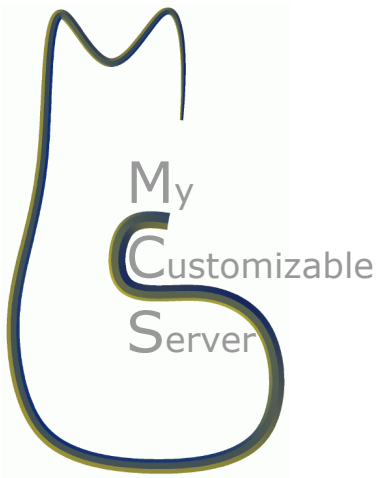
A note about MCS::VOTable classes



30 March 2007

G. Calderone[1] **and L. Nicastro**[2]
[1]*IASF–INAF, Palermo,* [2] *IASF–INAF, Bologna*

This page intentionally left blank.

1 Introduction

MCS is a C++ written library collecting classes aimed at implementing multi-thread network information server, database server, VOTable and/or FITS files access, interfaces for other programming languages, etc. A tool named MyRO allows to set up a record oriented privilege system. This way a single database table can be accessed with different privileges by different users. At the moment only MySQL is supported as database server.
VOTable classes are not yet completed. The final goal is to be able to transparently read and write VOTable or FITS files. Inclusion of the HTM and HEALPix sky pixelization scheme has been tested with success but they are not yet available to the users (require MySQL 5.1). We will also include some Astronomy specific classes by converting existing C libraries like the NOVAS, WCS and Ephem.
All the implemented classes are accessible to programming languages other than C++. Interfaces already exist for C, Fortran, IDL, PHP, Python. We plan to support Java, Perl and Tcl soon.
All these activities require time (i.e. man/woman power). We are seeking collaboration and support for this project. We believe the Virtual Observatory is the right environment for MCS to grow.

2 Reading VOTable files using the MCS library

The MCS library provides several facilities to read VOTable files using the C++ programming language. The main features are:

- capability to read files both on local or remote filesystem through HTTP and FTP in a transparent way;
- capability to parse the VOTable file one node at a time (SAX model) or as a whole (DOM model);
- no limitations on file size because the file is being processed while it is read; it is not needed to load the entire file in memory or to the local filesystem before it can be parsed;
- XML validation takes place during parsing, that is while the file is being read; this is valid in both operating mode (SAX or DOM).

The classes used to read VOTable files are: vot::Parser_Stream, to read the file one node at a time, like with a SAX parser, and vot::Parser_Tree, to read the entire file as a whole into a memory tree, like with a DOM parser.

Each node of a VOTable is represented in the C++ code by a class with the same name of the node (for example *VOTable*, *Info*, *Tabledata*, etc...). Each class has the same attributes as the VOTable node counterpart, this way you can access VOTable data using a syntax which closely resemble the VOTable names:

```
VOTable->Description()->value()

VOTable->Resource(2)->Table(1)->Data()->
         TableData()->Row(0)->Column(5)->value()
```

etc...
When reading the file using the DOM model with the vot::Parser_Tree class you can navigate the tree both vertically (that is from one node to the parent or child nodes) and following the order nodes are stored in the file.

The MCS library also provides a third way to read data from a VOTable file: the vot::Parser_Table class which, using the vot::Parser_Stream class to read the file, presents the data through the mcs::RecordSet interface, which is the standard way in MCS to present tabular data. This way you can read a VOTable with exactly the same code needed to read a database table or any other data source that implements the mcs::RecordSet interface.

The following code shows how to read a VOTable file using both the vot::Parser_Stream and the vot::Parser_Tree class:

```
//Mandatory includes
#include  <mcsvo.hh>
using namespace mcs;
using namespace vot;

int main(int argc, char* argv[]) {
  string fn = argv[1];

  //The parser (SAX model)
  Parser_Stream stream;

  //The cursor
  NodePointer node;

  //Open file and start parsing
  stream.open(fn);

  //Loop through all the nodes
  while ((node = stream.next()).element)
    //Print node contents on standard output
    node.element->print(true);


  //Do the same job, but using the DOM model
  Parser_Tree tree(false);

  //Open file and parse all nodes at once.
  tree.open(fn);

  //Recursively print all node's content
  tree.root()->print(true);

  //Retrieve a value
```

```
  cout <<
       tree.root()->child_Resource(0)->child_Table(0)
                  ->child_Field(3)->ucd()
       << endl;
}
```

Actually, the MCS library can read only tables that use the Tabledata format. A B64 encoder/decoder is already enclosed in the library, therefore support for the Binary and FITS format will be added soon. In particular the features we would like to add in the near future are:

- Possibility to specify user name and password to read VOTable files from remote host, this way other protocols, like SSL, will be usable (the library internally makes use of CURL to retrieve remote files);
- read VOTable files that use the Binary format to store data; this can be done easily through the B64 decoder and the already implemented interfaces;
- read VOTable files that use the FITS format to store data; this can be done by feeding the appropriate FITSIO library routine(s) through a pipe, then reading the output data presenting them like in the other format cases. The code needed to extract the FITS file from the VOTable stream and write it to a pipe, from a separate thread, is already present (classes mcs::Pipe and vot::VOTableReaderSplit); it is to write the code to read the file with the FITSIO library and present the data through the mcs::RecordSet interface.

Once these features will be implemented, our VOTable reader will be capable to read VOTable and FITS file transparently, without specifying which type of file it is. In both cases files can be read from local filesystem or from the network, using any of the protocols available in the CURL library. Even when a VOTable file includes a remote reference (within a Link node), the remote file will be transparently downloaded and parsed.

We are also developing a class to write VOTable files: the vot::Writer_Stream class, which will let the user write a FITS or VOTable file using the << operator. This class will also be strongly linked with the mcs::Record class, so that reading from a MCS data source (for example a database table or a MCS server) and write into a VOTable or FITS file will be quite easy.
In addition we foresee the usage of SAMP[1] to interface all the MCS classes to other VO facilities.

Finally, we recall that we have already developed a number of interfaces to use the MCS library from other programming languages such as C, Fortran, IDL, PHP, Python, and other will be added soon (especially Java and Perl). This means that when the VOTable related classes will be available (in C++), they will be automatically available to these languages too.

[1] http://www.ivoa.net/cgi-bin/twiki/bin/view/IVOA/SampInfo