Decoding the parameters of a thrown C++ exception (0xE06D7363)

devblogs.microsoft.com/oldnewthing/20100730-00

July 30, 2010

Raymond Chen

Special preview content for my TechReady talk later today. I'd like to claim it was planned this way, but actually it was just a coincidence.

The Visual C++ compiler uses exception code `0xE06D7363` for C++ exceptions. Here's how you can decode the other parameters. (Handy if you're debugging a crash dump.)

Note that this information falls under the category of implementation detail. There is no guarantee that this method will continue to work in the future, so don't write code that relies on it. It's just a debugging tip.

When the C++ exception is raised, the exception code is 0xE06D7363 and there are three (possibly four) parameters.

- Parameter 0 is some internal value not important to the discussion.
- Parameter 1 is a pointer to the object being thrown (sort of).
- Parameter 2 is a pointer to information that describes the object being thrown.
- Parameter 3 is the `HINSTANCE` of the DLL that raised the exception. (Present only on 64-bit Windows.)

The object being thrown is pretty much the object being thrown, except that sometimes there is some junk in front that you have to skip over. Once you figure out what it is, you can dump it. (I haven't bothered trying to figure out exactly how much; I just dump bytes and figure out the correct start of the object by inspection.) But what is it? That's what Parameter 2 tells you, but in a very roundabout way.

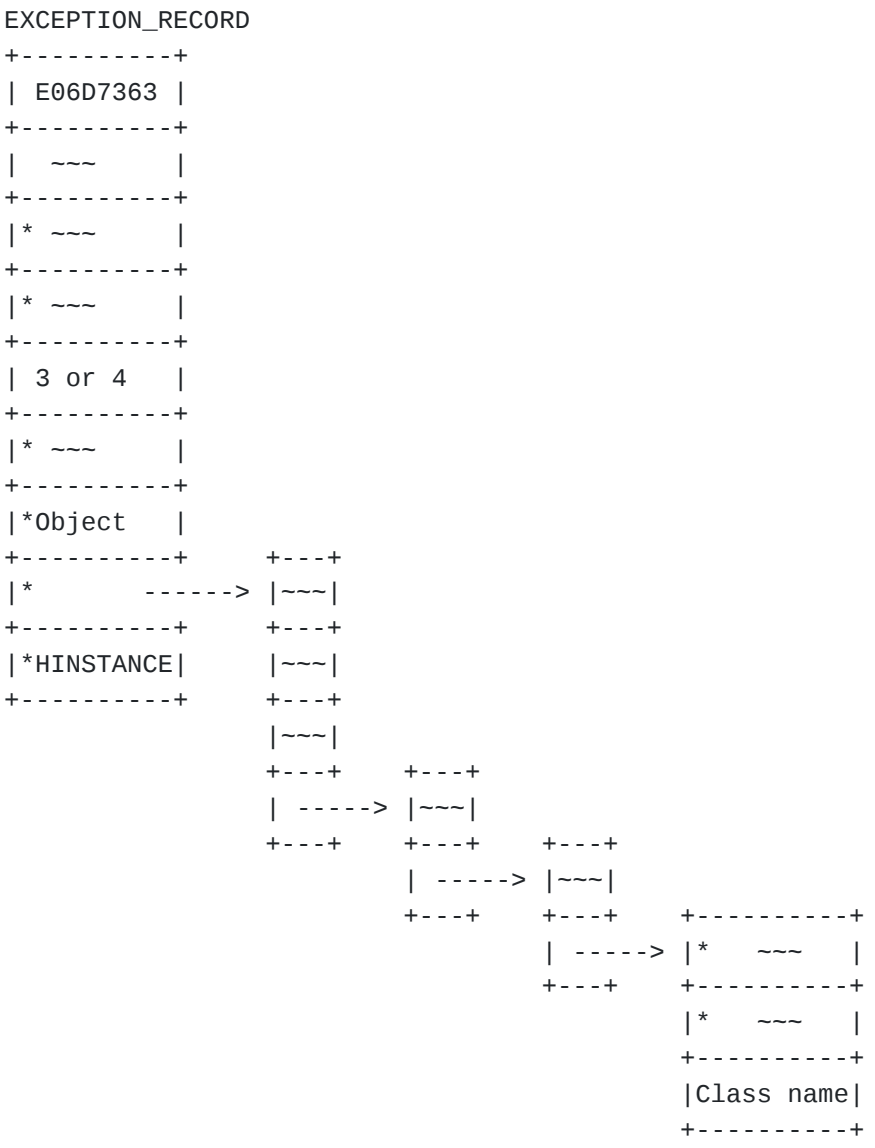
Take Parameter 2 and go to the fourth `DWORD` and treat it as a pointer. (On 64-bit systems, you have to add this value to the `HINSTANCE` passed as Parameter 3 to convert it to a pointer.)

