
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: doesn't 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 overridden 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 being 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 314 times
