Hands-On Lab

Lab Manual

HOL CSI 20 - How to use Enterprise Services through the .NET Framework

Please do not remove this manual from the lab
The lab manual will be available from CommNet

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

©2005 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, MS, Windows, Windows NT, MSDN, Active Directory, BizTalk, SQL Server, SharePoint, Outlook, PowerPoint, FrontPage, Visual Basic, Visual C++, Visual J++, Visual InterDev, Visual SourceSafe, Visual C#, Visual J#, and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries.

Other product and company names herein may be the trademarks of their respective owners.

Contents

Lab 1: Using Enterprise Services

Lab Objective

Estimated time to complete this lab: 60 minutes

The objective of this lab is introduces you to Enterprise Services, which makes COM+ services available to your .NET applications.

- Building a Class Library and Test Application
- Using Enterprise Services
- Using AutoComplete
- Using Object Pooling and Just-in-Time Activation

Exercise 1 – Building a Class Library and Test Application

In this exercise, you will create a class library and console test application in Visual C# .NET. You will use the Microsoft Visual Studio .NET Integrated Development Environment (IDE) to create, compile, and run this lab.

1. Create a new project in the IDE

a. Click **Start->Programs->Microsoft Visual Studio 2005 Beta 2->Microsoft Visual Studio 2005 Beta 2**

b. Click **File->New->Project**.

c. In the Project Types pane, click **Visual C#** under **Project types**.

d. In the Templates pane, click **Class Library**.

e. Type "**HelloEnterpriseServices**" in the Name field, and then choose a location for your project.

f. Click **OK**.

2. Rename class1

a. Click View->Solution Explorer.

b. In the Solution Explorer, right-click on the file **class1.cs**.

c. Select Rename.

d. Change the name to "**HelloTx.cs**".

3. Add a reference to System.Windows.Forms

If you want to use a message box in Visual C#, you must add a reference to the assembly that defines the MessageBox class, which is System.Windows.Forms.dll

a. Click **View->Solution Explorer**.

b. Right-click **References** and click **Add Reference…**

c. With the .NET tab selected, select the component name **System.Windows.Forms**.

d. Click **OK**.

4. Add a using statement for System.Windows.Forms

a. At the top of the "**HelloTx.cs**" source file add the following:

```
using System.Windows.Forms;
```

5. Add a method to the class

a. Add the following function to your HelloTx class:

```
public void SayHello(string Name)
{
  MessageBox.Show(string.Format("Hello {0}", Name));
}
```

6. Build your application

a. Select **Build**->**Build Solution**.

b. If the build is successful, you will see the following in the output window:

```
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

7. Add a test project

Class libraries cannot be run directly, so you need to add a test project that can create a class and call its methods.

a. Select **File->Add ->New Project…**

b. Select **Console Application** from the **Visual C#** project templates.

c. Type "**TestEnterpriseServices**" in the name field, and then choose a location for your project.

d. Click **OK**.

8. Add a reference from the test project to the class library

a. Click **View->Solution Explorer**.

b. Right-click on **References** under the **TestEnterpriseServices** project and click **Add Reference…**

c. Select the **Projects** tab, select the project **HelloEnterpriseServices**.

d. Select – HelloEnterpriseServices will now appear in the Selected Components list.

e. Click **OK**.

9. Mark the test project as the startup project

a. Click **View->Solution Explorer**.

b. Right-click **TestEnterpriseServices** project.

c. Click **Set as StartUp Project**.

10. Rename Program in TestEnterpriseServices

a. In the Solution Explorer, right-click on the file **program.cs**.

b. Select **Rename**.

c. Change the name to **TestMain.cs**.

11. Add a using statement for HelloEnterpriseServices

a. At the top of the **TestMain.cs** source file from the TestEnterpriseServices project add the following:

```
using HelloEnterpriseServices;
```

12. Add a call to the HelloTx object

a. Add a new static method called **ShowTransaction** to the TestMain class. Replace <Your Name> with your own name:

```
public static void ShowTransaction()
{
    try
    {
        HelloTx MyTxObj = new HelloTx();
        MyTxObj.SayHello("<Your Name>");
    }
    catch(Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

b. Add a new method **PauseForExit** to the TestMain class:

```
public static void PauseForExit()
{
  Console.WriteLine("Press <Enter> to exit...");
  Console.ReadLine();
}
```

c. Modify the body of the **Main** method of the TestMain class by adding the two lines shown:

```
static void Main(string[] args)
{
  ShowTransaction();
  PauseForExit();
}
```

13. Build and run the test program

a. Click **Debug->Start Debugging**.

b. You should see a message box saying "Hello to <Your Name>" – You may have to look underneath other windows to find this message box.

