WG21-P0251R0
2016-02-11

Unified Call Syntax Wording

Bjarne Stroustrup [bs@ms.com]

Herb Sutter [hsutter@microsoft.com]

Abstract

This is the proposed wording for a unified call syntax based on the idea that **f(x,y)** can invoke a member function, **x.f(y)**, if there are no **f(x,y)**. The inverse transformation, from **x.f(y)** to **f(x,y)** is not proposed. This wording is the result of EWG discussions based on

- N4165: Herb Sutter: *Unified Call Syntax*
- N4174: Bjarne Stroustrup: *Call syntax: x.f(y) vs. f(x,y)*
- N4474: Bjarne Stroustrup and Herb Sutter: *Unified Call Syntax: x.f(y) and f(x,y)*
- P0131R0: Bjarne Stroustrup: *Unified call syntax concerns*.

As agreed in EWG at the Kona meeting, the intent of the wording is:

- For **f(x,y)**, see if **f(x,y)** is valid and if so do that call; otherwise try **x.f(y)**.

That's it. The code examples are extensive primarily to show that there are no intended changes beyond that.

Wording

In 5.2.2 Function call [expr.call], add a new paragraph[12]:

If a call of a function designated by an *unqualified-id* does not have a valid resolution and its first argument denotes a class object, invoke the member function of that name for the class of the first argument as if called using the dot operator with the remaining arguments (if any) as the arguments to the member function.

[[Example:

```
namespace N {
        struct Y { void k(); };
        void h(Y);
};

struct X { void k(); };

void h(X);
```

```
struct S {
    void f(X x)
    {
        g(x);    // invoke S::g(X): (*this).g(x)
        h(x);   // invoke ::h(X)
        k(x);    // invoke X::k(): x.k()
        ::k(x);  // error: there is no global k
     }

     void g(X);

     void f2(N::Y y)
     {
           g(y);   // invoke S::g(N::Y): (*this).g(y)
           h(y);   // invoke S::h(N::Y); N.h() found by ADL
           k(y);   // invoke N::Y::k(): y.k()
           N::k(y); // error: there is no k in namespace N
     }

     void g(N::y);
};

   ]]

   [[ Example:

   struct A {
                virtual void h();
                static void k(A&);
   };

   struct B : A { };

   void f(B& b)
   {
                b.h();         // OK (as ever)
                h(b);          // OK: b.h();
                b.k();         // error (as ever)
                k(b);          // error: Try b.k(), but k() requires an argument
                A::h(b);     // error: Qualified name, so no transformation to b.h()
                A::k(b);     // OK (as ever)
   }

]]
```