How do I convert an HRESULT to a Win32 error code?

devblogs.microsoft.com/oldnewthing/20061103-07

November 3, 2006

Raymond Chen

Everybody knows that you can use the `HRESULT_FROM_WIN32` macro to convert a Win32 error code to an `HRESULT`, but how do you do the reverse?

Let’s look at the definition of `HRESULT_FROM_WIN32`:

```
#define HRESULT_FROM_WIN32(x) \
  ((HRESULT)(x) <= 0 ? ((HRESULT)(x)) \
: ((HRESULT) (((x) & 0x0000FFFF) | (FACILITY_WIN32 << 16) | 0x80000000)))
```

If the value is less than or equal to zero, then the macro returns the value unchanged. Otherwise, it takes the lower sixteen bits and combines them with `FACILITY_WIN32` and `SEVERITY_ERROR`.

How do you reverse this process? How do you write the function `WIN32_FROM_HRESULT`?

It’s impossible to write that function since the mapping provided by the `HRESULT_FROM_WIN32` function is not one-to-one. I leave as an execise to draw the set-to-set mapping diagram from `DWORD` to `HRESULT`. (Original diagram removed since people hate VML so much, and I can’t use SVG since it requies XHTML.) If you do it correctly, you’ll have a single line which maps 0 to `S_OK`, and a series of blocks that map blocks of 65536 error codes into the same `HRESULT` space.

<!– Let's draw a diagram that shows how the `HRESULT_FROM_WIN32` function works:

Win32 HRESULT

The little sliver at the top is the mapping of zero to zero. The big white box at the bottom is the mapping of all negative numbers to corresponding negative numbers. And the rainbow represents the mapping of all the positive values, mod 65536, into the range 0x80070000 through 0x8007FFFF.

–>

Notice that the values in the range 1 through 0x7FFFFFFF are impossible results from the `HRESULT_FROM_WIN32` macro. Furthermore, values in the range 0x80070000 through 0x8007FFFF could have come from quite a few original Win32 codes; you can't pick just one.

But let's try to write the reverse function anyway:

```
BOOL WIN32_FROM_HRESULT(HRESULT hr, OUT DWORD *pdwWin32)
{
 if ((hr & 0xFFFF0000) == MAKE_HRESULT(SEVERITY_ERROR, FACILITY_WIN32)) {
  // Could have come from many values, but we choose this one
  *pdwWin32 = HRESULT_CODE(hr);
  return TRUE;
 }
 if (hr == S_OK) {
  *pdwWin32 = HRESULT_CODE(hr);
  return TRUE;
 }
 // otherwise, we got an impossible value
 return FALSE;
}
```

Of course, we could have been petulant and just written

```
BOOL WIN32_FROM_HRESULT_alternate(HRESULT hr, OUT DWORD *pdwWin32)
{
 if (hr < 0) {
  *pdwWin32 = (DWORD)hr;
  return TRUE;
 }
 // otherwise, we got an impossible value
 return FALSE;
}
```

because the `HRESULT_FROM_WIN32` macro is idempotent: `HRESULT_FROM_WIN32(HRESULT_FROM_WIN32(x)) == HRESULT_FROM_WIN32(x)`. Therefore you would be technically correct if you declared that the "inverse" function was trivial. But in practice, people want to try to get "x" back out, so that's what we give you.

Now that you understand how the `HRESULT_FROM_WIN32` macro works, you can answer this question, based on an actual customer question:

> Sometimes, when I import data from a scanner, I get the error "The directory cannot be removed." What does this mean?

You will have to use some psychic powers, but I think you're up to it.

One unfortunate aspect of both `HRESULT`s and Win32 error codes is that there is no single header file that contains all the errors. This is understandable from a logistical point of view: Multiple teams need to make up new error codes for their components, but the `winerror.h` file is maintained by the kernel team. If `winerror.h` were selected to be the master repository for all error codes, it means that any team that wanted to add a new error code or change an existing one would have to pester the kernel team to make the change for them. Things get even more complicated if those teams have their own SDK. For example, suppose both the DirectX and Windows Media teams wanted to include the new `winerror.h` in their corresponding SDKs. If you install the SDKs in the wrong order (and how are you supposed to know which should be installed first, DirectX 8 or WMSDK 6?), you can end up regressing your `winerror.h` file. It's the version conflict problem, but without the benefit of version resources.

Many teams have prevailed upon the kernel team to reserve a chunk of error codes just for them.

Networking	2100–2999
Cluster	5000–5999
Traffic Control	7500–7999
Active Directory	8000–8999
DNS	9000–9999
Winsock	10000–11999
IPSec	13000–13999
Side By Side	14000–14999

There is room for only 65535 Win32 error codes, and over an eighth of them have already been carved out by these "block assignments". I wonder if we will eventually run out of error codes prematurely due to having given away error codes in too-large chunks. (Some sort of analogy with IPv4 could be made here but I'm not going to try.)

Raymond Chen

Follow