Subject: Helper for internazionalize arrays of literals
Posted by mdelfede on Sat, 09 Jan 2010 13:59:43 GMT
View Forum Message <> Reply to Message

Starting to internationalize an app, I faced to this problem :

```
char *myTable =
{
    "ONE",
    "TWO",
    "THREE"
};
```

which can't be internationalized by aid of 't_' macro. It can be done with 'tt_' macro, but it need patched code to retrieve the internationalized string with GetLngString macro.
So I've thought about a better way, and coded this :

```
class StringTable : public Vector<String>
{
 public:
  StringTable &operator,(const char *s) { Add(s); return *this; }
  const char *operator[](int i) { ASSERT(i < GetCount()); return ~At(i); }
};

#define STRINGTABLE(s) StringTable s; INITBLOCK { s,
#define ENDTABLE ; }
```

This allows to define a character table like that :

```
STRINGTABLE(myTable)
    t_("One"),
    t_("Two"),
    t_("Three")
ENDTABLE;
```

Access is as before with myTable[], so no need to code changes; as a small benefit, the string translation is done at load time just once, and not every time the string is needed.
What do you think about ? Do you know a better way to achieve the same purpose ?

Ciao

Max

Subject: Re: Helper for internazionalize arrays of literals
Posted by mdelfede on Sat, 09 Jan 2010 20:39:57 GMT
View Forum Message <> Reply to Message

This one corrects a subtle bug and delays loading of strings (needed to setup properly the language) :

```
class StringTable : public Vector<String>
{
 public:
  StringTable &operator,(const char *s) { Add(s); return *this; }
  String operator[](int i) { ASSERT(i < GetCount()); return GetLngString(At(i)); }
};

#define STRINGTABLE(s) StringTable s; INITBLOCK { s,
#define ENDTABLE ; }
```

Stringtable must be defined with tt_ :

```
STRINGTABLE(Test)
    tt_("One"),
    tt_("Two"),
    tt_("Three")
ENDTABLE;
```

The tt_ macros are needed just to make theide export the translation; can be avoided if no need to sync translations.

Ciao

Max

Subject: Re: Helper for internazionalize arrays of literals
Posted by mirek on Sun, 10 Jan 2010 13:26:06 GMT
View Forum Message <> Reply to Message

mdelfede wrote on Sat, 09 January 2010 15:39This one corrects a subtle bug and delays loading of strings (needed to setup properly the language) :

```
class StringTable : public Vector<String>
{
 public:
  StringTable &operator,(const char *s) { Add(s); return *this; }
```

```
  String operator[](int i) { ASSERT(i < GetCount()); return GetLngString(At(i)); }
};

#define STRINGTABLE(s) StringTable s; INITBLOCK { s,
#define ENDTABLE ; }
```

Stringtable must be defined with tt_ :

```
STRINGTABLE(Test)
    tt_("One"),
    tt_("Two"),
    tt_("Three")
ENDTABLE;
```

The tt_ macros are needed just to make theide export the translation; can be avoided if no need to sync translations.

Ciao

Max

Still not sure whether all of this is worth the trouble. Does calling GetLngString directly so much difference?

Subject: Re: Helper for internazionalize arrays of literals
Posted by mirek on Sun, 10 Jan 2010 13:40:54 GMT

View Forum Message <> Reply to Message

[quote title=luzr wrote on Sun, 10 January 2010 08:26]mdelfede wrote on Sat, 09 January 2010 15:39This one corrects a subtle bug and delays loading of strings (needed to setup properly the language) :

```
class StringTable : public Vector<String>
{
 public:
  StringTable &operator,(const char *s) { Add(s); return *this; }
  String operator[](int i) { ASSERT(i < GetCount()); return GetLngString(At(i)); }
};

#define STRINGTABLE(s) StringTable s; INITBLOCK { s,
#define ENDTABLE ; }
```

Stringtable must be defined with tt_ :

```
STRINGTABLE(Test)
    tt_("One"),
    tt_("Two"),
    tt_("Three")
ENDTABLE;
```

The tt_ macros are needed just to make theide export the translation; can be avoided if no need to sync translations.

Ciao

Max

Still not sure whether all of this is worth the trouble....

BTW, there IMO could be even more effective solutions. E.g.:

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

using namespace Upp;

struct tt_char {
 const char *s;

 String ToString() const       { return GetLngString(s); }
 operator const char *() const { return ToString(); }
};

CONSOLE_APP_MAIN
{
 static tt_char x[] = {
  tt_("Aborted by user."),
  tt_("Two"),
  tt_("Three")
 };

 SetLanguage(LNG_('I','T','I','T'));
 DDUMP(x[0]);
}
```

Subject: Re: Helper for internazionalize arrays of literals
Posted by mdelfede on Mon, 11 Jan 2010 07:25:32 GMT
View Forum Message <> Reply to Message

luzr wrote on Sun, 10 January 2010 14:40BTW, there IMO could be even more effective solutions.
E.g.:

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

using namespace Upp;

struct tt_char {
 const char *s;

 String ToString() const       { return GetLngString(s); }
 operator const char *() const { return ToString(); }
};

CONSOLE_APP_MAIN
{
 static tt_char x[] = {
  tt_("Aborted by user."),
  tt_("Two"),
  tt_("Three")
 };

 SetLanguage(LNG_('I','T','I','T'));
 DDUMP(x[0]);
}
```

mhhhh.... right, yours is much better. I was wrongly thinking that wasn't possible to use a static initializer with a class.
BTW, imho it's worth the trouble because you don't need to remember using the GetLngString on every places you need it... More, with your solution you prcatically don't have any overhead to achieve this.

Ciao

Max

Subject: Re: Helper for internazionalize arrays of literals
Posted by mdelfede on Mon, 11 Jan 2010 10:09:07 GMT
View Forum Message <> Reply to Message

BTW, I had to change it like this :

```
struct StringTable {
 const char *s;

 String ToString() const       { return GetLngString(s); }
 operator const char *() const { return ToString(); }
 operator String() { return ToString(); }
 operator Value() { return ToString(); }
};
```

Otherwise it didn't work here :

```
DropList d;
int i = 0;
d.Add(i, myStringTable[i]);
```

Don't know exactly why he didn't pick automatically the char * --> Value conversion. Even adding the String() operator wasn't enough.

Ciao

Max

Subject: Re: Helper for internazionalize arrays of literals
Posted by mirek on Mon, 11 Jan 2010 10:12:40 GMT
View Forum Message <> Reply to Message

mdelfede wrote on Mon, 11 January 2010 05:09BTW, I had to change it like this :

```
struct StringTable {
 const char *s;

 String ToString() const       { return GetLngString(s); }
 operator const char *() const { return ToString(); }
 operator String() { return ToString(); }
 operator Value() { return ToString(); }
};
```

Otherwise it didn't work here :

```
DropList d;
int i = 0;
d.Add(i, myStringTable[i]);
```

Don't know exactly why he didn't pick automatically the char * --> Value conversion. Even adding the String() operator wasn't enough.

Ciao

Max

Well, I believe it is "one-conversion-operator" rule: C++ never goes through more than single conversion. tt_char -> const char * -> Value is two conversions. (And String does not change much there...).

Mirek

Subject: Re: Helper for internazionalize arrays of literals
Posted by mdelfede on Mon, 11 Jan 2010 10:17:13 GMT
View Forum Message <> Reply to Message

So, why did it work before ? I had just the 'String operator[]' in my previous class, not a Value operator.

Max
