Subject: Re: using Ctrl::Add; required for templates / overloaded virtual functions
Posted by tojocky on Thu, 24 Jun 2010 12:13:08 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 24 June 2010 13:18hi mrjt,

sure this one works, because Mid offers own Add(Ctrl&) *additionally* and so does my app with the Ctrls that either *dont* have own Add() overloads at all or have additional override Add(Ctrl&) properly (virtual or not, no matter), but it does not work with those only having incompatible Add() overloads.

just comment out Mid::Add(Ctrl&) and leave only Add(int) in Mid...boom.

i dont even need to access it at Top level (this would represent my template class, Mid is a CtrlLib Ctrl, Base is Ctrl implementation, thanks for breaking this one down , so Top can go...

```
struct Base {
 int result;
 virtual void Add(Ctrl &) { result = 1; }
};

struct Midd : public Base{
 //virtual void Add(Ctrl &) { result = 2; } //EITHER this, virtual or not, no metter
 using Base::Add; //OR this, saves compilation preserving complete Add overloads pool for
compiler
 void Add(int)   { result = 4; }
 void Nothing()   { result = 10; }
};

GUI_APP_MAIN
{
 Label l;
 Midd md;

 //option 1: << desired >>
 void (Midd::* mfp)(Ctrl &) = &Midd::Add; //problems even if Mid, not only Top
 (md.*mfp)(l);
}
```

i think this is more of a coding guideline, trying to sum up:

option 1:
when overloading (not overriding) a public virtuel base class function, be sure to have a proper override for it as well (maybe just calling base function)
or place a "using" statement to help compiler keep at least the public fnctions pool complete througout the hierarchy.

option 2:
use different function names

Very nice observation!

I made some tests and work perfectly:

```
struct Base1 {
 int result;
 virtual void Add(Ctrl &) { result = 1; }
 virtual void Add(int)   { result = 2; }
 virtual void Add()   { result = 3; }
};

struct Midd : public Base1{
 //virtual void Add(Ctrl &) { result = 2; } //EITHER this, virtual or not, no metter
public:
 virtual void Add(int)   { result = 4; }
 void Nothing()   { result = 10; }
private:
 using Base1::Add; //OR this, saves compilation preserving complete Add overloads pool for
compiler
};

GUI_APP_MAIN
{

 Label l1;
 Midd md1;

 //option 1: << desired >>
 void (Midd::* mfp1)(Ctrl &) = &Midd::Add; //problems even if Mid, not only Top
 (md1.*mfp1)(l1);
 Exclamation(Format("%d", md1.result));

 void (Midd::* mfp12)(int) = &Midd::Add; //problems even if Mid, not only Top
 (md1.*mfp12)(3);
 Exclamation(Format("%d", md1.result));

 void (Midd::* mfp13)() = &Midd::Add; //problems even if Mid, not only Top
 (md1.*mfp13)();
 Exclamation(Format("%d", md1.result));
}
```

and output are:

1

4
3

Thank you!

P.S. Maybe it is a bug of C++ if not compiling without using?
