



Module 12

Creating Reusable Types and Assemblies

- Examining Object Metadata
- Creating and Using Custom Attributes
- Generating Managed Code
- Versioning, Signing, and Deploying Assemblies

Lesson 1: Examining Object Metadata

- What Is Reflection?
- Loading Assemblies by Using Reflection
- Examining Types by Using Reflection
- Invoking Members by Using Reflection
- Demonstration: Inspecting Assemblies

What Is Reflection?

- Reflection enables you to inspect and manipulate assemblies at run time
- The **System.Reflection** namespace contains:
 - **Assembly**
 - **TypeInfo**
 - **ParameterInfo**
 - **ConstructorInfo**
 - **FieldInfo**
 - **MemberInfo**
 - **PropertyInfo**
 - **MethodInfo**

- The **Assembly.LoadFrom** method

```
var assemblyPath = "...";  
var assembly = Assembly.LoadFrom(assemblyPath);
```

- The **Assembly.ReflectionOnlyLoad** method

```
var assemblyPath = "...";  
var rawBytes = File.ReadAllBytes(assemblyPath);  
var assembly = Assembly.ReflectionOnlyLoad(rawBytes);
```

- The **Assembly.ReflectionOnlyLoadFrom** method

```
var assemblyPath = "...";  
var assembly = Assembly.ReflectionOnlyLoadFrom(assemblyPath);
```

- Get a type by name

```
var assembly = FourthCoffeeServices.GetAssembly();  
var type = assembly.GetType("...");
```

- Get all of the constructors

```
var constructors = type.GetConstructors();
```

- Get all of the fields

```
var fields = type.GetFields();
```

- Get all of the properties

```
var properties = type.GetProperties();
```

- Get all of the methods

```
var methods = type.GetMethods();
```

- Instantiate a type

```
var type = FourthCoffeeServices.GetHandleErrorType();  
...  
var constructor = type.GetConstructor(new Type[0]);  
...  
var initializedObject = constructor.Invoke(new object[0]);
```

- Invoke methods on the instance

```
var methodToExecute = type.GetMethod("LogError");  
var initializedObject = FourthCoffeeServices.InstantiateHandleErrorType();  
...  
var response = methodToExecute.Invoke(initializedObject,  
    new object[] { "Error message" }) as string;
```

- Get or set property values on the instance

```
var property = type.GetProperty("LastErrorMessage");  
var initializedObject = FourthCoffeeServices.InstantiateHandleErrorType();  
...  
var lastErrorMessage = property.GetValue(initializedObject) as string;
```

Demonstration: Inspecting Assemblies

In this demonstration, you will create a tool that you can use to inspect the contents of an assembly.

Lesson 2: Creating and Using Custom Attributes

- What Are Attributes?
- Creating and Using Custom Attributes
- Processing Attributes by Using Reflection
- Demonstration: Consuming Custom Attributes by Using Reflection

What Are Attributes?

- Use attributes to provide additional metadata about an element
- Use attributes to alter run-time behavior

```
[DataContract(Name = "SalesPersonContract", IsReference=false)]
public class SalesPerson
{
    [Obsolete("This property will be removed in the next release.")]
    [DataMember]
    public string Name { get; set; }

    ...
}
```

Derive from the **Attribute** class or another attribute

```
[AttributeUsage(AttributeTargets.All)]
public class DeveloperInfo : Attribute
{
    private string _emailAddress;
    private int _revision;

    public DeveloperInfo(string emailAddress, int revision)
    {
        this._emailAddress = emailAddress;
        this._revision = revision;
    }
}
```

```
[DeveloperInfo("holly@fourthcoffee.com", 3)]
public class SalePerson
{
    ...
}
```

Use reflection to access the metadata that is encapsulated in custom attributes

```
var type = FourthCoffee.GetSalesPersonType();  
  
var attributes = type.GetCustomAttributes(typeof(DeveloperInfo),  
false);  
  
foreach (var attribute in attributes)  
{  
    var developerEmailAddress = attribute.EmailAddress;  
    var codeRevision = attribute.Revision;  
}
```

Demonstration: Consuming Custom Attributes by Using Reflection



In this demonstration, you will use reflection to read the **DeveloperInfo** attributes that have been used to provide additional metadata on types and type members.

