A Standard OpenMath C++ Library

Summary

Package

Requirements

- OpenMath C library from INRIA
- Standard C++ compliance with STL and exceptions (egcs 2.9x at least)

Files system hierarchy

The whole hierarchy (described below) is contained in the OMCPPvXXX directory.

README.html README.ps README.pdf	this readme file
LICENSE	the licence information
src/	the sources files
src/*.h ; *.cpp	C++ files
src/Makefile	the make file
doc/	the API documentation
doc/index.html	the html index of the API documentation

Installation

Step 1

Uncompress the file `OMCPPvXXX.tgz` by typing the following command:

`tar zxf OMCPPvXXX.tgz`
This will create a new directory `OMCPPvXXX/` containing the whole library hierarchy described above.

Step 2

Set the environment variable `OMCDIR` to indicate what is the directory containing the OpenMath C library:
`OMCDIR=`*`the main directory for the OpenMath C library`* (example: `OMCDIR=/root/OpenMath/OMC`)
`export OMCDIR`
You are encouraged to write these two lines above in your `.bashrc` or `/etc/profile.local` or any other initialization file to make the changes persistent.

Step 3

Go to the `src/` subdirectory then compile the OpenMath C++ library by typing the following:

```
cd OMCPPvXXX/src
make clean
make
```

Remarks

Some tests may be automatically executed at the end of `make`.
The installation should be terminated with the file `libOMCPP.a` and the pipe executable as a sample application.

Content

Conventions

A unique header file gives access to the whole library, namely `"OmHeaders.h"`.
All the public classes begin with the `"Om"` prefix.
Errors from the C library as well as the C++ library are handled by the class `OmException`.

The low structured level

The lowest level of feature provides a very light and direct encapsulation of the basic OpenMath C library. Here is:

- You can **read** OpenMath tokens through the **`OmInputDevice`** class.
- You can **write** OpenMath tokens through the **`OmOutputDevice`** class.
- You can use IO physical **streams** (files or strings) through the `OmStream` class, which is specialized in `OmInputFileStream, OmInputStringStream, OmOutputFileStream, OmOutputStringStream`.

Remarks:

The `OmInputDevice` and `OmOutputDevice` corresponds to the `OMdev` structure in the C library.

The `OmStream` corresponds to the `OMIO` structure in the C library.

The high structured level

The highest level of feature provides a structured object model to define and manipulate OpenMath objects as **tree nodes**.

There are two kinds of nodes specialized from `OmNode`:

- Leaves specialized from `OmFinalNode` which correspond to the basic OpenMath objects with no children:
  -> standard `BigInteger, ByteArray, Float, Integer, PInstruction, String, WString, Symbol, Variable`
- Nodes which correspond to the compound OpenMath objects and can have children (however some constraint rules may apply):
  -> standard `Application, Binding, Error`
  -> library specific `Object, Document`

The OpenMath **attributions** as defined in the Standard are provided here as lists of attributes associated to `OmNode` objects. Thus there is no specific class to represent attributions, the user simply append/remove attributes pairs (symbol,value) through the `OmNode` interface.

To take **comments** into account, the (non perfect) choice has been made to associate a list of comments to each `OmNode` object. By convention, comments are considered to be located *before* the object while parsing or printing OpenMath files. If this constraint of prefixing is not respected, then some loss of comments may be observed. For critical cases about perfect parsing of comments the low structured level may be preferable.

There exists three main operations on `OmNode` objects:

- You can **read** structured OpenMath objects by using the **`OmNode::resurrect`** method.
- You can **write** structured OpenMath objects by using the **`OmNode::hibernate`** method.
- You can **collect comments** by using the **`OmCommentCollector`** class.

Note that `resurrect` cannot parse comments as objects because they are distinct from structured OpenMath objects. However, comments *inside* an object are implicitely parsed by `resurrect`. In general, you *must* collect comments with `OmCommentCollector` before calling `resurrect`, and then you associate the collected comments to the resurrected object.

Bug reports

For comments, suggestions and bug reports email to *cmagnani@acm.org*