ONCE UPON A TIME there was a programmer. This programmer had written many programs over the years and they worked rather well. One day, the spirit of the great OS appeared before him and presented a brand new way of programming—one that would allow the programmer to write many more programs that would work even better than before.

But the poor programmer looked at the new OS and realized that it would take many years for him to convert his previous programs (which worked rather well) to the new OS (besides, he was really more interested in cashing out what was left of his options and going sailing for a few years than in spending the next two decades porting code). And he saw there were still a few things the new OS couldn't do that he would still have to do the old way.

However, the great OS spirit appeared before him and gave him a magic lamp. He rubbed the magic lamp and a giant blue genie appeared who sang very funny songs to Disney music. The genie magically ported all of the existing code to the new OS. The genie also gave the programmer a magic ring that would cause new classes to spontaneously appear anytime the programmer needed some functionality that was missing from the OS.

The programmer went home promptly at 5:00 P.M., cashed in his remaining stock options, and went to Disney World (having been given some free tickets by the giant blue genie).

And he lived happily ever after.

Alas, if only it were so....

Unfortunately, Disney holds the copyright on giant blue genies so Microsoft was unable to provide us with a way to magically port all of our existing COM-based applications and components to .NET. All of the magic rings have been allocated to the Lord of the Rings movies.¹ So, vast as the .NET class library is, there still remain tasks it cannot accomplish. Fortunately, Microsoft did the next best thing. They invested a huge effort to make sure that new .NET programs can continue to

---

¹ With which I have no connection whatsoever.
use COM components and that they can call the underlying Win32 API calls. The former is accomplished through a feature called COM Interoperability (interop, for short). The latter, through a feature called Platform Invoke (P-Invoke). Most of the objects and attributes I’ll be referring to in this chapter can be found in the System.Runtime.InteropServices namespace.

Though these are really two different features, they are nevertheless closely related because both involve leaving the .NET Framework to work with unmanaged code.²

**COM Interop**

COM interop is the part of the .NET Framework that has to do with interoperability of existing COM components and applications with .NET components and applications. When thinking about COM interop, there are two key ideas to keep in mind:

- .NET components can only work directly with other .NET components. The .NET components all exist in memory that is managed by the CLR and are all garbage collected when they are no longer referenced by root-level variables.

- COM components can only work directly with other COM components. They all work with a specific low-level implementation that involves the use of tables of virtual pointers. COM components are reference counted—every reference to a COM object must increment a reference count. When the reference is freed, it must decrement the reference count. When the reference count reaches zero, the object is freed.

The .NET interop system acts as a gateway between .NET components and COM components. It makes a COM component look like a .NET component to other .NET objects. It makes .NET components look like COM components to other COM objects. This is illustrated in Figure 15-1.

When a .NET object wishes to use a COM object, it uses a Runtime Callable Wrapper (RCW). This is a .NET assembly that makes a COM object look like a regular .NET object.

When a COM object wishes to use a .NET object, the runtime creates a COM Callable Wrapper (CCW) that looks and works just like a COM object.

---

² Code that is not managed by the .NET Framework.