Document Number: WG21/N2386=07-0246
Revision of: WG21/N2312=07-0172
Date: 2007-08-06
Reply to: Michael Spertus
mike_spertus@symantec.com

Namespace Regions

The problem

Although sometimes prone to overuse, the *using-directive* and *using-declaration* constructs are frequently important and valuable. From adding simple convenience, such as replacing the awkward

```
std::cout << std::string("Hello ") + "world" << std::endl;
```

with the more natural

```
using namespace std;
...
cout << string("Hello ") + "world" << endl;
```

to simplifying organization-wide conventions, *using-directive*s and *using-declaration*s are a widely used part of the language.

Unfortunately, *using-directive*s and *using-declaration*s are generally are not suitable for use in header files because they risk polluting the namespace of the source files that include the header files. What makes this especially bad is that the prevalence of templates in modern C++ programming means that much if not most code is in header files. Indeed, almost the entire Boost library consists of header files. Of course, it is possible to include such declarations on a method by method basis. However, the common best practice of keeping methods short makes this technique of limited value.

Even worse, normal use of extensible literals requires using-directives or using-declations as the use of explicit scope qualifiers with them is not allowed. For example, the namespace containing the `_miles` literal must be implicitly search in expressions like `26_miles`.[1] As a result, it is virtually impossible to use extensible literals in headers (or template implementation files) without leaking namespaces.

It is not an overstatement to say that the result of this has been that the *using-directive* and *using-declation* constructs have proved largely useless for me. As a result, most of my code (at least that in header files) looks like the awkward “Hello world” example above instead of the more natural version that uses namespace `std`.

Namespace regions

In order to make using-directives and using-declarations better suited for use in header

[1] Although allowing a syntax like `units::26_miles` might be a worthwhile enhancement to extensible literals.

files and template implementation files, we propose to allow statements like the following:

```
namespace A {
  extern using namespace B; // Strong using
  using namespace C; // Still leaks outside
  private using namespace std; // Not visible outside this region
  class B {
  void foo() {
    cout << string("Hello ") + "world!" << endl; // OK
  }
}
Using namespace A;
void f() {
  cout << "foo"; // Ill-formed. Leakage prevented.
}
```

The exact same reasoning applies to using declarations

```
namespace A {
  int i;
  int j;
}
namespace B {
  using A::i;
  private using A::j; // Don't leak this
  …
}

using namespace B;
int
main()
{
  i = 1; // OK
  j = 1; // Ill-formed. Leakage prevented
  return 0;
}
```

This allows much greater control over the region in which using-directives and declarations are in effect. Of course, this does not help headers in the global namespace, but we do not regard that as a bad thing, as it would be just one more reason to create namespaces for ones headers.

Technically, a `private using` directive or declaration says that the privately-used namespace is not added to the list of nominated namespaces when the containing namespace is nominated.

No special treatment is required for argument dependent lookup, which essentially adds nominated namespaces to lookups, so privately-used namespaces are (properly) not leaked in argument dependendent lookup.

Proposed wording

In section 7.3.3 [namespace.udecl] change the BNF as follows:

> *using-declaration*:
> private$_{opt}$ using *typename$_{opt}$* ::$_{opt}$ *nested-name-specifier unqualified-id* ;
> private$_{opt}$ using :: *unqualified-id* ;

In section 7.3.4 [namespace.udir], change the BNF as follows:

> *Using-directive:*
> private$_{opt}$ using namespace ::$_{opt}$ *nested-name-specifier$_{opt}$ namespace-name* ;

Change the start of paragraph 4 of 7.3.4 as follows:

> The *using-directive* is transitive: if a scope contains a non-private *using-directive* that nominates a second namespace that itself contains *using-directive*s, the effect is as if the *using-directive*s from the second namespace also appeared in the first. [Example:

```
namespace M {
  int i;
}
namespace N {
  int i;
  using namespace M;
}
namespace P {
  private using namespace M;
  void k() { i = 7; }
}
void f()
{
  using namespace N;
  i = 7; // error: both M::i and N::i are visible
}
void g()
{
  using namespace P;
  i = 7; // error; P only using M::i internally
}
```

Add a note to 3.4.2

In paragraph 2 of 3.4.3.2, change the phrase "transitive closure of all namespaces nominated by non-private *using-directives* in X.."

In the example in 3.4.3.2, add the class

```
class W {
public:
      private using namespace A;
}
```

In the function h() in 3.4.3.2, add the line

```
W::g(3); // Ill-formed. A is only used internally by W
```

There will be additional wording