Subject: PROPOSAL: delegate like separation of method and args in Callbacks
Posted by kohait00 on Fri, 16 Jul 2010 06:09:09 GMT
View Forum Message <> Reply to Message

hi folks,

here comes an interesting mod for the Callback layer.

In the current Callback implementation, the arguments (or parameters) to the stored method informations are kept together in a class.

```
template <class OBJECT_, class METHOD_, class T>
struct CallbackMethodActionArgPte : public CallbackActionArg<T> {
 Ptr<OBJECT_>  object;
 METHOD_       method;
 T             arg;

 void    Execute() { if(object) (object->*method)(arg); }

 CallbackMethodActionArgPte(OBJECT_ *object, METHOD_ method, T arg)
 : object(object), method(method), arg(arg)  {}
};
```

this is not veeery handy, if one thinks of Callbacks beeing a 'almost' delegate (not too strictly though, here, our delegates can only be named methods, no anonymous code). one key feature about delegates (in C#) i.e. is that they actually dont care about the class type (or only at the beginning maybe). sth like "i dont care the class type instance as long as the method matches my signature". thus it is possible to reuse a delegate, bind it to a another (maybe even different) class type method (with same signature) and continue. but this is not what its about here

imagine you created a Callback somewhere in a templated function, but dont keep the information about the class type up to upper layer because you dont need it there, the code is too general. but you know the method types and want to change the parameters stored in a Callback. what do? here a proposal that makes cool stuff possible: (example for 1 parameter, others analogue)

Callback.h

```
//new class for separated arguments from class/method types
template <class T>
struct CallbackActionArg : public CallbackAction {
 T             arg;
 CallbackActionArg(T arg) : arg(arg) {}
};

template <class OBJECT_, class METHOD_, class T>
struct CallbackMethodActionArg : public CallbackActionArg<T> {
```

```
 OBJECT_  *object;
 METHOD_   method;
// T         arg;

 void    Execute() { (object->*method)(arg); }

 CallbackMethodActionArg(OBJECT_ *object, METHOD_ method, T arg_)
 : CallbackActionArg<T>(arg_), object(object), method(method) {}
};
```

this makes the following possible

```
 Callback cb4;

 CallbackActionArg<int> * pcba = NULL; //no class information here
 pcba = new CallbackMethodActionArg<MyCallback, void (MyCallback::*)(int), int>(this,
&MyCallback::Action1, 45); //created stronlgy typed and assigned a method, delegate like :) could
be created later with a DELEGATE() helper or sth.

 cb4 = Callback(pcba);
 cb4();
 pcba->arg = 65; //here we can change the argument, but dont need to know which class type /
instance it was bound to
 cb4();
```

i tried / tested it. it works fine. please analyze the aproach, give feedback, i think it is pretty usefull. a usecase was described above. i can provide a fully edited proposal Callback.h

---