C++ namespace parlor tricks

devblogs.microsoft.com/oldnewthing/20180516-00

May 16, 2018

Raymond Chen

These tricks may be obvious, but at least I'm going to write them down.

It is common to import an entire namespace into the global namespace. I'm not saying it's a good idea, but it is common. The most notorious example is to put

```
using namespace std;
```

to import the `std` namespace into the global namespace.

When working with the Windows Runtime, you often have rather deep namespaces. For example, we saw some time ago that we were operating in the `Windows::System::Profile::SystemManufacturers` namespace. This is quite a mouthful, and it is common to put a

```
using namespace Windows::System::Profile::SystemManufacturers;
```

in your program just to save yourself the hassle of typing it all out.

However, things get complicated if you create name collisions.

For example, if you are using WRL, you will be working with the `ABI::Windows::System::Profile::SystemManufacturers` namespace, but if you do a

```
using namespace ABI::Windows::System::Profile::SystemManufacturers;
```

you now have a problem because you have imported the name `SmbiosInformation` twice:

```
using namespace Windows::System::Profile::SystemManufacturers;
using namespace ABI::Windows::System::Profile::SystemManufacturers;
```

After these two declarations, the name `SystemInformation` is now ambiguous. It could refer to `Windows::System::Profile::SystemManufacturers::SystemInformation`, via the first `using` declaration, or it oculd refer to `ABI::Windows::System::Profile::SystemManufacturers::SystemInformation, via the second using declaration.`

I've worked around this by using namespace aliases:

```
namespace wspsm = Windows::System::Profile::SystemManufacturers;
namespace awspsm = ABI::Windows::System::Profile::SystemManufacturers;
```

This lets me use `wspsm::SmbiosInformation` and `awspsm::SmbiosInformation` to refer to the C++/CX or ABI versions, respectively.

However, this gets clunky once you have multiple namespaces you want to access:

```
namespace wspsm = Windows::System::Profile::SystemManufacturers;
namespace awspsm = ABI::Windows::System::Profile::SystemManufacturers;
namespace wwspsm = winrt::Windows::System::Profile::SystemManufacturers;

namespace wuvm = Windows::UI::ViewManagement;
namespace awuvm = ABI::Windows::UI::ViewManagement;
namespace wwuvm = winrt::Windows::UI::ViewManagement;

namespace wsc = Windows::Security::Cryptography;
namespace awsc = ABI::Windows::Security::Cryptography;
namespace wwsc = winrt::Windows::Security::Cryptography;
```

because you have to juggle all these aliases. </`P>

But there's a more attractive solution: Move names around by importing them into another namespace. (The name for this technique is "namespace composition", covered in sections 14.4.3 and 14.4.4 of *The C++ Programming Language*.)

```
namespace ABI
{
  using Windows::System::Profile::SystemManufacturers;
  using Windows::UI::ViewManagement;
  using Windows::Security::Cryptography;
}

namespace cx
{
  using Windows::System::Profile::SystemManufacturers;
  using Windows::UI::ViewManagement;
  using Windows::Security::Cryptography;
}

namespace winrt
{
  using Windows::System::Profile::SystemManufacturers;
  using Windows::UI::ViewManagement;
  using Windows::Security::Cryptography;
}
```

The first block of `using` declarations imports the contents of the `ABI::Windows::System::Profile::SystemManufacturers`, `ABI::Windows::UI::ViewManagement`, and `ABI::Windows::Security::Cryptography` namespaces into the

`ABI` namespace.

Similarly for the other two blocks.

The upshot of this is that you can now do this

Old and busted	
New hotness	**To get**
`awspsm::SmbiosInformation`	`ABI::Windows::System::Profile::SystemManufacturers::SmbiosInformation`
`ABI::SmbiosInformation`	
`wspsm::SmbiosInformation`	`Windows::System::Profile::SystemManufacturers::SmbiosInformation`
`cx::SmbiosInformation`	
`wwspsm::SmbiosInformation`	`winrt::Windows::System::Profile::SystemManufacturers::SmbiosInformation`
`winrt::SmbiosInformation`	
`awuvm::ApplicationView`	`ABI::Windows::UI::ViewManagement::ApplicationView`
`ABI::ApplicationView`	
`wuvm::ApplicationView`	`Windows::UI::ViewManagement::ApplicationView`
`cx::ApplicationView`	
`wwuvm::ApplicationView`	`winrt::Windows::UI::ViewManagement::ApplicationView`
`winrt::ApplicationView`	
`awsc::Cryptographic-Buffer`	`ABI::Windows::Security::Cryptography::CryptographicBuffer`
`ABI::Cryptographic-Buffer`	
`wsc::Cryptographic-Buffer`	`Windows::Security::Cryptography::CryptographicBuffer`
`cx::Cryptographic-Buffer`	
`wwsc::Cryptographic-Buffer`	`winrt::Windows::Security::Cryptography::CryptographicBuffer`
`winrt::Cryptographic-Buffer`	

In particular, this trick works with Windows Runtime classes because as a general rule, Windows Runtime type names are unique across all Windows Runtime namespaces, so you won't inadvertently introduce a name collision by `using` a bunch of Windows Runtime namespaces together.

The general rule makes Windows Runtime types easier to search for (both on the Web and in your code) because you will have fewer false positives.

Bonus chatter: The exception to the general rule is DirectX. Windows Runtime naming conventions permit the same name to be used in different versions of DirectX. This isn't a problem because in practice, each application picks one version of DirectX and sticks with it; applications don't try to mix-and-match different versions of DirectX.

Bonus bonus chatter: The above rule is on the books, but has yet to be exercised. As of this writing, the only version of DirectX in the Windows Runtime is DirectX11.

Raymond Chen

Follow