
Subject: Polymorphic XML

Posted by [mdelfede](#) on Mon, 11 Jan 2010 12:57:32 GMT

[View Forum Message](#) <> [Reply to Message](#)

I need to save/restore a class hierarchy to/from stream; the result should be something like that :

```
<Pages>
  <Page class = "myclass">
    ....here object content, as per Xmlize
  </Page>
</Pages>
```

I already have a class factory able to create class by name, but I don't know how to put all together. In particular, I need to work with streams, not files; the code should be able to read/write from the current byte of stream, and leave it positioned on next data.

Assuming 'Page' is my hierarchy base class, it should be :

```
Page *CreatePage(const char *className); <== DONE
```

```
Array<String> GetAvailablePages(void); <== DONE
```

```
class page
{
protected:
    virtual void Xmlize(XmlIo xml) = 0; <== EASY, I think
public:
    static Page *LoadStream(Stream &s); <== ???
    virtual bool SaveStream(Stream &s); <== ???
}
```

Any hint on how to do it ?

Ciao

Max

Subject: Re: Polymorphic XML

Posted by [mdelfede](#) on Mon, 11 Jan 2010 19:26:30 GMT

[View Forum Message](#) <> [Reply to Message](#)

Well, got it !

Here the relevant code :

Panel containing pages, which are derived from 'Page' base class,
xmlize contained pages and restore page list on load :

```
void JobPane::Xmlize(XmlIO xml)
{
    xml
    ("Pages", pages)
    ;
    if(xml.IsLoading())
    {
        for(int i = 0; i < pages.GetCount(); i++)
            pageList.Add(pages[i].GetPosition() + "-" + pages[i].GetDescription());
        pageList.SetCursor(pageList.GetCount()-1);
    }
}
```

Base class definition, relevant parts :

```
// Page base class -- inside each jobpane along with pagelist
class Page : public ParentCtrl
{
    friend void __SetPageClass(Page *page, const char *className);
private:
    const char *className;

protected:
    .....
public:

    typedef Page CLASSNAME;

    // constructor
    Page();

    // gets ascii name of page class
    String GetClassName(void) { return className; }
    .....
    // xml i/o
    virtual void Xmlize(XmlIO xml) {}

}; // END Class Page
```

Note, here Xmlize does nothing, as the base class is just empty.
Could do more, indeed.

Now the polymorphic array of pages :

```
class Pages : public Array<Page>
{
public:

    void Xmlize(XmlIO xml);
};
```

And the xmlize part for Pages class, which does the trick :

```
// xmlize pages
void Pages::Xmlize(XmlIO xml)
{
    if(xml.IsStoring())
    {
        for(int i = 0; i < GetCount(); i++)
        {
            Page &page = operator[](i);
            String tag = page.GetClassName();
            ::Xmlize(xml.Add(tag), page);
        }
    }
    else
    {
        Clear();
        for(int i = 0; i < xml->GetCount(); i++)
        {
            if(xml->Node(i).IsTag())
            {
                String tag = xml->Node(i).GetTag();
                Page *page = CreatePage(tag);
                if(page)
                {
                    page->Xmlize(xml.At(i));
                    Add(page);
                }
            }
        }
    }
}
```

CreatePage(const String &className) is the class factory, which creates an empty Page-derived class of type 'className'

Here the files, PageFactory.h

```
class Page;
void __LamelRegisterPage(const char *className, Page *(*Create)(void), const char *desc, int
index);
void __SetPageClass(Page *page, const char *className);

#define LAMELL_REGISTER(pageClass, pageDesc, pageIndex) \
Page * __##_pageClass##_Create(void) \
{ \
Page *page = (Page *)new pageClass; \
__SetPageClass(page, #pageClass); \
return page; \
} \
INITBLOCK { \
__LamelRegisterPage(#pageClass, __##_pageClass##_Create, pageDesc, pageIndex);\
}

Page *CreatePage(const char *className);
Page *CreatePage(int idx);
Array<String> GetAvailablePages(void);
```

And PageFactory.cpp (important part) :

```
struct __LamelPageFactory
{
// class creation
Page *(*Create)(void);

// class name
String className;

// page description
String desc;

// page 'index' used to sort
// pages in list
int index;
};

typedef ArrayMap<String, __LamelPageFactory> __LamelPageArray;

__LamelPageArray &LamelPageFactory()
{
```

```

static One<__LamelPageArray> pages;

if(!pages)
    pages = new __LamelPageArray;

return *pages;
}

void __LamelRegisterPage(const char *className, Page *(*Create)(void), const char *desc, int
index)
{
    __LamelPageArray &pages = LamelPageFactory();

    __LamelPageFactory *page = new __LamelPageFactory;
    page->Create = Create;
    page->className = className;
    page->desc = desc;
    page->index = index;
    pages.Add(className, page);
}

void __SetPageClass(Page *page, const char *className)
{
    page->className = className;
}

Page *CreatePage(const char *className)
{
    __LamelPageArray &pages = LamelPageFactory();
    __LamelPageFactory &f = pages.Get(className);
    Page *p = f.Create();

    // update units upon page creation
    globalSettings().PropagateUnitsChanges();
    return p;
}
.....

