libplugin

0.1

Generated by Doxygen 1.8.1.2

Sat Apr 20 2013 14:20:36

Contents

Chapter 1

libplugin documentation

libplugin is a simple, easy to use and portable way to handle plugin system in a C++ application. This is handled by a single little class, Plugin.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 Plugin Class Reference

The main class of the library, the plugin handling one.

`#include <plugin.h>`

Public Member Functions

- Plugin (std::string lib)

  Overloaded constructor.
- Plugin ()

  The most basic constructor.
- void set_file (std::string p_file)

  Changes the filename of the plugin file.
- int load ()

  Loads the plugin.
- int unload ()

  Unloads the plugin.
- int reload ()

  Reloads the plugin.
- int do_process (void *args)

  Processes the main function of the plugin.
- void * plugin_info ()

  Returns a pointer on custom plugin info.
- std::string get_file ()

  Returns the file name of the plugin.
- bool is_loaded ()

  Returns true if the plugin is successfully loaded.

4.1.1 Detailed Description

The main class of the library, the plugin handling one.

This class is intended to provide an interface between a dynamic plugin and your main application. Plugins file (.dll or .so depending of your OS) are loaded through this class. Plugin file shall provide the following methods :

- `int on_load()`

- `int on_process(void* args)`
- `int on_unload()`
- `void* plugininfo()`

If those methods are not provided, the loading of the plugin shall fail.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Plugin::Plugin (std::string *lib*)

Overloaded constructor.

Overloaded constructor which allow you to specify the path to the module you wish to load.

Parameters

in	*lib*	the path (absolute or relative) to the module you wish to load

4.1.2.2 Plugin::Plugin ( )

The most basic constructor.

A more simple constructor, no filename is specified.

4.1.3 Member Function Documentation

4.1.3.1 int Plugin::do_process (void * *args*)

Processes the main function of the plugin.

You can pass a struct with several fields containing all the input AND output information you want your plugin to process. The return code of the function is totally up to you, it will not make the class react in anyway. You can cast your pointer using the **OBJ_TO_VOID** macro from the *macro.h* file. In your plugin you shall use the **VOID_TO_OBJ** macro to cast it back to the type you want.

Parameters

in,out	*args*	This is a pointer on void to the parameter you want to pass to the function. This can be the cast of the pointer on a struct, an object or whatever you want.

See Also

OBJ_TO_VOID
VOID_TO_OBJ

4.1.3.2 string Plugin::get_file ( )

Returns the file name of the plugin.

4.1.3.3 bool Plugin::is_loaded ( )

Returns true if the plugin is successfully loaded.

Returns true if the plugin has successfully been loaded. If one of the functions described at the begining of this doc is missing, for exemple, the plugin won't be loaded and calls to its methods will fail.

4.1.3.4 int Plugin::load ( )

Loads the plugin.

4.1.3.5 void * Plugin::plugin_info ( )

Returns a pointer on custom plugin info.

This function acts in a similar way than *do_process*, it returns a pointer on void, which is actually a memory area which you need to cast back to a type you can use. For exemple you can imagine that this function, in your plugin will return a pointer on a custom PluginInfo class which contains information such as the author of the plugin, its version, dependecies and so on.

The cast to a pointer on an object can be done simply with the macro **VOID_TO_OBJ**.

If thre plugin is not loaded properly, this will return NULL

See Also

VOID_TO_OBJ

4.1.3.6 int Plugin::reload ( )

Reloads the plugin.

This function reloads the plugin.

This is the same than to perform the following : `plugin.unload(); plugin.load();`

4.1.3.7 void Plugin::set_file (std::string *p_file*)

Changes the filename of the plugin file.

Parameters

in	*p_file*	Filename (relative or absolute)

4.1.3.8 int Plugin::unload ( )

Unloads the plugin.

Unloads the plugin by freeing the handle on the dynamic library. This will call the on_unload() method of the plugin.

The documentation for this class was generated from the following files:

- include/libplugin/plugin.h
- src/plugin.cpp

Chapter 5

File Documentation

5.1 include/libplugin.h File Reference

```
#include "libplugin/macros.h"
#include "libplugin/plugin.h"
```

5.2 include/libplugin/macros.h File Reference

Macros

- #define OBJ_TO_VOID(obj, out) void ∗out = (void∗)obj;
- #define VOID_TO_OBJ(vd, type, out) type ∗out = (type∗) vd;
- #define PLUGIN_NOT_LOADED -1

5.2.1 Macro Definition Documentation

5.2.1.1 #define OBJ_TO_VOID(*obj, out*) void ∗out = (void∗)obj;

This macro is used to cast a pointer onto an object 'obj' to a pointer to void 'out' (of course you can change the names)

5.2.1.2 #define PLUGIN_NOT_LOADED -1

A simple define value

5.2.1.3 #define VOID_TO_OBJ(*vd, type, out*) type ∗out = (type∗) vd;

This macro is used to cast a pointer onto void 'vd' to a pointer onto an object 'out' with the type 'type'. So if I call `VOID_TO_OBJ(arg, char, str)` then it will be the same as `char* str = (char*) arg` and it will cast arg to a string.

5.3 include/libplugin/plugin.h File Reference

Header to include to use the plugin class.

```
#include <iostream>
#include <string>
#include <sstream>
#include "macros.h"
```

Classes

- class Plugin

  The main class of the library, the plugin handling one.

5.3.1 Detailed Description

Header to include to use the plugin class.

Author

Thomas Maurice tmaurice59@gmail.com

Version

0.1

5.4 src/plugin.cpp File Reference

```
#include <plugin.h>
```

5.4.1 Detailed Description

Author

Thomas Maurice

Index