CHAPTER 25

Reflection and Attributes

- Metadata and Reflection
- The Type Class
- Getting a Type Object
- What Is an Attribute?
- Applying an Attribute
- Predefined, Reserved Attributes
- The Conditional Attribute
- More About Applying Attributes
- Custom Attributes
- Accessing an Attribute
Metadata and Reflection

Most programs are written to work on data. They read, write, manipulate, and display data. (Graphics are a form of data.) For some types of programs, however, the data they manipulate is not numbers, text, or graphics, but information about programs and program types.

- Data about programs and their classes is called metadata and is stored in the programs’ assemblies.
- A program can look at the metadata of other assemblies or of itself, while it’s running. When a running program looks at its own metadata, or that of other programs, it’s called reflection.

An object browser is an example of a program that displays metadata. It can read assemblies and display the types they contain, along with all the characteristics and members.

This chapter will look at how your programs can reflect on data using the Type class and how you can add metadata to your types using attributes.

Note To use reflection, you must use the System.Reflection namespace.

The Type Class

Throughout this text we’ve described how to declare and use the types available in C#. These include the predefined types (int, long, string, and so on), types from the BCL (Console, IEnumerable, and so on), and user-defined types (MyClass, MyDel, and so on). Every type has its own members and characteristics.

The BCL declares an abstract class called Type, which is designed to contain the characteristics of a type. Using objects of this class allows you to get information about the types your program is using.

Since Type is an abstract class, it cannot have actual instances. Instead, at run time, the CLR creates instances of a class derived from Type (RuntimeType) that contains the type information. When you access one of these instances, the CLR returns a reference, not of the derived type but of the base class Type. For simplicity’s sake, though, throughout the rest of the chapter, we’ll call the object pointed at by the reference an object of type Type, although technically it’s an object of a derived type that is internal to the BCL.

The following are important things to know about Type:

- For every type used in a program, the CLR creates a Type object that contains the information about the type.
- Regardless of the number of instances of a type that are created, there is only a single Type object associated with all the instances.