The previous chapter introduced you to the process of building COM DLLs using raw C++ and IDL. Although it is illuminating to see exactly what takes place under the hood to build a COM server from the ground up, I am sure you agree that the process is tedious and error prone. In this chapter, I draw your attention to the creation of COM servers using two popular frameworks, namely the Active Template Library 3.0 (ATL 3.0) and Microsoft Visual Basic 6.0. This chapter also illustrates how to make use of a core development tool: the OLE/COM Object Viewer (oleview.exe), which will be used throughout this text.

Obviously, entire books have been written about COM development using ATL 3.0 and VB 6.0. To be sure, a single chapter cannot do justice to each COM framework. However, given that this book is all about getting the COM and .NET architecture to coexist in harmony, I feel compelled to cover the basics of each of these COM toolkits. Even if you are a seasoned veteran of ATL 3.0 and VB 6.0, I invite you to read along and build the sample applications, given that you make use of these COM servers later in the text. So without further introductory fluff, let’s formalize the role of ATL.

The Role of the Active Template Library

ATL is a very popular C++ COM development framework that consists of a number of templates, magic macros, and base class types. The overall goal of ATL is to provide default boilerplate implementations for the necessary COM infrastructure (IUnknown, class factories, IDispatch, and so on), giving you more time to concentrate on the business problem you are trying to solve. Consider for
example the C++ ComCar you created in Chapter 2. Although all you really wanted
to do was allow the outside world to access this functionality:

// The essence of the ComCar.
STDMETHODIMP ComCar::SpeedUp(long delta)
{
    m_currSpeed += delta;
    return S_OK;
}

STDMETHODIMP ComCar::CurrentSpeed(long* currSp)
{
    *currSp = m_currSpeed;
    return S_OK;
}

STDMETHODIMP ComCar::CrankTunes()
{
    MessageBox(NULL, "Cranking music!", "ComCar", MB_OK);
    return S_OK;
}

you were required to implement the methods of IUnknown, build IDL type defini-
tions, and construct a class factory, as well as contend with several DLL exports
(not to mention register the critical server information in the system registry). If
you choose to build your COM servers using C++, ATL will be a welcome addition
to your programmer's bag of tricks.

Understand that even though ATL does provide stock implementations of
numerous COM atoms, you are always able to extend and override this default
behavior if you so choose. In any case, ATL does not exonerate you from the need
to understand IDL or the constructs of COM. To illustrate the basics, you will
construct an ATL DLL that mimics the functionality of the previous
RawComCar.dll.

**Generating the Component Housing**

Every ATL project begins by making use of the ATL COM AppWizard utility, which
can be activated from the File | New menu of the Visual Studio 6.0 IDE (Figure 3-1).
If you wish to follow along, name your project AtlCarServer and click the OK
button.