Microsoft

LESSON 3.3-3.4

10754 Microsoft .NET Fundamentals

Understand Version Control, Assemblies, and Metadata

Lesson Overview

How does the Microsoft® .NET Framework manage versions, assemblies, and metadata?

In this lesson, you will learn:

- How .NET versions are identified.
- The role of assemblies in a .NET application.
- How .NET incorporates metadata in compiled assemblies.

Guiding Questions

1. How are different versions of a library or application identified?
2. What are assemblies?
3. What metadata does .NET create when assemblies are compiled?

Anticipatory Set

Imagine that you have created a database application for keeping track of a vinyl record collection. You have identified your most recent update of the application as version 2.0.

- You make a major update that adds social networking capabilities, but it requires the user to convert the database to a newer format. This change means that their data no longer works with version 2.0. What version number should you use to identify this updated software?
- Instead of changing the database, you decide to retain the old format and implement only some of the social networking features; this allows users to go back to running version 2.0 if they prefer. How should you identify this version of the application?

Versioning Basics

- Versioning of projects in the .NET Framework is done at the assembly level—that is, each assembly has its own version that is independent of other assemblies.
- Each assembly has a version number as part of its identity. Two otherwise identical assemblies with different version numbers are considered by the run time to be completely different assemblies.
- By default, the common language runtime will run only applications with the assembly versions they were built with; this policy can be overridden by the developer.
- This version checking applies only to strongly named assemblies.

Version Numbers

- This version number is represented as a four-part string with the following format:

  <major version>.<minor version>.<build number>.<revision>

- For example, version 1.5.1254.0 indicates 1 as the major version, 5 as the minor version, 1254 as the build number, and 0 as the revision number.

Version Number Components

- **Major**: Assemblies with the same name but different major versions are not interchangeable.
- **Minor**: If the name and major version number on two assemblies are the same, but the minor version number is different, this indicates significant enhancement with the intention of backward compatibility.
- **Build**: A difference in build number represents a recompilation of the same source. Different build numbers might be used when the processor, platform, or compiler changes.
- **Revision**: Assemblies with the same name, major and minor version numbers, and different revisions are intended to be fully interchangeable. A higher revision number might be used in a build that fixes a security hole in a previously released assembly.

Using Multiple Versions

- Side-by-side execution
  - The .NET Framework has the ability to store and execute multiple versions of an application or component on the same computer.
- Three guidelines for implementing side-by-side execution:
  - Use strongly named assemblies (which include version information as part of their naming structure).
  - Use version-aware storage, such as the Global Assembly Cache (GAC), which do not overwrite previous versions of an assembly.
  - Design the application or library to run in isolation by using version-specific directory structures and registry entries.

Definition of Assemblies

Think of an assembly as a collection of types and resources that form a logical unit of functionality and are built to work together.

- Assemblies form the fundamental unit of deployment, version control, reuse, activation scoping, and security permissions for a .NET-based application.
- Assemblies take the form of an executable (.exe) file or dynamic-link library (.dll) file, and are the building blocks of the .NET Framework.
- Assemblies provide the common language runtime with the information that it needs to be aware of type implementations.

Creating Assemblies

Assemblies can be static or dynamic:

- Static assemblies can include .NET Framework types (interfaces and classes), as well as resources for the assembly (bitmaps, JPEG files, resource files, and so on).
  - Static assemblies are stored on disk in portable executable (PE) files.
  - Static assemblies are generally created using an application template or a class library template in Microsoft Visual Studio®.
- Dynamic assemblies are run directly from memory and are not saved to disk before execution. You can save dynamic assemblies to disk after they have executed.
  - You can also use common language runtime application programming interfaces (APIs), such as Reflection.Emit, to create dynamic assemblies.

Metadata

- Metadata is defined as “data about data.”
- Metadata provides a description (or other details) about a file or a set of data.
- Many web pages include metadata that provides information about the page: its author, the authoring tool used to create the page, keywords about the content, and so on.
- This information can be accessed and used for a variety of purposes, such as cataloging the page for a search engine or optimizing the way that the page is displayed.

Assembly Metadata

- In the .NET Framework, every assembly contains a collection of data that describe how the elements in the assembly relate to each other.
- This metadata is important for making assemblies usable across languages, and it ensures that assemblies are “self-describing.”
  - An assembly’s metadata contains everything needed to interact with another assembly or module.
- Metadata enable the use of attributes, or user-defined metadata, that give the developer more control over how an assembly behaves at run time.

Assembly Manifest

- Assembly metadata is stored in a PE file (or a portion of a PE file).
  - In the case of a single-file assembly, the manifest is contained in the single file; if an assembly includes multiple files (such as resources), the compiler creates a separate manifest file.
- Used by the Just-In-Time (JIT) compiler when the application is executed.
- Stored as Extensible Markup Language (XML). View a manifest file using a standard text editor.

Assembly Manifest Components

The assembly manifest includes the following:

- Assembly name and version number
- Information about the culture or language the assembly supports
- Strong name information (a public key)
- List of all files in the assembly
- Type reference information: Information used by the run time to map a type reference to the file that contains its declaration and implementation
- Information on referenced assemblies: A list of other assemblies that are referenced statically by the assembly

Ticket Out the Door

1. Explain the four different parts of a version number (for example, 1.7.234.0).
2. Define the term “assembly” in the context of .NET applications.
3. Where does the .NET compile store assembly metadata? Give a few examples of the type of data that it stores.