On resolving the type vs member conflict in C++, revisited

devblogs.microsoft.com/oldnewthing/20190829-00

August 29, 2019

Raymond Chen

Some time ago, I wrote about the type vs. member conflict, known informally as *The Color Color problem*. I may have started in the deep end of the pool, so here's a little bit of getting-up-to-speed so that article might make more sense.

```
namespace Windows::UI::Xaml
{
  enum class Visibility { Collapsed, Visible };

  struct Style { /* ... */ };

  namespace Controls
  {
    struct UIElement
    {
    public:
      /* ... */

      // returns current visibility
      Windows::UI::Xaml::Visibility Visibility();

      // change visibility
      void Visibility(Windows::UI::Xaml::Visibility value);

      // returns current style
      Windows::UI::Xaml::Style Style();

      // change style
      void Style(Windows::UI::Xaml::Style value);
    };
  }
}
```

The fundamental problem here is that there is a name conflict between the type `Style` and the method `Style`. There is also a name conflict between the type `Visibility` and the method `Visibility`.

When used from within the `UIElement` class, or any class derived from it, the names `Style` and `Visibility` refer to the methods `UIElement::Style` and `UIElement::Visibility`, rather than to the types.

In language-speak, these are *unqualified names*, meaning that the name is just hanging out by itself without any clues as to where to find it. You're asking the compiler to figure out what you're referring to. And if you are using the name in the context of a class, the members of the class have priority over names outside the class.

In other words, the method names `Style` and `Visibility` cause the type names to become hidden. (Another name for this is *shadowing*.)

Some people tut-tut at this problem and declared, "You silly Windows people, using Pascal case for your names. If you had followed the language standard naming pattern, this problem wouldn't even exist!"

The C++ language standard naming convention has the same problem. In the C++ standard library, type names are `snake_case`, and method names are also `snake_case`. The method

```
mutex_type* std::shared_lock::mutex() const noexcept;
```

has a name `mutex` that shadows the type name `std::mutex`. If you derive from `std::shared_lock` and try to use a `mutex`, you're going to get the method, not the type.

Even outside of Windows, type hiding is not a purely theoretical problem: The `sys/stat.h` header file defines a structure called `struct stat`, as well as a function `stat()`. As a result, you are forced to say `struct stat` in order to get the structure. Writing `stat` by itself gets you the function.

So keep your eye open for the `Color Color` problem, even if your use case doesn't involve `Color`.

Raymond Chen

Follow