Subject: Re: Controls & classes design questions
Posted by mirek on Sun, 19 Aug 2007 09:51:16 GMT
View Forum Message <> Reply to Message

mrjt wrote on Sun, 19 August 2007 05:22Then I must be misunderstanding something. I would be grateful if you could tell me why the following code works if virtual methods are non-moveable:

```
struct BaseClass
{
 BaseClass() { int1 = 1; }

 int int1;
 virtual int  GetInt()   { return int1; }
 virtual String  GetString()  { return "A String"; }
};

struct DerivedClass : public BaseClass, public Moveable<DerivedClass>
{
 DerivedClass() { int2 = 99; }

 int int2;
 virtual int  GetInt()   { return int2; }
 virtual String  GetString()  { return "This is a derived class"; }
};

GUI_APP_MAIN
{
 Vector<DerivedClass> v;

 v.Add(DerivedClass());
 v.Add(DerivedClass());

 for (int i = 0; i < v.GetCount(); i++)
  PromptOK(Format("Int: %d String: %s", v[i].GetInt(), v[i].GetString()));
}
```

James

Sure it does work - at least with GCC and MSC.

The problem is that C++ standard allows memcpy only for POD types. U++ extends this to moveable types, but that is technically violating C++ standard - such thing is undefined by standard.

Therefore we try to keep such extension as thin as possible. There is really not much practical advantage to having types with vtable moveable and in the same time, it is not that much unlikely that some compiler would implement virtual methods in a way that would not work with U++.