Subject: using Ctrl::Add; required for templates / overloaded virtual functions
Posted by kohait00 on Wed, 23 Jun 2010 12:14:26 GMT
View Forum Message <> Reply to Message

hi folks,

this one is sort of freak situation in my application. nevertheless i provide it here for discussion. consider the following example.

```
class MyD
 : public ArrayCtrl
 //works with Label (it has no own Add() overloads,
 //does not work with ArrayCtrl, it has own Add() overloads
{
public:
 typedef MyD CLASSNAME;
 virtual ~MyD() {}
};

GUI_APP_MAIN
{
 Label l;
 MyD md;

 //option 1: << desired >>

 void (MyD::* mfp)(Ctrl &) = NULL;
 mfp = (void (MyD::*)(Ctrl &)) &MyD::Add;
 //only works if ArrayCtrl:: class has a using Ctrl::Add
 //otherwise
 //error C2440: 'type cast' : cannot convert
 //from 'overloaded-function'
 //to 'void (__thiscall Upp::Ctrl::* )(Upp::Ctrl &)'
 //None of the functions with this name in scope match the target type

 (md.*mfp)(l);

 //option 2: doenst work anyway, because binding to Ctrl::Add is clear.
 void (Ctrl::* mfp2)(Ctrl &) = NULL;
 mfp2 = (void (Ctrl::*)(Ctrl &)) &Ctrl::Add; //(void (Ctrl::*)(Ctrl &))
 (md.*mfp2)(l); //does *not* call overriden Add(Ctrl&), but the one from Ctrl::

 ttest().Run();
}
```

problem is following:

i need to reference the virtual & overloaded ArrayCtrl::Add(Ctrl &) member function with a member function pointer (in template environment, but this one produces same errors and solution is applicable there as well). i access the Add(Ctrl&) function from topmost derived class MyD and it works fine with all Ctrl's that have no own Add() function *overloads* (not virtual overrides). but i.e. ArrayCtrl which has Add(Value&) and others, prevents this one from compiling.

key line is

```
 mfp = (void (MyD::*)(Ctrl &)) &MyD::Add;
```

for which MSC produces C2440 error.

i managed to solve it by adding a
ArrayCtrl.h:427

```
using Ctrl::Add;
```

which enables the compiler to deduce stuff explicitly, since ArrayCtrl now explicitly provides access to Ctrl::Add(Ctrl&) and the line compiles, both in MSC and GCC.

now i think of it as a general point to take position to.

why not providing using Ctrl::XXX ; for functions, which we know are used often, are even virtual, are public as well, and are beeing overloaded by a deriving class?

(so far i found this to be the case for ArrayCtrl and DropList, sure there are some more)

what is your point?

PS: i could possibly circumvent the problem by specifying an intermediate class like that, exposing Ctrl::Add explicitly, but is it the clean way?

```
class MyArrayCtrl
 : public ArrayCtrl
{
public:
 typedef MyArrayCtrl CLASSNAME;
 virtual ~MyArrayCtrl() {}
 using Ctrl::Add;
};
```

sorry for the long post, problem is not trivial though
attached a test project

File Attachments
1) ttest.rar, downloaded 236 times

Subject: Re: using Ctrl::Add; required for templates / overloaded virtual functions
Posted by mrjt on Wed, 23 Jun 2010 14:38:18 GMT
View Forum Message <> Reply to Message

I don't understand.

Option 2 is correct - ArrayCtrl doesn't overload Add(Ctrl &) (it's not even virtual), so obviously it's being called on the Ctrl base class.

Option 1 doesn't work because ArrayCtrl doesn't overload Add(Ctrl &), so the pointer signature is incorrect.

If you add the following to MyD then both options work correctly:

```
class MyD
 : public ArrayCtrl
{
public:
 typedef MyD CLASSNAME;
 virtual ~MyD() {}

 void    Add(Ctrl& ctrl) { Ctrl::Add(ctrl); Update(); }
};
```

Option 1: Calls MyD::Add

```
void (MyD::* mfp)(Ctrl &) = &MyD::Add;
(md.*mfp)(l);
```

Option2 calls Ctrl::Add

```
void (Ctrl::* mfp2)(Ctrl &) = &Ctrl::Add;
(md.*mfp2)(l);
```

This is also possible, and perhaps most useful:

```
void (MyD::* mfp)(Ctrl &) = &Ctrl::Add;
```

