EXTENDING THE GOMD DAEMON:
How to add a "support" class to the GOMD daemon

AUTHOR: Gian Paolo "rejected" Ghilardi, GOMD Team
DATE: April, the 1st 2004
HOWTO VERSION: 1.0

What's a "support class"?

A support class is a standard C++ class of the GOMD daemon implementing the support for a particular feature (i.e.: an external program like chpox).
In other words, you can imagine a "support class" like a static, hard-coded plugin for the GOMD daemon.

Step 1: Creating the header file (.h) for the support class

Create your own support class is really easy! :)

Every support class is made up of two files (an header and a source file) and follows a minimal interface.

Let's suppose we want to support a "dummy" feature.
First of all, we've to write the header (.h) file for a new support class.
Its name follows a simple convention:<SUPPORT_NAME>Support.h

```
/**************************************************************************
 *dummySupport.h-description
 * -------------------
 *begin: April, the 1st 2004
 *copyright: (C) 2003 by the gomd group at savannah.org/
 *email: jp80@libero.it, mosixview@t-online.de
 **************************************************************************/

/**
 * NOTES:
 * Created by rejected
 * Example of valid "Support" class header (for the GOMD Daemon)
 *
 * The (!) symbol means that function/var is required to conform
 * with the standard "Support"-class interface
 */

#ifndef __GOMD_DUMMY_SUPPORT_H //(!) format is __GOMD_[SUPPORT_NAME]_H
       #define __GOMD_DUMMY_SUPPORT_H

       #include <string>
       #include "utils.h"
       #include "constants.h"

       using namespace std;

       class dummySupport
       {
              //vars
              public:
              bool m_isActive; //(!) maintain the current status of this "Support" class
              utils m_util;//used to read values

              //functions
              public:
              dummySupport();//(!) default constrcutor => calls isEnabled()
                             //to get the status of the feature
              bool isEnabled();//(!) used to get the status of the
                               //feature (fills m_isActive)
       };
#endif // end of __GOMD_DUMMY_SUPPORT_H
```

Step 2: Creating the source code file (.cpp) for the support class

Now we've to create the "real" code. :)
At least we've to implement the two "standard" functions (the class constructor and the isEnabled() function) as specified in the header file.

As for the header file, we've to follow a simple convention for the name of the source code file:<SUPPORT_NAME>Support.cpp.

```
/***************************************************************************
 *dummySupport.cpp-description
 * -------------------
 *begin: April, the 1st 2004
 *copyright: (C) 2003 by the gomd group at savannah.org/
 *email: jp80@libero.it, mosixview@t-online.de
 ***************************************************************************/

/**
 * NOTES:
 * Created by rejected
 * Example of valid "Support" class source file (for the GOMD Daemon)
 */

#include "dummySupport.h"

/**
 * default constructor for dummySupport class
 */
dummySupport::dummySupport()
{
      //the constructor invokes the standard isEnabled() function
      //that checks if the features is enabled/running/installed/...
      m_isActive = isEnabled();
}

/**
 * checks if dummySupport feature is enabled on the local node
 * @return true if dummySupport is enabled on local node; false otherwise
 */
bool dummySupport::isEnabled()
{
      //NOTE: in this case we suppose the feature is enabled if we can
      //find/read a specific file. Obviously, you could create
      //your feature-specific test! :)
      if(m_util.readLineFromFile(DUMMY_STATUS_FILEPATH) == UTILS_IO_ERROR)
      {
            return false;
      }
      return true;
}
```

Step 3: Create the global instance of your support class in the daemon initializer.

Now we've to integrate the new code in the global initializer of the daemon: its class (globalInitializer) creates a single instance for each support class (following the "Singleton" Design Pattern) and initializes it. This will render the class "global" and you'll be able to use it from every point in the daemon code.

To add your support class to the daemon, just open globalInitializer.h header file, find the section with the "includes" and insert a new line with the name of the header for you new support class.

```
[...]
#include "constants.h"
#include "gomd2gomd.h"
#include "gomdSearcher.h"
#include "llSupport.h"
#include "lmsSupport.h"
#include "dummySupport.h" // <= we've inserted this line to import dummySupport stuff
#include "scxCheck.h"
[...]
```

Now we'll create the object (properly a pointer to an object as we'll create the "real" object later): find the vars-declaration section and add a new line as below.

```
[...]
chpoxSupport*              m_chpoxSupport;
chSupport*          m_chSupport;
clusterSnapshot     m_cs;
dummySupport*              m_dummySupport; // <= we've inserted this line to create a
pointer
//to a dummySupport object (we'll create this later)
gomd2gomd           m_g2g;
gomdSearcher*       m_gs;
llSupport*          m_llSupport;
lmsSupport*         m_lmsSupport;
[...]
```

Last thing is to initilize the object!
Just open globalInitializer.cpp (source file), find the constructor of this class (globalInitializer::globalInitializer(string localIP, string localHostname)) and add something like.

```
/********************************
 * DUMMY-SUPPORT object *
 ********************************/
m_dummySupport = new dummySupport();
if(dummySupport->m_isActive==true)
{
      //...DO SOMETHING...
}
```

Using your new support class!

Now you can use your brand new support class via the global instance of the globalInitializer class!

Example:

```
//[where you can get a pointer to the globalInitializer => i.e. main.cpp file]
m_globalInitializer->dummySupport->[FUNCTION/VAR/...];
```

Conclusion.

Enjoy your brand new support class!!! :)