Next, go to the second `DWORD` and treat it as a pointer. (Again, on 64-bit systems, it's really an offset from the `HINSTANCE`.)

Next, go to the second `DWORD` and treat it as a pointer. (64-bit systems: you know the drill.)

Finally, skip over the first two `void*` s and the rest is the class name.

Here's a picture, rendered in high-tech ASCII line drawing. Pointer-sized fields are marked with an asterisk, and fields whose value are unknown or not important are marked with tildes.



"When in doubt, add another level of indirection" appears to be the mantra here.

Here's a real-world example I had to debug. This came from a crash dump in a third-party application reported via Windows Error Reporting, so all debugging has to be done without source code or symbols.

```
0:008> .exr 00000000`015dede0
ExceptionAddress: 000007fefd23bb5d (KERNEL32!RaiseException+0x39)
   ExceptionCode: e06d7363 (C++ EH exception)
  ExceptionFlags: 00000001
NumberParameters: 4 // this is running on 64-bit Windows
   Parameter[0]: 0000000019930520
   Parameter[1]: 00000000015def30 // object being thrown
   Parameter[2]: 00000000100cefa8 // magic Parameter 2
   Parameter[3]: 0000000010000000 // HINSTANCE
```

According to the cookbook, we follow Parameter 2:

```
0:008> dd 00000000100cefa8 l4
00000000`100cefa8  00000000 00000000 00000000 000cefc8
                                               ^^^^^^^^
```

and take the fourth `DWORD`. Since this is a 64-bit machine, we add it to the `HINSTANCE` before dumping. (If this were a 32-bit machine, we would just dump it directly.)

```
0:008> dd 100cefc8 l2
00000000`100cefc8  00000005 000ceff8
                            ^^^^^^^^
```

Now we take the second `DWORD` (add the `HINSTANCE` since this is a 64-bit machine) and then dump it again:

```
0:008> dd 100ceff8 l2
00000000`100ceff8  00000001 000d6670
                            ^^^^^^^^
```

Okay, we're within striking distance now. Since this is a 64-bit machine, we add the `HINSTANCE` to the offset. And on all platforms, we add two pointers (which is 0x10 on a 64-bit machine and 8 on a 32-bit machine). The result should be an ASCII string representing the class name:

```
0:008> da 100d6670+10
00000000`100d6680  ".PEAVCResourceException@@"
```

If you ignore the decorations, you see that this is telling you that the object thrown was a `CResourceException`.

And for old time's sake, here's a 32-bit version I just made up now.

```
0:000> .exr 0008f2e4
ExceptionAddress: 7671b046 (kernel32!RaiseException)
   ExceptionCode: e06d7363 (C++ EH exception)
  ExceptionFlags: 00000001
NumberParameters: 3 // 32-bit platform
   Parameter[0]: 19930520
   Parameter[1]: 0008f384 // object being thrown
   Parameter[2]: 10cfed60 // magic Parameter 2
0:000> dd 10cfed60 l4
10cfed60  00000000 00000000 00000000 10db297c
0:000> dd 10db297c l2
10db297c  00000004 10db2990
0:000> dd 10db2990 l2
10db2990  00000001 10dbccac
0:000> da 10dbccac+8
10dbccb4  ".PAVCFileException@@"
```

Anyway, back to the original problem: Knowing that the object being thrown was a `CResourceException` was a big help, because that's a class used by MFC, so I have additional information as to what it does and how it's used. This turns out to have been the necessary foothold to identify the source of the problem, which will be the subject of a future write-up.

Raymond Chen

Follow