(Tested with latest SVN and MSC8)

Subject: Re: using Ctrl::Add; required for templates / overloaded virtual functions
Posted by kohait00 on Wed, 23 Jun 2010 15:04:58 GMT
View Forum Message <> Reply to Message

hi mrjt, thanks for replying

i think you got me wrong (which is no surprise, the problem is weired).

as pointed out in your post, ArrayCtrl does not implement / override Add(Ctrl&), but an Add(Ctrl&) implementation is accessible through the public interface inherited by Ctrl, but this is implicit for the compiler.

when i do

mfp = (void (MyD::*)(Ctrl &)) &MyD::Add;

i could access the implicit Add(Ctrl&) from Ctrl. but as soon as ArrayCtrl offers own Add() *overloads* (not overrides) the compiler is only looking at the ArrayCtrl Add() variants, moaring the Add(Ctrl&) does not exist among the Add(Value&)..etc.

i need the above syntax to access the *topmost* Add(Ctrl&) implementation due to a template access, so &Ctrl::Add is no option here. (in things like Splitter Add(Ctrl&) is overridden, which i need to use generally)

option 2 was just a quick shot, and *always* executes Ctrl::Add(Ctrl&) which is correct and logical, but not what i need. i need the topmost implementation of Add(Ctrl&).

i hope you got my point here.

thus the "solution", an explicit using Ctrl::Add; statement in ArrayCtrl which explicitly completes the set of overloaded Add functions, the compiler is looking for.

nevertheless, i managed to get around this using the intermediate class like described which is ok for me, i cant exceed to change the code for my purposes

the point was whether adding using statement should be done generally or not when deriving from a class and implementing functions which have same name. which causes compilation errors some times (some rare times though).

Subject: Re: using Ctrl::Add; required for templates / overloaded virtual functions
Posted by mrjt on Thu, 24 Jun 2010 08:51:24 GMT
View Forum Message <> Reply to Message

I see your problem. I think you may have run up against the limit's of the C++ compiler there. Congratulations

Of course the real problem is the manner used for overloading the Add(Ctrl &) method. If a method is going to be overloaded it should be done properly. For instance, this code works exactly as you would like:

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

```
struct Mid : public Base{
 virtual void Add(Ctrl &) { result = 2; }
 void Add(int)   { result = 4; }
};

struct Top : public Mid{
 void Nothing()  { result = 0; }
};


GUI_APP_MAIN
{
 Label l;
 Top md;

 //option 1: << desired >>
 void (Top::* mfp)(Ctrl &) = &Top::Add;
 (md.*mfp)(l);
}
```
The function pointer calls Mid::Add(Ctrl &).

So either the Ctrls that overload Add(Ctrl &) should be changed (surely they can use ChildAdded?) or Add(Ctrl &) should be made virtual.

Subject: Re: using Ctrl::Add; required for templates / overloaded virtual functions
Posted by kohait00 on Thu, 24 Jun 2010 10:18:37 GMT
View Forum Message <> Reply to Message

hi 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

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?

Subject: Re: using Ctrl::Add; required for templates / overloaded virtual functions
Posted by kohait00 on Thu, 24 Jun 2010 12:52:29 GMT
View Forum Message <> Reply to Message

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

who knows to find something similar on the net was quite difficult, there is lots of stuff concerning C2440 error, but quite few related to this one...

the question at the end, what to do with this, which of the 2 options should U++ set as code guideline? which is esier

---

Subject: Re: using Ctrl::Add; required for templates / overloaded virtual functions
Posted by mirek on Tue, 06 Jul 2010 07:26:09 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Thu, 24 June 2010 08:52Quote:
P.S. Maybe it is a bug of C++ if not compiling without using?

who knows to find something similar on the net was quite difficult, there is lots of stuff concerning C2440 error, but quite few related to this one...

the question at the end, what to do with this, which of the 2 options should U++ set as code guideline? which is esier

I would say, use different method name....

In any case, code provided is simply nasty. I would not dare to use anything like that unless in some really dire situation

---

Subject: Re: using Ctrl::Add; required for templates / overloaded virtual functions
Posted by kohait00 on Tue, 06 Jul 2010 08:16:12 GMT
View Forum Message <> Reply to Message

Quote:
use different method

also my opinion, thats esiest and cleanest anyway. so we should carefully inspect the interfaces we override / present, not to overload stuff wrong way.