Lesson 3: Generating Managed Code

- What Is CodeDOM?
- Defining a Type and Type Members
- Compiling a CodeDOM Model
- Compiling Source Code into an Assembly

What Is CodeDOM?

- Define a model that represents your code by using:
 - The **CodeCompileUnit** class
 - The **CodeNamespace** class
 - The **CodeTypeDeclaration** class
 - The **CodeMemberMethod** class
- Generate source code from the model:
 - Visual C# by using the **CSharpCodeProvider** class
 - JScript by using the **JScriptCodeProvider** class
 - Visual Basic by using the **VBCodeProvider** class
- Generate a .dll or a .exe that contains your code

Defining a type with a **Main** method

```
var unit = new CodeCompileUnit();

var dynamicNamespace = new CodeNamespace("FourthCoffee.Dynamic");
unit.Namespaces.Add(dynamicNamespace);

dynamicNamespace.Imports.Add(new CodeNamespaceImport("System"));

var programType = new CodeTypeDeclaration("Program");
dynamicNamespace.Types.Add(programType);

var mainMethod = new CodeEntryPointMethod();
programType.Members.Add(mainMethod);

var expression = new CodeMethodInvokeExpression(
    new CodeTypeReferenceExpression("Console"), "WriteLine",
    new CodePrimitiveExpression("Hello Development Team...!!"));
```


Generate source code files from your CodeDOM model

```
var provider = new CSharpCodeProvider();

var fileName = "program.cs";
var stream = new StreamWriter(fileName);
var textWriter = new IndentedTextWriter(stream);

var options = new CodeGeneratorOptions();
options.BlankLinesBetweenMembers = true;

var compileUnit = FourthCoffee.GetModel();
provider.GenerateCodeFromCompileUnit(
    compileunit,
    textWriter,
    options);

textWriter.Close();
stream.Close();
```

Generate an assembly from your source code files

```
var provider = new CSharpCodeProvider();

var compilerSettings = new CompilerParameters();
compilerSettings.ReferencedAssemblies.Add("System.dll");
compilerSettings.GenerateExecutable = true;
compilerSettings.OutputAssembly = "FourthCoffee.exe";

var sourceCodeFileName = "program.cs";
var compilationResults = provider.CompileAssemblyFromFile(
    compilerSettings,
    sourceCodeFileName);

var buildFailed = false;
foreach (var error in compilationResults.Errors)
{
    var errorMessage = error.ToString();
    buildFailed = true;
}
```

Lesson 4: Versioning, Signing, and Deploying Assemblies

- What Is an Assembly?
- What Is the GAC?
- Signing Assemblies
- Versioning Assemblies
- Installing an Assembly into the GAC
- Demonstration: Signing and Installing an Assembly into the GAC
- Demonstration: Specifying the Data to Include in the Grades Report Lab

What Is an Assembly?

- An assembly is a collection of types and resources
- An assembly is a versioned deployable unit
- An assembly can contain:
 - IL code
 - Resources
 - Type metadata
 - Manifest

What Is the GAC?

- The GAC provide a robust solution to share assemblies between multiple application on the same machine
- Find the contents of the GAC at `C:\Windows\assembly`
- Benefits:
 - Side-by-side deployment
 - Improved loading time
 - Reduced memory consumption
 - Improved search time
 - Improved maintainability

- Sign an assembly:

- Create a key file

```
sn -k FourthCoffeeKeyFile.snk
```

- Associate the key file with an assembly

```
[assembly: AssemblyKeyFileAttribute("FourthCoffeeKeyFile.snk")]
```

- Delay the signing of an assembly:

