Subject: Question: Simple plugin implementation
Posted by koldo on Sun, 23 Jan 2011 21:20:33 GMT
View Forum Message <> Reply to Message

Hello all

I will ask you the question with an example based in OfficeAutomation:

There is an API of functions to handle spreadsheets. As they can be handled using OpenOffice, LibreOffice, Excel or other programs, the same API can be set for all programs. In run time it is possible to choose which program to use.

To do it, I have prepared something similar to the enclosed code. However, I think it is ugly and something much better could be done in U++ using C++.

Do you have any idea?

```
#include <Core/Core.h>

using namespace Upp;


class SpreadsheetPlugin {
public:
 virtual bool Open(char *filename)    {return false;};
 virtual bool SetData(int row, int col, Value val)  {return false;};
};


class OpenSpreadsheet : public SpreadsheetPlugin {
 virtual bool Open(char *filename);
 virtual bool SetData(int row, int col, Value val);
};

bool OpenSpreadsheet::Open(char *filename) {
 // Do stuff
}

bool OpenSpreadsheet::SetData(int row, int col, Value val) {
 // Do stuff
}

class ExcelSpreadsheet : public SpreadsheetPlugin {
 virtual bool Open(char *filename);
 virtual bool SetData(int row, int col, Value val);
};

bool ExcelSpreadsheet::Open(char *filename) {
```

```
 // Do stuff
}

bool ExcelSpreadsheet::SetData(int row, int col, Value val) {
 // Do stuff
}


class Spreadsheet : public SpreadsheetPlugin {
private:
 SpreadsheetPlugin *data;

public:
 Spreadsheet() {data = 0;};
 ~Spreadsheet() {
  if (data)
   delete data;
 }
 void Init(String type) {
  if (type == "Open" || type == "Libre")
   data = new OpenSpreadsheet();
  else
   data = new ExcelSpreadsheet();
 }

 virtual bool Open(char *filename)    {return data->Open(filename);}
 virtual bool SetData(int row, int col, Value val) {return data->SetData(row, col, val);}
};


CONSOLE_APP_MAIN
{
 Spreadsheet spreadsheet;

 spreadsheet.Init("Libre");
 spreadsheet.Open("c:\\myfile.xls");
 spreadsheet.SetData(4, 6, "Hello world");
}
```

Subject: Re: Question: Simple plugin implementation
Posted by mirek on Sun, 23 Jan 2011 22:53:54 GMT
View Forum Message <> Reply to Message

Well, this is pretty standard aprroach in fact. And we are using something quite similar e.g. in image format support.

You can make it a little bit nicer using

```
class Spreadsheet : public SpreadsheetPlugin {
private:
 One<SpreadsheetPlugin> data;
```

or perhaps you can avoid Spreadsheet and provide

```
CreateSpreadsheet(One<Spreadsheet>& x, String type);
```

but it is hard to say what is better...

Mirek

Subject: Re: Question: Simple plugin implementation
Posted by koldo on Mon, 24 Jan 2011 08:44:27 GMT
View Forum Message <> Reply to Message

Oooh, I thought it was possible to do it smarter with templates.

Subject: Re: Question: Simple plugin implementation
Posted by koldo on Tue, 25 Jan 2011 13:50:22 GMT
View Forum Message <> Reply to Message

Hello

This is other focus. It permits adding plugins just by adding files without touching base class. It is based on Painter example.

For example to add Excel support, it would be:

```
#include <Core/Core.h>

using namespace Upp;

#include "Spreadsheet.h"

class ExcelSpreadsheet : public SpreadsheetPlugin {
 Spreadsheet_METHOD_LIST
};

ExcelSpreadsheet excelSpreadsheet;
```

```
INITBLOCK {
 RegisterPlugin("Excel", dynamic_cast<SpreadsheetPlugin *>(&excelSpreadsheet));
}

bool ExcelSpreadsheet::Open(const char *filename) {
 puts("ExcelSpreadsheet::Open");
 return false;
}

bool ExcelSpreadsheet::SetData(int row, int col, Value val) {
 puts("ExcelSpreadsheet::SetData");
 return false;
}
```

File Attachments

1) PluginDemo.7z, downloaded 329 times

Subject: Re: Question: Simple plugin implementation
Posted by mirek on Tue, 25 Jan 2011 14:34:39 GMT
View Forum Message <> Reply to Message

koldo wrote on Tue, 25 January 2011 08:50Hello

This is other focus. It permits adding plugins just by adding files without touching base class. It is based on Painter example.