Conclusion

In this lab, you created a simple class library and a console application to test the class library. This is the pattern developers typically follow when building test harnesses that test their middle-tier components. In the next exercise, you will begin to take advantage of Enterprise Services by adding transaction support to your component.

Exercise 2 – Using Enterprise Services

In this exercise, you will you will modify the class library created in Exercise 1 to make use of transactions provided by Enterprise Services. You will also learn about dynamic registration as you run your application and observe its interaction with the Distributed Transaction Coordinator.

1. Open the HelloEnterpriseServices.sln project

Use the project from the following folder as a starting point: C:\Microsoft Hands-On-Lab\HOL-CSI20\Sources\Exercises\Exercise 2\HelloEnterpriseServices.

a. Click **Start->Programs->Microsoft Visual Studio 2005 Beta 2->Microsoft Visual Studio 2005 Beta 2**

b. Click **File->Open->Project/Solution**.

c. Navigate to **C:\Microsoft Hands-On-Lab\HOL-CSI20\Sources\Exercises\Exercise 2\HelloEnterpriseServices**

d. Select **HelloEnterpriseServices.sln**

e. Click **Open**

2. Add a reference to System.EnterpriseServices for both the HelloEnterpriseServices and TestEnterpriseServices projects

a. Click **View->Solution Explorer**.

b. Right-click on **References** under the **HelloEnterpriseServices** project and click **Add Reference…**

c. With the .NET tab selected, select the component name **System.EnterpriseServices**.

d. Click **OK**.

e. Repeat the same above procedure steps (b-d) for **TestEnterpriseServices**.

f. At the top of the "**HelloTx.cs**" file file add the following line

```
using System.EnterpriseServices;
```

Visual C# requires that all types referenced through inheritance be referenced at compile time. Therefore, TestEnterpriseServices must have a reference to this type even though it does not directly use EnterpriseServices.

3. Add a transaction attribute to HelloTx

a. Add the transaction attribute to the HelloTx class as shown below:

```
[Transaction]
public class HelloTx
```

The transaction attribute tells EnterpriseServices that all the methods of this class require a transaction. You can pass different values to the transaction constructor to specify other values like RequiresNew, Supported, Disabled, or Ignored. Because this class is using a transaction, it is also required to enable Just-in-Time activation. This means that your class should be stateless because it will be deactivated after every method call.

4. Modify the HelloTx class to be derived from ServicedComponent

a. Add ServicedComponent to the HelloTx class declaration as shown below:

```
[Transaction]
public class HelloTx : ServicedComponent
```

Classes that take advantage of the services provided by EnterpriseServices must inherit from ServicedComponent.

5. Modify the SayHello function to commit or abort the transaction

a. Modify the SayHello function as shown below:

```
public void SayHello(string Name)
{
        Guid txID = ContextUtil.TransactionId;
        string Msg = string.Format("Hello {0} - Commit transaction {1}?", Name, txID);
        switch (MessageBox.Show(Msg,"Transaction", MessageBoxButtons.YesNo))
        {
                case DialogResult.Yes:
                        ContextUtil.SetComplete();
                        break;
                case DialogResult.No:
                        ContextUtil.SetAbort();
                        break;
        }
}
```

The HelloTx class participates in the transaction by voting on the transaction outcome. In Visual C# .NET you can affect the object context through static methods of the ContextUtil class.

Note: It is necessary to obtain the transaction ID from ContextUtil to force Enterprise Services to actually obtain a transaction. No transaction is created until it is required. In most business applications, you will be writing to a database that would cause a transaction to be created.

6. Generate a public/private key pair

a. Open the Visual Studio .NET command prompt by clicking **Start->All Programs-> Microsoft Visual Studio 2005 Beta 2->Visual Studio Tools-> Visual Studio 2005 Command Prompt**.

b. Change directory to the directory where the HelloEnterpriseServices.sln file is located by typing: .

```
cd "C:\Microsoft Hands-On-Lab\HOL-CSI20\Sources\Exercises\Exercise 2\HelloEnterpriseServices"
```

c. At the command prompt, type the following command to generate a new key file:

```
sn /k mykey.snk
```

EnterpriseServices requires that assemblies be signed before they can be registered with the COM+ catalog.

