# The real cost of compatibility is not in the hacks; the hacks are small potatoes

devblogs.microsoft.com/oldnewthing/20070723-00

July 23, 2007

Raymond Chen

Commenter Myron A. Semack asks how much faster Windows would be if you took out the backward compatibility stuff. Myron is so anxious about this that he asked the question a second time. Asking a question twice typically counts as a reason *not* to answer it, but since I had already written up the answer, I figured I'd post it anyway. Oh great, and now he asked it a third time. Myron is so lucky I already wrote up the answer, because if I hadn't I would've just skipped the topic altogether. I don't respond well to nagging.

The answer is, "Not much, really."

Because the real cost of compatibility is not in the hacks. The hacks are small potatoes. Most hacks are just a few lines of code (sometimes as few as zero), so the impact on performance is fairly low. Consider a compatibility hack for programs that mess up `IUnknown::QueryInterface`:

```
...
ITargetInterface *pti = NULL;
HRESULT hr = pobj->QueryInterface(
                 IID_ITargetInterface, (void**)&pti);
if (SUCCEEDED(hr) && !pti) hr = E_FAIL;
```

The compatibility hack here was just two lines of code. One to set the `pti` variable to `NULL` and another to check for a common application error and work around it. The incremental cost of this is negligible.

Here's an example of a hack that takes zero lines of code:

```
HINSTANCE ShellExecute(...)
{
 ...
 return (HINSTANCE)42;
}
```

I count this as zero lines of code because the function has to return *something*. You may as well return a carefully-crafted value chosen for compatibility. The incremental cost of this is zero.

No, the real cost of compatibility is in the design.

If you're going to design a feature that enhances the window manager in some way, you have to think about how existing programs are going to react to your feature. These are programs that predate your feature and naturally know nothing about it. Does your feature alter the message order? Does it introduce a new point of re-entrancy? Does it cause a function to begin dispatching messages that previously did not? You may be forced to design your feature differently in order to accommodate these concerns. These issues aren't things you can "take out"; they are inherently part of the feature design.

Consider for example color NTSC. (Videophiles like to say that NTSC stands for "never twice the same color.")

The NTSC color model is backward compatible with the existing system for black-and-white television. How much cheaper would your color television be if you could take out the backward compatibility circuitry? That question misses the point. The backward compatibility is in the design of the NTSC color signal. It's not a circuit board (or, to be more historically accurate, a set of vacuum tubes) that you can pull out. You can't "take out" the compatibility stuff from your television set. The compatibility is fundamentally part of the way the NTSC color signal works.

Raymond Chen

**Follow**