For example to add Excel support, it would be:

```
#include <Core/Core.h>

using namespace Upp;

#include "Spreadsheet.h"

class ExcelSpreadsheet : public SpreadsheetPlugin {
 Spreadsheet_METHOD_LIST
};

ExcelSpreadsheet excelSpreadsheet;

INITBLOCK {
 RegisterPlugin("Excel", dynamic_cast<SpreadsheetPlugin *>(&excelSpreadsheet));
}
```

```
bool ExcelSpreadsheet::Open(const char *filename) {
 puts("ExcelSpreadsheet::Open");
 return false;
}

bool ExcelSpreadsheet::SetData(int row, int col, Value val) {
 puts("ExcelSpreadsheet::SetData");
 return false;
}
```

I guess even better pattern can be found with RasterImage...

Subject: Re: Question: Simple plugin implementation
Posted by koldo on Thu, 27 Jan 2011 09:34:14 GMT
View Forum Message <> Reply to Message

Hello Mirek

A little bit better (not as rich as StreamRaster, but clearer for me). Added "new" instead of ugly global var:

```
#include <Core/Core.h>

using namespace Upp;

#include "Spreadsheet.h"

class ExcelSpreadsheet : public SpreadsheetPlugin {
 Spreadsheet_METHOD_LIST
};


INITBLOCK {
 RegisterPlugin<ExcelSpreadsheet>("Excel");
}

bool ExcelSpreadsheet::Open(const char *filename) {
 puts("ExcelSpreadsheet::Open");
 return false;
}

bool ExcelSpreadsheet::SetData(int row, int col, Value val) {
```

```
 puts("ExcelSpreadsheet::SetData");
 return false;
}
```

However all possible plugins are initialized, not just the one to be used.

It is the same in class StreamRaster (file Raster.h):

template <class T> static StreamRaster *FactoryFn() { return new T; }

Every registered class has to be initialized.

[Edit: Simplified INITBLOCK with templates]

Subject: Re: Question: Simple plugin implementation
Posted by koldo on Thu, 27 Jan 2011 12:58:36 GMT
View Forum Message <> Reply to Message

Hello Mirek

One question. Following with StreamRaster, it seems registered classes are never deleted:

Here PNGRaster is registered
StreamRaster::Register<PNGRaster>();

So
static void Register() { AddFormat(&StreamRaster::FactoryFn<T>); }

FactoryFn creates a new T
template <class T> static StreamRaster *FactoryFn() { return new T; }

Added here:
```
void StreamRaster::AddFormat(RasterFactory factory)
{
 INTERLOCKED_(sAnyRaster)
  Map().Add((void *)factory);
}

Vector<void *>& StreamRaster::Map()
{
 static Vector<void *> x;
 return x;
}
```

So it is assigned to a void *

Subject: Re: Question: Simple plugin implementation
Posted by fudadmin on Thu, 27 Jan 2011 16:34:03 GMT
View Forum Message <> Reply to Message

Can these things to be made as universal plugins implementation for upp?

Subject: Re: Question: Simple plugin implementation
Posted by mirek on Thu, 27 Jan 2011 19:53:53 GMT
View Forum Message <> Reply to Message

koldo wrote on Thu, 27 January 2011 07:58Hello Mirek

One question. Following with StreamRaster, it seems registered classes are never deleted:

What you register is the "factory function". What would you want to delete?

Mirek

Subject: Re: Question: Simple plugin implementation
Posted by mirek on Thu, 27 Jan 2011 19:54:32 GMT
View Forum Message <> Reply to Message

fudadmin wrote on Thu, 27 January 2011 11:34Can these things to be made as universal plugins implementation for upp?

If you have in mind something like .dll, then the answer is NO.

