RootContainer.

1.3 Object references with common names (CNs)

All CCopasiObjects in a hierarchical tree can be referenced by a so called common name (CN). A CN is unique within the tree. The CN reflects the position of the given object in the tree. It is a STL string and is composed of the types

and names of all the direct ancestors of the object. [insert a description of how CNs actually work]

The CN is determined uniquely from the type and name of the object and from its position in a tree. It does not depend on where in the memory the object is located (like pointers would). The object doesn't even need to exist to have a valid CN. Therefore CNs can be used (and are indeed used) to reference CCopasiObjects in copasi files.

The structure of CNs also allows to identify corresponding objects in different subtrees.

The CN of an object is retrieved with the getCN() method. It returns a CCopasiObjectName object. CCopasiObjectName is essentially an STL string with some additional methods that help parsing the CN.

To locate the object for a given CN use the static CCopasiContainer::ObjectFromName(cn) method.

There is also a mechanism to handle renaming of objects (since the CN is constructed using the object name it is usually invalidated if an object is renamed): If a variable that holds a CN is declared as a CRegisteredObjectName it will be automatically added to a global static list of registered CNs. If an object that is refered to by one of the registered CNs is renamed, the CN will be changed accordingly.

1.4 Values of objects

Some CCopasiObjects have a value. The current implementation allows boolean, floating point, integer, and string values. The CCopasiObject class provides a uniform interface to calculate and access those values. Objects that have a floating point or integer value can be used in a plot.

1.4.1 Accessing the value

The isValueBool(), isValueDbl(), isValueInt(), isValueString() methods can be used to check if a given object has a value (it can only have one value of one type). The getValuePointer() method returns a pointer to this value. Since it returns a void* pointer it needs to be cast to the appropriate type: bool, C_FLOAT64, C_INT32, std::string.

The value can be set using the setObjectValue(...) method (which is declared for C_FLOAT64, C_INT32 and bool arguments).

1.4.2 Update method

Some objects need to do some extra actions when their value is set (e.g. a CMetab needs to adjust the initial particle number if the initial concentration is changed). CCopasiObject also provides an interface for that. Every object can have an update method which takes one argument (of the type corresponding to the value type of the object). This method can be set using the setUpdateMethod(*object, *member_function). It needs to be a member method

of an object. Both the pointer to the object and the member pointer of the method needs to be given to the setUpdateMethod() method. Note that the update method can be a method of a different object than the one that needs to be updated. In many cases it will be the parent object. The update method should be set in the constructor.

1.4.3 Refresh method

Some objects need to do some calculations to ensure their value is uptodate. This is not done automatically by accessing the value (since accessing the value usually happens through the pointer returned by getValuePointer()). So the functor returned by getRefresh() needs to be called explicitly before accessing the value of an object.

There may be some rules that specify that calling refresh is not necessary in some circumstances. E.g. when calling output routines during a calculation all variables of the model should already have an uptodate value. But these rules have to be documented elsewhere.

1.4.4 Dependencies

The values of some objects depend on values of other objects. These dependencies can be queried using the getDirectDependencies(), getAllDependencies(), and hasCircularDependencies() methods. CCopasiObject also has a comparison operator (operator<()) which can be used to sort a set of objects according to their dependencies (so that the refresh methods can then be called in the right order). I'm not sure this is implemented right now (April 2006).

The way this mechanism is used for output is that at the start of a calculation task a list of all objects that need to be output is generated. This list is (should be?) sorted according to the dependencies, than a sorted list of refresh functors of this objects is stored. At each output step the list of refresh functors is called in the right order, after that the output can be done using the pointers to the values (for plots) or the print() methods (for reports). By doing this it can be avoided to call a refresh functor several times if an object appears in the output several times.

1.5 Output in reports

Each CCopasiObject has a print(std::ostream*) method which is used for output in reports. Simple object that have a value just print that value to the stream, more complex objects (like numerical methods) generate a formatted output of their current settings or current state.

The refresh method (including the dependencies) should be called before calling the print() method.

1.6 CCopasiObjectReference

1.7 Hints for implementing derived classes

1.7.1 Constructors

The standard constructor of a CCopasiObject has four arguments: name, parent, type, and flags. The flags argument is an unsigned 32 bit integer that contains some flags that specify certain properties of objects (e.g. the type of its value, whether it is a container, ...). The meaning of the different bits can be found in the CCopasiObject.h header.

Constructors of derived classes should at least have the first two arguments: name and parent. The other arguments are only necessary if you expect that the class will be used as a base class for further derived classes which may want to change the flags or type.

2 CModelEntity