In the conversion to 64-bit Windows, why were some parameters not upgraded to SIZE_T?

devblogs.microsoft.com/oldnewthing/20121029-00

October 29, 2012

Raymond Chen

James wonders why many functions kept `DWORD` for parameter lengths instead of upgrading to `SIZE_T` or `DWORD_PTR`.

When updating the interfaces for 64-bit Windows, there were a few guiding principles. Here are two of them.

- Don't change an interface unless you really need to.
- Do you really need to?

Changing an interface causes all sorts of problems when porting. For example, if you change the parameters to a COM interface, then you introduce a breaking change in everybody who implements it. Consider this hypothetical interface:

```
// namedobject.idl
interface INamedObject : IUnknown
{
    HRESULT GetName([out, string, sizeof(cchBuf)] LPWSTR pszBuf,
                    [in] DWORD cchBuf);
};
```

And here's a hypothetical implementation:

```
// contoso.cpp
class CContosoBasicNamedObject : public INamedObject
{
    ...
    HRESULT GetName(LPWSTR pszBuf, DWORD cchBuf)
    {
        return StringCchPrintfW(pszBuf, cchBuf, L"Contoso");
    }
    ...
};
```

Okay, now it's time to 64-bit-ize this puppy. So you do the natural thing: Grow the `DWORD` parameter to `DWORD_PTR`. Since `DWORD_PTR` maps to `DWORD` on 32-bit systems, this is a backward-compatible change.

```
// namedobject.idl
interface INamedObject : IUnknown
{
    HRESULT GetName([out, string, sizeof(cchBuf)] LPWSTR pszBuf,
                    [in] DWORD_PTR cchBuf);
};
```

Then you recompile the entire operating system and find that the compiler complains, "Cannot instantiate abstract class: CContosoBasicNamedObject." Oh, right, that's because the `INamedObject::GetName` method in the implementation no longer matches the method in the base class, so the method in the base class is not overridden. Fortunately, you have access to the source code for `contoso.cpp`, and you can apply the appropriate fix:

```
// contoso.cpp
class CBasicNamedObject : public INamedObject
{
    ...
    HRESULT GetName(LPWSTR pszBuf, DWORD_PTR cchBuf)
    {
        return StringCchPrintfW(pszBuf, cchBuf, L"Basic");
    }
    ...
};
```

Yay, everything works again. A breaking change led to a compiler error, which led you to the fix. The only consequence (so far) is that the number of "things in code being ported from 32-bit Windows to 64-bit Windows needs to watch out for" has been incremented by one. Of course, too much of this incrementing, and the list of things becomes so long that developers are going to throw up their hands and say "Porting is too much work, screw it." Don't forget, the number of breaking API changes in the conversion from 16-bit to 32-bit Windows was only 117.

You think you fixed the problem, but you didn't. Because there's another class elsewhere in the Contoso project.

```
class CSecureNamedObject : public CBasicNamedObject
{
    ...
    HRESULT GetName(LPWSTR pszBuf, DWORD cchBuf)
    {
        if (IsAccessAllowed())
        {
            return CBasicNamedObject::GetName(pszBuf, cchBuf);
        }
        else
        {
            return E_ACCESSDENIED:
        }
    }
}
```

The compiler did not raise an error on `CSecureNamedObject` because that class is not abstract. The `INamedObject::GetName` method from the `INamedObject` interface is implemented by `CBasicNamedObject`. All abstract methods have been implemented, so no "instantiating abstract class" error.

On the other hand, the `CSecureNamedObject` method *wanted* to override the base method, but since its parameter list didn't match, it ended up creating a separate method rather than an override. (The `override` pseudo-keyword not yet having been standardized.) As a result, when somebody calls the `INamedObject::GetName` method on your `CSecureNamed-Object`, they don't get the one with the security check, but rather the one from `CBasic-NamedObject`. Result: Security check bypassed.

These are the worst types of breaking changes: The ones where the compiler doesn't tell you that something is wrong. Your code compiles, it even basically runs, but it doesn't run *correctly*. Now, sure, the example I gave would have been uncovered in security testing, but I chose that just for drama. Go ahead and substitute something much more subtle. Like say, invalidating the entire desktop when you pass `NULL` to `InvalidateRect`.

Okay, so let's look back at those principles. Do we really need to change this interface? The only case where expanding to `SIZE_T` would make a difference is if an object had a name longer than 2 billion characters. Is that a realistic end-user scenario? Not really. Therefore, don't change it.

Remember, you want to make it *easier* for people to port their program to 64-bit Windows, not harder. The goal is *make customers happy*, not *create the world's most architecturally pure operating system*. And customers aren't happy when the operating system can't run their programs (because every time the vendor try to port it, they keep stumbling over random subtle behavior changes that break their program).

Raymond Chen

Follow