Mirek

Subject: Re: Question: Simple plugin implementation
Posted by koldo on Thu, 27 Jan 2011 20:47:04 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 27 January 2011 20:53koldo wrote on Thu, 27 January 2011 07:58Hello Mirek

One question. Following with StreamRaster, it seems registered classes are never deleted:

What you register is the "factory function". What would you want to delete?

Mirek

Hello Mirek

If to register a new class FactoryFn does a new T that is added to Map() to a static Vector<void *> x, I have not seen how it is deleted.

Subject: Re: Question: Simple plugin implementation
Posted by mirek on Fri, 28 Jan 2011 09:33:47 GMT
View Forum Message <> Reply to Message

koldo wrote on Thu, 27 January 2011 15:47mirek wrote on Thu, 27 January 2011 20:53koldo wrote on Thu, 27 January 2011 07:58Hello Mirek

One question. Following with StreamRaster, it seems registered classes are never deleted:

What you register is the "factory function". What would you want to delete?

Mirek

Hello Mirek

If to register a new class FactoryFn does a new T that is added to Map() to a static Vector<void *> x, I have not seen how it is deleted.

That 'void *' is a pointer to a function. What do you want to delete?

If Vector itself, it gets deleted with all other static objects.

Mirek

Subject: Re: Question: Simple plugin implementation
Posted by mirek on Fri, 28 Jan 2011 09:34:51 GMT
View Forum Message <> Reply to Message

To be more specific:

```
void Fn() {
  ....
}

void (*fn)() = &Fn;
```

delete fn; //WTF?

Subject: Re: Question: Simple plugin implementation
Posted by koldo on Sat, 29 Jan 2011 00:08:18 GMT
View Forum Message <> Reply to Message

Hello Mirek

After a couple of hours looking like an stupid Raster.h and Raster.cpp... , you are right: Registering a new bitmap class in U++ is including a pointer to a function that will create an object of that class when necessary ...

As the bitmap class is always assigned to a One<StreamRaster>, deletion is assured

---

Subject: Re: Question: Simple plugin implementation
Posted by koldo on Mon, 07 Feb 2011 10:16:01 GMT
View Forum Message <> Reply to Message

He he

Following with it here it is something perhaps better than StreamRaster plugin system.

In StreamRaster it is done a "new" any time a function is called. But is is possible to do it only once:

```
class StaticPlugin {
public:
 StaticPlugin();
 ~StaticPlugin();
 bool Init(const char *name);

 template <class T>
 static void Register(const char *name) {
  PluginData& x = Plugins().Add();
  x.name = name;
  x.New = New<T>;
  x.Delete = Delete<T>;
 }

protected:
 void *data;
```

```
private:
 String name;

 template <class T> static void *New()   {return new T;};
 template <class T> static void Delete(void *p)  {delete static_cast<T *>(p);};

 struct PluginData {
  String name;
  void *(*New)();
  void (*Delete)(void *);
 };

 static Array<PluginData>& Plugins();
};
```

The plugin class is stored in a void *data, with its "new" and "delete", in Register() (thanks to templates).

This way, if a plugin is used, StaticPlugin uses the right "new". And in ~StaticPlugin it is used the right "delete" .

Subject: Re: Question: Simple plugin implementation
Posted by mirek on Fri, 11 Feb 2011 15:50:37 GMT
View Forum Message <> Reply to Message

Well, but then you keep all plugins in memory, including all data in memory they needed for the last operation or you need to implement some sort of 'free' for them.

Allocating/deallocating memory is actually quite fast operation in U++.

Subject: Re: Question: Simple plugin implementation
Posted by koldo on Fri, 11 Feb 2011 21:47:06 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 11 February 2011 16:50Well, but then you keep all plugins in memory, including all data in memory they needed for the last operation or you need to implement some sort of 'free' for them.

Allocating/deallocating memory is actually quite fast operation in U++.
Hello Mirek

It is only necessary to allocate the used plugin. It is done in
INITBLOCK {

```
 RegisterPlugin<ExcelSpreadsheet>("Excel");
}
```

In fact it is interesting to have it allocated while the plugin is used as it can keep some variables.

For example for an spreadsheet, the Open() method load some variables that will be used by other methods like SetData(Value v, int row, int col).