7. Add an assembly attribute to use the key file to sign the assembly

a. In the Solution Explorer, double-click on the AssemblyInfo.cs file under the HelloEnterpriseServices project.

b. Add the following line at the top of the file:

```
using System.EnterpriseServices;
```

Importing EnterpriseServices gives you access to the assembly-level attributes defined for EnterpriseServices class libraries such as ApplicationName.

c. Add the following lines to the end of the file:

```
[assembly: AssemblyKeyFile(@"..\..\mykey.snk")]
[assembly: ApplicationName("Hello Enterprise Services")]
```

d. In the Solution Explorer, double-click on the TestMain.cs file under the TestEnterpriseServices project.

e. Add the following line at the top of the file:

```
using HelloEnterpriseServices;
```

The application name attribute determines the name of the COM+ application containing this assembly.

8. Start the Component Services administrative tool

a. Click **Start-> Administrative Tools ->Component Services**.

b. Expand the tree view under **Console Root-> Component Services->Computers->My Computer->COM+ Applications**

*Notice that there is no application named "Hello Enterprise Services" under **COM+ Applications**.*

9. Run the program

a. Switch to Visual Studio 2005 Beta.

b. Select **Debug | Start Without Debugging**.

*Once the application has started, you will see a message box asking you to commit or abort the transaction. Before you respond, return to the Component Services administrative tool, refresh the view by selecting the **COM+ Applications** node and pressing **F5**, and notice that there is now a "Hello Enterprise Services" application. Don't wait too long though. The transaction will time out after 60 seconds.*

10. View the component transaction

a. Switch to Component Services.

b. In the Component Services administrative tool, select the Distributed Transaction Coordinator node.

c. Expand the Distributed Transaction Coordinator node and select Transaction List.

Notice that the transaction created for your component is listed on the transaction list. This transaction will remain listed until it is resolved either when you select **Yes** or **No**, or when the transaction times out after 60 seconds (the default time-out). If the transaction times out and you answer **Yes** to commit the transaction, an exception saying "The root transaction wanted to commit, but the transaction aborted" will be thrown by Enterprise Services. Your component cannot catch this exception because it has already been deactivated. The client code must catch and deal with this type of error.

11. Commit or abort the transaction

a. Switch back to the message box displayed by your application.

b. To commit the transaction, click **Yes;** to abort the transaction, click **No**.

12. View the transaction statistics

a. Select Transaction Statistics in the Component Services administrative tool.

b. You will notice that there are probably many committed transactions already listed. If you aborted the transaction, it is likely that you will see only the one aborted transaction in the statistics.

The other committed transactions occurred when EnterpriseServices registered your assembly in the COM+ catalog.

13. View the Hello Enterprise Services application

a. In the Component Services administrative tool, refresh the COM+ Applications view and expand it. You will notice that a new application "Hello Enterprise Services" has been added to the list.

This application was added through a process known as dynamic registration. The first time a class that uses EnterpriseServices is loaded, it will be added to the COM+ catalog if the class has not previously been registered with COM+. If you build the application multiple times, the components will appear multiple times in the catalog. This allows you to have several different versions of your component in the COM+ catalog at the same time.

Note: The process must be running with Administrative privileges on the local machine in order to use dynamic registration

Conclusion

In this exercise, you added EnterpriseServices to your component, learned how to deploy your applications by running them with dynamic registration, and learned how your applications interact with the COM+ runtime. In the next section, we will examine more advanced techniques using EnterpriseServices.

Exercise 3 – Using AutoComplete

In this exercise, you will again modify the class library created in the previous exercises to take advantage of the AutoComplete feature

1. Open the HelloEnterpriseServices.sln project

Use the project from the following folder as a starting point: **C:\Microsoft Hands-On-Lab\HOL-CSI20\Sources\Exercises\Exercise 3\HelloEnterpriseServices**

2. Add a new method to HelloTx

a. Open the HelloTx.cs file and add the following method to the HelloTx class:

```
[AutoComplete]
public void RandomOutcome()
{
  string Message;
  if (!RandomCommit(out Message))
          throw new Exception(Message);
  else
          Console.WriteLine(Message);
}
```

This function uses another function called RandomCommit to determine if the transaction should be committed or not. If RandomCommit returns False, this function will throw an exception. Because the method is tagged with the AutoComplete attribute, throwing this exception will cause the transaction to abort. If you want the transaction to commit, you don't have to do anything because success is assumed with the AutoComplete attribute. Using AutoComplete simplifies the code by eliminating the need to catch exceptions and call ContextUtil.SetAbort.

