Module 11

Integrating with Unmanaged Code

Module Overview

- Creating and Using Dynamic Objects
- Managing the Lifetime of Objects and Controlling Unmanaged Resources

Lesson 1: Creating and Using Dynamic Objects

- What Are Dynamic Objects?
- What Is the Dynamic Language Runtime?
- Creating a Dynamic Object
- Invoking Methods on a Dynamic Object
- Demonstration: Interoperating with Microsoft Word

What Are Dynamic Objects?

- Objects that do not conform to the strongly typed object model
- Objects that enable you to take advantage of dynamic languages, such as IronPython
- Objects that simplify the process of interoperating with unmanaged code

What Is the Dynamic Language Runtime?

The DLR provides:

- Support for dynamic languages, such as IronPython
- Run-time type checking for dynamic object
- Language binders to handle the intricate details of interoperating with another language

Creating a Dynamic Object

- Dynamic objects are declared by using the **dynamic** keyword

```
using Microsoft.Office.Interop.Word;
...
dynamic word = new Application();
```

- Dynamic objects are variables of type **object**
- Dynamic objects do not support:
  - Type checking at compile time
  - Visual Studio IntelliSense

Invoking Methods on a Dynamic Object

- You can access members by using the dot notation

```
string filePath = "C:\\FourthCoffee\\Documents\\Schedule.docx";
...
dynamic word = new Application();
dynamic doc = word.Documents.Open(filePath);
doc.SaveAs(filePath);
```

- You do not need to:
  - Pass **Type.Missing** to satisfy optional parameters
  - Use the **ref** keyword
  - Pass all parameters as type **object**

Demonstration: Interoperating with Microsoft Word

In this demonstration, you will use dynamic objects to consume the Microsoft.Office.Word.Interop COM assembly in an existing .NET Framework application.

Lesson 2: Managing the Lifetime of Objects and Controlling Unmanaged Resources

- The Object Life Cycle
- Implementing the Dispose Pattern
- Managing the Lifetime of an Object
- Demonstration: Upgrading the Grades Report Lab

The Object Life Cycle

- When an object is created:
  1. Memory is allocated
  2. Memory is initialized to the new object
- When an object is destroyed:
  1. Resources are released
  2. Memory is reclaimed

Implementing the Dispose Pattern

Implement the **IDisposable** interface

```
public class ManagedWord : IDisposable
{
   bool _isDisposed;

   ~ManagedWord
   {
       Dispose(false);
   }

   public void Dispose()
   {
      Dispose(true);
      GC.SuppressFinalize(this);
   }

   protected virtual void Dispose(bool isDisposing) { ... }

}
```

Managing the Lifetime of an Object

- Explicitly invoke the **Dispose** method

```
var word = default(ManagedWord);
try
{
   word = new ManagedWord();
   // Code to use the ManagedWord object.
}
finally
{
   if(word!=null) word.Dispose();
}
```

- Implicitly invoke the **Dispose** method

```
using (var word = default(ManagedWord))
{
   // Code to use the ManagedWord object.
}
```

Demonstration: Upgrading the Grades Report Lab

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

Lab: Upgrading the Grades Report

- Exercise 1: Generating the Grades Report by Using Word
- Exercise 2: Controlling the Lifetime of Word Objects by Implementing the Dispose Pattern

Logon Information

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

Estimated Time: 60 minutes

Lab Scenario

You have been asked to upgrade the grades report functionality to generate reports in Word format. In Module 6, you wrote code that generates reports as an XML file; now you will update the code to generate the report as a Word document.

Module Review and Takeaways

- Review Question(s)