```

You must register each Page derived class (in its source file) with this :

```
LAMELL_REGISTER(Trave, t_("Simply supported beam"), 10);
```

Where 'Trave' is the derived class type, the text is a description and the number is just an index (both non-important, just part of my app)

As a note, each Page derived class MUST define Xmlize() AND call

it's base class Xmlize function; as example, here :

```
class Trave : public WithTraveLayout<Page>
{
    int test;
public:
    typedef Trave CLASSNAME;

    Trave();

    // xml i/o
    void Xmlize(XmlIO xml) { this->Page::Xmlize(xml); xml("test", test);}
};
```

With all that stuff, from 'JobPane' class you can load/save it's pages simply with:

```
// read/saves job on file
bool JobPane::Load(String const &fileName)
{
    return LoadFromXMLFile(*this, fileName);
}

bool JobPane::Save(String const &fileName)
{
    return StoreAsXMLFile(*this, "LAMELL", fileName);
}
```

Hope it can be useful for somebody

Ciao

Max

Subject: Re: Polymorphic XML
Posted by [mirek](#) on Tue, 12 Jan 2010 07:05:32 GMT
[View Forum Message](#) <> [Reply to Message](#)

```
__LamelPageArray &LamelPageFactory()
{
    static One<__LamelPageArray> pages;

    if(!pages)
```

```
pages = new __LamelPageArray;

return *pages;
}
```

Why not

```
__LamelPageArray &LamelPageFactory()
{
    static __LamelPageArray pages;
    return pages;
}
```

?

Also, I believe that

```
void CreateClassInstance(One<BaseClass>& x);
```

(Note that you can do x.Create<DerivedClass>())

or even in some cases

```
void CreateClassInstance(Array<BaseClass>& x);
```

is cleaner class factory interface. (But depends on your taste).

Mirek

Subject: Re: Polymorphic XML
Posted by [mdelfede](#) on Tue, 12 Jan 2010 09:42:08 GMT
[View Forum Message](#) <> [Reply to Message](#)

```
luzr wrote on Tue, 12 January 2010 08:05
__LamelPageArray &LamelPageFactory()
{
    static One<__LamelPageArray> pages;

    if(!pages)
        pages = new __LamelPageArray;

    return *pages;
}
```

```
}
```

Why not

```
__LamelPageArray &LamelPageFactory()  
{  
    static __LamelPageArray pages;  
    return pages;  
}
```

?

Yep, better... and it could be made a static member of Page class, also.

Quote:

Also, I believe that

```
void CreateClassInstance(One<BaseClass>& x);
```

(Note that you can do x.Create<DerivedClass>())

or even in some cases

```
void CreateClassInstance(Array<BaseClass>& x);
```

is cleaner class factory interface. (But depends on your taste).

Mirek

mhhh... I don't understand your point. I need the class creation at runtime by name, so an ascii string, not a type.

Anyways, thinking about it, all the stuff could be polished a bit, also using callbacks, maybe, instead of function pointers.

BTW, yesterday I was in a hurry because the problem was starting boring me, so I solved it quick and dirty

Ciao

Max

Subject: Re: Polymorphic XML
Posted by [mirek](#) on Tue, 12 Jan 2010 13:22:04 GMT
[View Forum Message](#) <> [Reply to Message](#)

mdelfede wrote on Tue, 12 January 2010 04:42Quote:
Also, I believe that

```
void CreateClassInstance(One<BaseClass>& x);
```

(Note that you can do x.Create<DerivedClass>())

or even in some cases

```
void CreateClassInstance(Array<BaseClass>& x);
```

is cleaner class factory interface. (But depends on your taste).

Mirek

mhhh... I don't understand your point. I need the class creation at runtime by name, so an ascii string, not a type.

No, it is not about the class name, but you avoid "naked" pointer (naked in sense that it points to the heap and is the only reference to the heap object).