1. Open the properties for the project
2. Click the Signing tab
3. Select the Sign the assembly check box
4. Specify a key file
5. Select the Delay sign only check box

- A version number of an assembly is a four-part string:

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

- Applications reference particular versions of assemblies

```
<configuration>  
  <runtime>  
    <assemblyBinding xmlns="...">  
      <dependentAssembly>  
        <assemblyIdentity name="FourthCoffee.Core"  
          publicKeyToken="32ab4ba45e0a69a1" culture="en-us" />  
        <bindingRedirect oldVersion="1.0.0.0" newVersion="2.0.0.0"/>  
      </dependentAssembly>  
    </assemblyBinding>  
  </runtime>  
</configuration>
```

Installing an Assembly into the GAC

Install an assembly in the GAC by using:

- Global Assembly Cache tool
- Microsoft Windows Installer

Examples:

- Install an assembly by using Gacutil.exe:

```
gacutil -i "<pathToAssembly>"
```

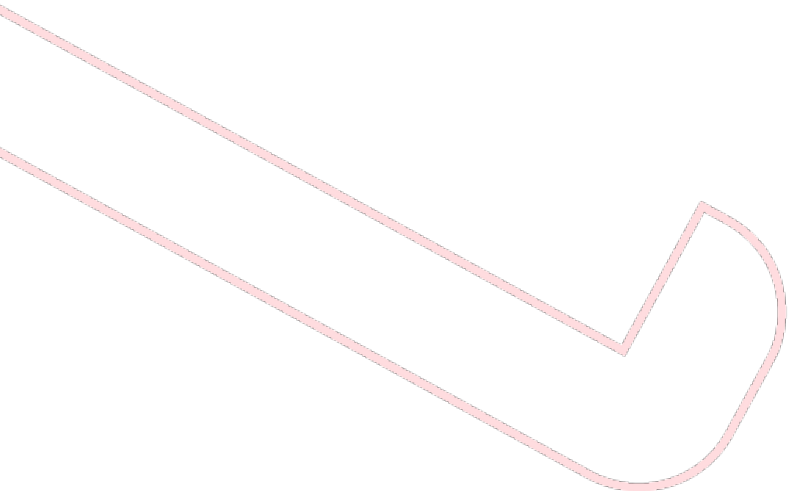
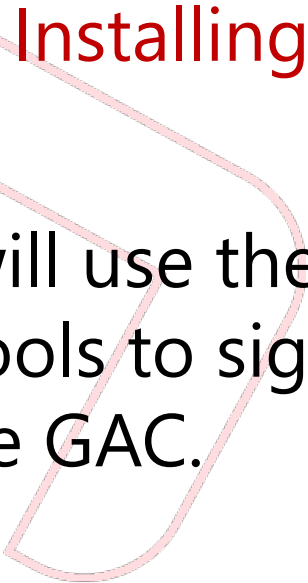
- View an assembly by using Gacutil.exe:

```
gacutil -l "<assemblyName>"
```


Demonstration: Signing and Installing an Assembly into the GAC



In this demonstration, you will use the Sn.exe and Gacutil.exe command-line tools to sign and install an existing assembly into the GAC.



Demonstration: Specifying the Data to Include in the Grades Report Lab

In this demonstration, you will see the tasks that you will perform in the lab for this module.

Lab: Specifying the Data to Include in the Grades Report

- Exercise 1: Creating and Applying the IncludeInReport attribute
- Exercise 2: Updating the Report
- Exercise 3: Storing the Grades.Utilities Assembly Centrally (If Time Permits)

Logon Information

- Virtual Machine: 20483B-SEA-DEV11, MSL-TMG1
- User Name: Student
- Password: Pa\$\$w0rd

Estimated Time: 75 minutes

You decide to update the Grades application to use custom attributes to define the fields and properties that should be included in a grade report and to format them appropriately. This will enable further reusability of the Microsoft Word reporting functionality.

You will host this code in the GAC to ensure that it is available to other applications that require its services.

Module Review and Takeaways

- Review Question(s)