3. Make a call to RandomOutcome from the test program

a. Open TestMain.cs from the TestEnterpriseServices project.

b. Modify the Main() method as follows (by adding the line: DoRandomOutcomeTx();):

```
static void Main(string[] args)
{
  ShowTransaction();
  DoRandomOutcomeTx();
  PauseForExit();
}
```

Now our test program will call RandomOutcome 50 times, which will randomly choose to commit or abort transactions. When transactions are aborted, RandomOutcome will throw an exception stating that the transaction was aborted.

4. Run the program

1. Select **Debug | Start Without Debugging**.

After the initial message box asking you to commit or abort the transaction, you will see a list or transactions that were committed or aborted based on the random outcome.

Conclusion

The AutoComplete feature simplifies the development of .NET class libraries by eliminating the need for you to catch exceptions and call SetComplete or SetAbort.

Exercise 4 – Object Pooling and Just-in-Time Activation

In this exercise, you will modify a class that caches data values to observe the impact of object pooling as a performance enhancement.

1. Open the HelloEnterpriseServices.sln project

Use the project from the following folder as a starting point: **C:\Microsoft Hands-On-Lab\HOL-CSI20\Sources\Exercises\Exercise 4\HelloEnterpriseServices**

2. Modify Main to call GetStuffFromCache in TestMain.cs.

a. Open TestMain.cs from the TestEnterpriseServices project.

b. Modify Main() as shown below:

```
static void Main(string[] args)
{
  ShowTransaction();
  DoRandomOutcomeTx();
  GetStuffFromCache();
  PauseForExit();
}
```

3. Add the GetRandomCacheValue function

a. Add a new function **GetRandomCacheValue** to the TestMain class as shown below:

```
public static string GetRandomCacheValue(Random r)
{
        using (StuffCache cache = new StuffCache())
        {
                int index;
                index = r.Next(0, 5000);
                return String.Format("Stuff({0})={1}", index, cache.Stuff[index]);
        }
}
```

It may seem strange to have a function that creates a new cache every time it is called, since this defeats the purpose of caching. The reason is that you are going to run the application without caching to observe the performance gains provided by object pooling.

4. Run the program

a. Select **Debug** | **Start Without Debugging**.

Notice as you run this program that it will display 100 values randomly selected from the cache and list how long it took to get these values. If you check the output pane in Visual Studio, you will notice that the StuffCache constructor is called each time the GetRandomCacheValue function is called.

5. Modify the StuffCache class in HelloTx.cs

a. Add the object pooling and Just-in-Time activation attributes to the class StuffCache by adding the first two lines as shown below:

```
[ObjectPooling(true, 1, 5)]
[JustInTimeActivation]
public class StuffCache : ServicedComponent
```

The object pooling attribute creates an object pool for this class with a minimum pool size of 1 and a maximum pool size of 5. This class also uses Just-in-Time activation, which means that the object will be returned to the pool after each method call.

6. Add the CanBePooled override to the StuffCache class

a. Add the code shown below:

```
protected override bool CanBePooled()
{
        return true;
}
```

When your object is deactivated, EnterpriseServices calls CanBePooled to determine if it is safe to return your object to the pool or if the garbage collector should dispose of the object.

7. Run the program

a. Select **Debug** | **Start Without Debugging**.

If you check the output pane in Visual Studio, you will notice that this time the StuffCache constructor is called only once. You will also notice that the execution is now significantly faster.

Conclusion

Object pooling can offer significant performance gains by allowing you to cache expensive-to-obtain resources or to create resources in a pooled object that can be used by many clients over a period of time.

Lab Summary

In this lab you performed the following exercises.

- Building a Class Library and Test Application
- Using Enterprise Services
- Using AutoComplete
- Using Object Pooling and Just-in-Time Activation

In this lab, you learned how to create an Enterprise Service application, and used some of the services that Enterprise Services provides, viz. Transaction, Object Pooling and Just-in-Time Activation.