LEWG, SG14: D0040R0   
22-9-2015   
Scott Wardle, Roberto Parolin  
[swardle@gmail.com](mailto:swardle@gmail.com), [robertoparolin@gmail.com](mailto:robertoparolin@gmail.com)

**Comparing Virtual Methods**

**I. Summary**

It would be useful to compare if an instance of a class has a particular function in its virtual table.

**II. Motivation**

Traditional entity systems in c++ are used often in games. An example of one of these entity systems would be if you have a list of entity base classes some of which are overridden and not known at compile time and are data driven. You would then update these entities by looping over a vector of pointers to entity base class and calling each entity update function.

We could separate each entities based on the update function in the virtual table moving these overridden objects out of the fast path and have one function that updated many non-overridden entities. Then update each overridden entity slowly afterwards.

## III. Additional work

Sorting by using the virtual table it’s self not function as typeid, it might be useful in some cases (remember we don’t use rtti). However this is a separate problem. But maybe the destructor function could use used as a typeid for the class as there can be only one of these even with DLLs unless you inline the destructor.

**III. Discussion**

Roughly I want to do things like the following C# code can.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace cs\_func\_sort

{

class A

{

virtual public void Foo()

{

Console.WriteLine("A::Foo");

}

virtual public void Name()

{

Console.Write("Class A ");

}

}

class B : A

{

override public void Foo()

{

Console.WriteLine("B::Foo");

}

override public void Name()

{

Console.Write("Class B ");

}

}

class C : B

{

override public void Name()

{

Console.Write("Class C ");

}

}

delegate void fptrDelegate();

class Program

{

static void Main(string[] args)

{

List<A> list = new List<A>();

list.Add(new A());

list.Add(new C());

list.Add(new B());

list.Add(new A());

list.Add(new C());

Console.WriteLine("Print the list unsorted");

foreach (A cur in list)

{

cur.Name();

cur.Foo();

}

Console.WriteLine("Print the list sorted");

list.Sort(delegate (A itemA, A itemB)

{

fptrDelegate afoo = itemA.Foo;

fptrDelegate bfoo = itemB.Foo;

return afoo.Method.GetHashCode() - bfoo.Method.GetHashCode();

});

foreach (A cur in list)

{

cur.Name();

cur.Foo();

}

Console.WriteLine("Find all of the version with B");

// Can't think of a good way to get rid of creating this object in C# in C++

// in C++ it would be better if we didn’t have to create a instance of the class

// to make a pointer to function like the example below in yellow.

A testA = new B();

fptrDelegate testfoo = testA.Foo;

foreach (A cur in list)

{

fptrDelegate curfoo = cur.Foo;

if( testfoo.Method.Equals(curfoo.Method) )

{

// call B::Foo AND C::Foo!!

cur.Name();

cur.Foo();

}

}

}

}

}

**IV. Proposed Text**