Subject: Hopefully final C++11 related disruptive change: Callback now deprecated
Posted by mirek on Tue, 04 Oct 2016 08:28:31 GMT
View Forum Message <> Reply to Message

After numerous changes to better use C++11 features in U++, I was still a little bit uneasy about Function/Callback duality that was originally forced by overloading issues. Now after further investigation it looks like it is actually possible to deprecate Callback completely and replace everywhere with Function, with only slight impact on backward compatibility.

So the new model says there is single Upp::Function, similar to std::function (with two very important distinctions - empty Function call is NOP and it can be combined using <<) and convenience typedefs of Function to Event and Gate. Callback is replaced by Event<> (which is equivalent of Function<void ()>, Callback1<Foo> by Event<Foo> (Function<void (Foo)>) etc, Gate is similar to original Gate (e.g. Gate<int> is now Function<bool (int)>). Callback and related things (like THISBACK) are kept for BW compatibility.

In my 2 major project, conversion to this new model required no changes in one of them and fixing 5 lines in another. Things to fix are reported as compile time bugs. There are basically two situations to fix:

```
Ctrl x;
x <<= x.WhenSomething = THISBACK(Foo);
```

has to be rewritten as

```
x.WhenAction = x.WhenSomething = THISBACK(Foo);
```

(note that x <<= THISBACK(Foo) is OK).

```
Gate foo;
```

has to be rewritten as

```
Gate<> foo;
```

(however this second issue is very rare).

Hopefully this was the last thing to change in U++ before "C++11 embracing release"...

Mirek

Subject: Re: Hopefully final C++11 related disruptive change: Callback now deprecated
Posted by koldo on Wed, 05 Oct 2016 08:45:27 GMT
View Forum Message <> Reply to Message

:) Thank you.

Subject: Re: Hopefully final C++11 related disruptive change: Callback now deprecated
Posted by Mindtraveller on Mon, 10 Oct 2016 12:33:43 GMT
View Forum Message <> Reply to Message

Thank you. Is it possible to say at least for MSC15/CLANG3.4 how this "native" callback impacts effectivenes? Do we really achieve better timing replacing U++ callbacks?

Subject: Re: Hopefully final C++11 related disruptive change: Callback now deprecated
Posted by mirek on Thu, 13 Oct 2016 06:57:19 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Mon, 10 October 2016 14:33Thank you. Is it possible to say at least for MSC15/CLANG3.4 how this "native" callback impacts effectivenes? Do we really achieve better timing replacing U++ callbacks?

Well, it definitely was not a primary concern (I am OK if new are 'about the same speed'), however it is an interesting question. So I have put it to the test:

MSC15x64 Release MT

Callback, U++ classic:

```
#include <Core/Core.h>

using namespace Upp;

struct Test {
 int sum;

 Vector<Callback> cb;

 void Method1() {
  sum++;
 }

 void Method2(int d) {
```

```
  sum += d;
 }

 typedef Test CLASSNAME;

 Test() {
  const int N = 1000000;
  for(int j = 0; j < 50; j++) {
   cb.Clear();
   for(int i = 0; i < N; i++) {
    RTIMING("Adding Method1");
    cb.Add(THISBACK(Method1));
   }
   sum = 0;
   for(int i = 0; i < cb.GetCount(); i++) {
    RTIMING("Calling Method1");
    cb[i]();
   }
   RDUMP(sum);
   cb.Clear();
   for(int i = 0; i < N; i++) {
    RTIMING("Adding Method2");
    cb.Add(THISBACK1(Method2, i));
   }
   sum = 0;
   for(int i = 0; i < cb.GetCount(); i++) {
    RTIMING("Calling Method2");
    cb[i]();
   }
   RDUMP(sum);
  }
 }
};

CONSOLE_APP_MAIN
{
 Test();
}
```

Lambdas and Event<>, 'new U++':

```
#include <Core/Core.h>

using namespace Upp;
```

```
struct Test {
 int sum;

 Vector<Event<>> cb;

 void Method1() {
  sum++;
 }

 void Method2(int d) {
  sum += d;
 }

 typedef Test CLASSNAME;

 Test() {
  const int N = 1000000;
  for(int j = 0; j < 50; j++) {
   cb.Clear();
   for(int i = 0; i < N; i++) {
    RTIMING("Adding Method1");
    cb.Add([=] { Method1(); });
   }
   sum = 0;
   for(int i = 0; i < cb.GetCount(); i++) {
    RTIMING("Calling Method1");
    cb[i]();
   }
   RDUMP(sum);
   cb.Clear();
   for(int i = 0; i < N; i++) {
    RTIMING("Adding Method2");
    cb.Add([=] { Method2(i); });
   }
   sum = 0;
   for(int i = 0; i < cb.GetCount(); i++) {
    RTIMING("Calling Method2");
    cb[i]();
   }
   RDUMP(sum);
  }
 }
};

CONSOLE_APP_MAIN
{
 Test();
}
```

classic Callbacks:

TIMING Calling Method2: 221.43 ms - 4.43 ns ( 1.61 s / 50000000 ), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 50000000
TIMING Adding Method2 : 1.78 s - 35.55 ns ( 3.17 s / 50000000 ), min: 0.00 ns, max: 4.00 ms, nesting: 1 - 50000000
TIMING Calling Method1: 157.43 ms - 3.15 ns ( 1.55 s / 50000000 ), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 50000000
TIMING Adding Method1 : 1.67 s - 33.31 ns ( 3.06 s / 50000000 ), min: 0.00 ns, max: 3.00 ms, nesting: 1 - 50000000

new Event<>s:

TIMING Calling Method2: 117.74 ms - 2.35 ns ( 1.45 s / 50000000 ), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 50000000
TIMING Adding Method2 : 1.09 s - 21.87 ns ( 2.43 s / 50000000 ), min: 0.00 ns, max: 3.00 ms, nesting: 1 - 50000000
TIMING Calling Method1: 85.74 ms - 1.71 ns ( 1.42 s / 50000000 ), min: 0.00 ns, max: 1.00 ms, nesting: 1 - 50000000
TIMING Adding Method1 : 1.13 s - 22.55 ns ( 2.46 s / 50000000 ), min: 0.00 ns, max: 3.00 ms, nesting: 1 - 50000000

So interestingly, yes, it is a little bit faster. My guess there it is partly because of && (Adding part), partly because new C++11 implementation perhaps allows more inlining (Calling part).

Anyway, the main value for me is that about 4000 lines of code in classic Core were replaced by 100 lines in "C++11 Core" and 50+ of various types, macros and functions are now replaced by just 3 types and single macro. (Of course, we still these need to maintain these 50+ for BW compatibility and have now about 300 lines to achieve that. But the new system is overall much simpler).