Pugg 1.0.0

A Simple C++ Framework to load classes from dll files in an OO way.

Contact For bugs, feature requests and other stuff:
Tunç Bahçecioğlu tunc.bahcecioglu@gmail.com

- Pugg @ Bitbucket
- Pugg @ Sourceforge
- Pugg Website

Credits

- nuclex blog for giving me the idea and the basic source code to implement pugg.
- Sebastian (hackspider) for bugfix.
- Szymon Janikowski for fix in old tutorial.
- Alex Elzenaar for code improvements and code for Linux support.

Features

- Auto loading of dll files and classes
- Version control for loaded plugins
- Header only library
- OOP Design
- Platform independent

Requirements

- Somewhat current C++ compiler (No C++11 features were used in this version.)

Installation

No installation is required. Files to include are provided in the include folder of the release. On some Linux platforms you need to link with libdl library.

In order to build example projects use CMake

Tutorial

Suppose you have an animal class:

```
class Animal
{
public:
    Animal() {}
    virtual ~Animal() {}

    virtual std::string kind() = 0;
    virtual bool can_swim() = 0;
};
```

You want to load subclasses of Animal from dll files.

Add a version and a name to create a server:

```
class Animal
{
public:
    // ....
    static const int version = 1;
    static const std::string server_name() {return "AnimalServer";}
};
```

Then add a driver to tell the name and version of the server to Pugg.

```
class AnimalDriver : public pugg::Driver
```

```
class AnimalDriver : public pugg::Driver
{
public:
    AnimalDriver(std::string name, int version) :
        pugg::Driver(Animal::server_name(),name,version) {}
    virtual Animal* create() = 0;
};
```

Now in a separate project create your dll. Lets create a cat and its driver:

```
class Cat : public Animal
{
public:
    std::string kind() {return "Cat";}
    bool can_swim() {return false;}
};

class CatDriver : public AnimalDriver
{
public:
    CatDriver() : AnimalDriver("CatDriver", Cat::version) {}
    Animal* create() {return new Cat();}
};
```

Note that version is the same version from the Animal class at the time Cat is compiled. If Animal class changes in the future and increases its version Cat class will be outdated and won't be loaded by Pugg.

Let's export the Cat class from dll.

```
#include <pugg\Kernel.h>
#include "Cat.h"

#ifdef _WIN32
#  define EXPORTIT __declspec( dllexport )
#else
#  define EXPORTIT
#endif

extern "C" EXPORTIT void register_pugg_plugin(pugg::Kernel* kernel)
{
    kernel->add_driver(new CatDriver());
}
```

register_pugg_plugin function is a special function called by Pugg to load drivers from a dll file.

Now in our main app, we can import dlls and have fun

```
#include <pugg\Kernel.h>
#include <Animal.h>

#include <iostream>
#include <vector>

using namespace std;

int main()
{
    cout << "Loading plugins..." << endl;
    pugg::Kernel kernel;
    kernel.add_server(Animal::server_name(), Animal::version);
#ifdef WIN32
    kernel.load_plugin("PangeaAnimals.dll"); // Cat class is in this dll.
#else
    kernel.load_plugin("libPangeaAnimals.so");
#endif

    // we can load all drivers from a specific server
    vector<AnimalDriver*> drivers =kernel.get_all_drivers<AnimalDriver>(Animal::server_name());
    // to load a specific driver
```

```
    // AnimalDriver* animal_driver = kernel.get_driver<AnimalDriver>(Animal::server_name, "CatDriver");
    vector<Animal*> animals;
    for (vector<AnimalDriver*>::iterator iter = drivers.begin(); iter != drivers.end(); ++iter) {
        AnimalDriver& driver = *(*iter);
        animals.push_back(driver.create());
    }

    for(vector<Animal*>::iterator iter = animals.begin(); iter != animals.end(); ++iter) {
        Animal& animal = *(*iter);
        cout << "Animal kind = " << animal.kind() << endl;
        cout << "Can Animal Swim = " << animal.can_swim() << endl;
    }

    for(vector<Animal*>::iterator iter = animals.begin(); iter != animals.end(); ++iter) {
        delete *iter;
    }
}
```

Class Documentation

Driver

Driver class registers user class to Pugg.

```
// /include/pugg/Driver.h
namespace pugg {
class Driver
{
public:
    // server_name : Name of the server driver belongs to.
    // name        : Name of the driver
    // version     : version of the driver
    Driver(std::string server_name, std::string name, int version)
    virtual ~Driver() {}

    std::string server_name() const {return _server_name;}
    std::string name() const {return _name;}
    int version() const {return _version;}
};
}
```

Kernel

```
// /include/pugg/Kernel.h
namespace pugg {

class Kernel
{
public:
    ~Kernel(); // destroys drivers, servers and links to dll files.
    // loads drivers from the dll file
    bool load_plugin(const std::string& filename);
    // adds a server
    // name               : name of the server
    // min_driver_version : minimum driver version to load to server.
    //                      Lower version drivers are not loaded.
    void add_server(std::string name, int min_driver_version );
    // adds a driver. usually called in the register_pugg_plugin function of dlls.
    bool add_driver(pugg::Driver* driver);

    // gets a specific driver from a server
    template <class DriverType>
    DriverType* get_driver(const std::string& server_name, const std::string& name);
    // gets all drivers from a server
    template <class DriverType>
    std::vector<DriverType*> get_all_drivers(const std::string& server_name);
    // clears all drivers
    void clear_drivers();
    // clears all drivers, servers and plugins
```

```
    void clear();
};

}
```

Release Notes

1.0.0

Added clear methods. 1.0.0 release.

0.60

Finally Pugg supports Linux. CMAKE is used for creating the example projects on different platforms.

0.50

This version is an attempt to get back to Pugg development. Thus I replaced most of the code from the previous version with the version I use in my own projects. I omitted C++11 code (mostly auto and unique_ptr) in order not to break old compilers.

Wstring support is removed as it complicates the code a and I don't think it is used by anyone (If I'm wrong mail me and I will add it back).

Name convention changes break old code but it won't be very hard to replace old code.

History

1.0.0

- Main repository is moved to Bitbucket.
- Added clear methods.
- 1.0.0 release. 0.60
- Linux support
- Bug fixes, Code improvements etc...

0.55

- Switched to CMAKE build system for example projects.

0.51

- Removed unnecessary try catch block from Plugin.cpp

0.5

- New Documentation
- Changed to Boost license for more freedom.
- Name convention changes
- wstring support is removed
- Server class is in Kernel.h and is used from Kernel class.
- Changed to mercurial

0.41

- A memory leak occurring while trying to load dll files not having the registerPlugin function is fixed.

0.4

- std::wstring file name support

0.3

- Name convention changes
- Release pack for the library and related example projects are put together
- More source code Documentation

0.21

- Bug Fix related to UNICODE support

0.2

- Version control system on plug-in system is removed. Every driver's version is controlled separately.
- Server class became template, removing the need to write a server class for every driver and casting the drivers before usage.

0.11

- Bug fixes

0.1

- Initial release