X3J16/99-0010
WG21/N1187
February 21st, 1999
John H. Spicer, Edison Design Group
jhs@edg.com

Class Name Injection Issues

There are several areas in which the specification of class name injection is incomplete, or in which the specification renders previously well-formed programs ill-formed. This document suggests clarifications to make the specification more complete and to make more explicit any incompatibilities that may exist.

1. Clarification that base class names can be made inaccessible

With class name injection, when a base class name is used in a derived class, the name found is the injected name in the base class, not the name of the class in the scope containing the base class. Consequently, if the base class name is not accessible (e.g., because it is in a private base class) the base class name cannot be used unless a qualified name is used to name the class in the class or namespace of which it is a member.

Without class name injection the following example is valid. With class name injection, `A` is inaccessible in class `C`.

```
class A {};
class B : private A {};
class C : public B {
      A* p;
};
```

At the least, the standard should be more explicit that this is, in fact, ill-formed.

2. Name injection and templates

There is some controversy about whether class name injection applies to class templates. If it does apply, what is injected? Is a class name injected or is the thing that is injected actually a template?

Clause 9, paragraph 2 says “The *class-name* is also inserted into the scope of the class itself.” In general, clause 9 applies to both classes and class templates, so I would take this to mean that class name injection does indeed apply to class templates. One problem with this is that clause 9 uses the syntactic term *class-name*, which I would take to imply that the inserted name is always a class. This is clearly unacceptable for class templates as it makes the template itself unusable from within the template. For example:

```
template <class T> struct A {
  A<T*>     ptr;  // Invalid - A refers to class
};
```

Clearly the injected name must be usable as both a class and a class template. This kind of magic already exists in the standard. In 14.6.1 [temp.local], it says “Within the scope of a class template, when the name of the template is neither qualified nor followed by <, it is equivalent to the name of the template followed by the set of *template-parameters* enclosed in <>.”

The proposal here is that we clarify that name injection does indeed apply to class templates, and that it is the injected name that has the special property of being usable as both a class and a template name (as described in 14.6.1). This would eliminate the need for special wording regarding the qualification of the name, but would achieve the same result. This would also make this “special” name available to a derived class of a class template – something which is necessary if the benefits of class name injection are to be made uniformly available for class templates too.

```
template <class T> struct Base {
  Base *p;
  Base<T*> *p2;
  ::Base *p3;  // Error: only injected name usable as class
};

template <class T> struct Derived : public Base<T> {
  Base *p; // Now okay
  Base<T*> *p2;  // Still okay
  Derived::Base *p3;  // Now okay
};
```

Note that by giving the special attribute of being usable as both a class and a template to the injected name it is now clear where this attribute can and cannot be used.

3. Injected names and elaborated type specifiers

There have been some references to the injected name as a “public typedef”, although the standard does not refer to the injected name as such. The standard leaves open to debate whether or not the injected name can be used in an elaborated type specifier. It is proposed that the name injected into the class scope is the same kind of name that is put in the containing scope. Consequently, the injected name can be referenced from an elaborated type specifier. If the name was instead injected as a typedef, usage such as the following would no longer be permitted:

```
class A {
      class B {
            friend class A;  // Only okay of A is not a typedef
      };
};
```

4. Constructor names

5.1p7 says that *class-name*::*class-name* refers to the constructor when both *class-name*s refer to the same class. This is fine, as far as it goes, but I think the description needs to be expanded to cover cases like “`A<T>::A<T2>`”. If class A has a template constructor there is an ambiguity concerning whether the template argument list is being specified for the injected class name or for the constructor.

I propose a clarification that restates this rule as a component of name lookup. Specifically, if when doing a qualified lookup in a given class you look up a name that is the same as the name of the class, the entity found is the constructor and not the injected class name. In all other cases, the name found is the injected class name.

```
class B {};
class A : public B {
  A::B      ab;  // B is the inherited injected B
  A::A      aa;  // Error: A::A is constructor
};
```

Note that you never need to say something like “A::A” to name a type. It can’t be used to access a hidden type because in order to get to the second “A” you already had to have been able to name the first “A”.