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).