What is the highest numerical resource ID permitted by Win32?

devblogs.microsoft.com/oldnewthing/20110217-00

February 17, 2011

Raymond Chen

A customer asked the following question:

> What is the maximum legal value of a resource identifier? Functions like `LoadString` take a `UINT` as the resource ID, which suggests a full 32-bit range, but in practice, most resource IDs appear to be in the 16-bit range. Is there any particular history/precedent for avoiding large numbers as resource IDs? (I have a program that autogenerates string IDs, and having a full 32-bit range gives me some more flexibility in assigning the IDs, but I want to make sure I don't run afoul of any limitations either.)

Let's answer the literal question first, and then look at the misconceptions behind the question.

The maximum legal value for an integer resource identifier is 65535. You don't need any special psychic powers for this; it's right there in the `MAKEINTRESOURCE` macro:

```
#define MAKEINTRESOURCEA(i) ((LPSTR)((ULONG_PTR)((WORD)(i))))
#define MAKEINTRESOURCEW(i) ((LPWSTR)((ULONG_PTR)((WORD)(i))))
#ifdef UNICODE
#define MAKEINTRESOURCE  MAKEINTRESOURCEW
#else
#define MAKEINTRESOURCE  MAKEINTRESOURCEA
#endif // !UNICODE
```

The `MAKEINTRESOURCE` macro takes the integer you passed, casts it down to a 16-bit `WORD`, and then casts the result up to a `LPTSTR`, effectively generating a pointer whose top bits are all zero (a pointer into the first 64KB of the address space).

Right off the bat, you can see that integer resources are limited to 16-bit values, because if you pass anything bigger, it'll get truncated by the cast to `WORD`.

Why does this limitation exist? Because most resource loading functions overload a single `lpName` parameter (representing the resource identifier or name) as both an integer (identifier) and a string (name). You can't have the full range of integers and the full range of pointers simultaneously if you want to be able to distinguish the two cases, so you have to choose some rule by which you can tell them apart, and the rule chosen by Win32 is that if the value is in the range `0..0xFFFF`, then the value is treated as an integer; otherwise it is treated as a pointer to a string.

This convention comes from the days of 16-bit Windows, where 32-bit pointers consisted of a 16-bit selector in the high order word and a 16-bit offset in the low order word. The selector `0x0000` is permanently invalid, so that's a natural place to "sneak in" the integers: A "pointer" whose selector is `0x0000` is really an integer smuggled inside a pointers. There was no loss of expressiveness because integers in 16-bit Windows were, well, only 16-bits wide, so the two parameter spaces (strings and integer) neatly meshed with no overlap. (This partitioning of the address space also happily lines up with the convention that in Win32, the first 64KB of address space is permanently invalid.)

Okay, so that answers the literal question, but there's more going on. Fortunately, the customer provided context: The integer range he's interested in is string identifiers, not resource identifiers.

String identifiers are not resource identifiers. As we saw earlier, strings are gathered in bundles of 16. The bottom 4 bits of the string identifier specify which string in the bundle contains the string in question, while the remaining bits form the resource identifier of the bundle. We just learned that the resource identifier is a 16-bit value, so string identifiers can go up to 65536 × 16 – 1.

The customer was pleased with this explanation, contributing the additional insight that "a corollary to string bundling is that it's more efficient to use contiguous ranges of string identifiers (at least gathering them in blocks of 16) rather than sparsely generated ones."

Raymond Chen

Follow