msdn training

Module 12: Exploring Future Learning

Contents

Microsoft

Instructor Notes

Presentation:
45 minutes

Lab:
00 minutes

The module provides the students with pointers to some of the resources that are available for further study of advanced topics.

After completing this module, students will be able to:

- Locate resources for information about additional C# features.
- Use those resources to further develop any of the projects that they started earlier in this course.

Required materials

To teach this module, you need the following materials:

- Microsoft® PowerPoint® file 2609A_12.ppt
- Module 12, “Exploring Future Learning”

Preparation tasks

To prepare for this module, read all of the materials for this module.

How to Teach This Module

This section contains information that will help you to teach this module.

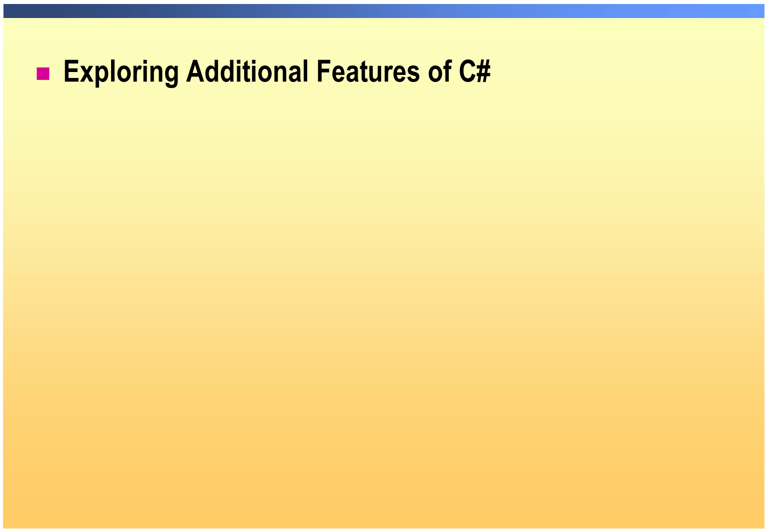
The intention of this module is to provide students with pointers to resources that are available for future study of advanced topics that are related to programming in C# in the Microsoft .NET development environment. For this reason, the content for each topic is limited to a brief introduction to advanced learning for the topic, and the focus is on pointing the students to the resources rather than teaching them the content.

Furthermore, there are no practices, demonstrations, labs, or review questions for this module.

Lesson: Exploring Additional Features of C#

All topics in this lesson are high-level overviews of advanced topics. Rather than teach that content, emphasize to the students that resources for self-paced future study are identified for each topic.

Overview



- **Exploring Additional Features of C#**

Introduction

This module provides an opportunity for you to explore some of the more advanced capabilities of C#, to practice the knowledge and skills that you acquired during the course, and to discuss your questions as a group.

Objectives

After completing this module, you will be able to:

- Locate resources for information about additional C# features.
- Use those resources to further develop any of the projects that you started earlier in this course.

Lesson: Exploring Additional Features of C#

- The C# Preprocessor
- Structs
- C# Threads and Threading
- Attributes and Reflection
- .NET Framework Security
- Interoperability
- .NET Remoting

Introduction

This lesson presents some additional features of C# that may interest you as you further develop your Microsoft® .NET-based projects. These advanced features are covered only briefly in this lesson. The focus of this lesson is to identify resources that you can use for further study and to provide an opportunity for you to develop a project that you started earlier in this course.

Lesson objectives

After completing this lesson, you will be able to:

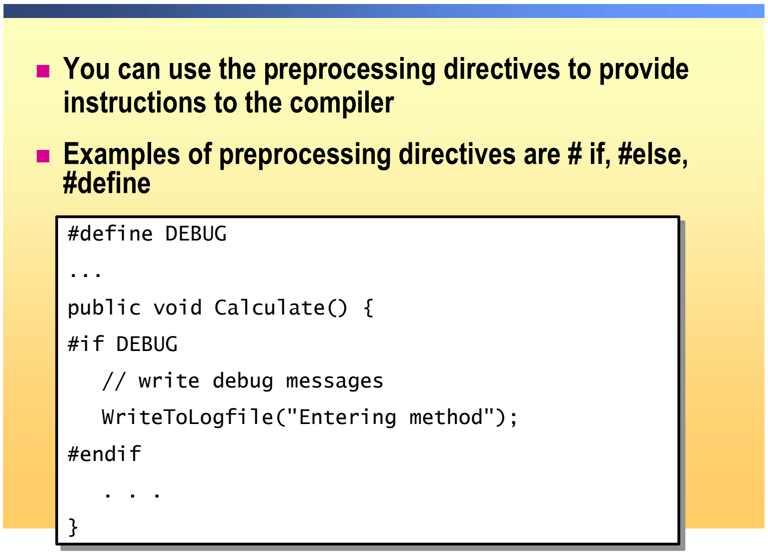
- Locate resources for information about advanced features of C#.
- Use those resources to further develop any of the projects that you started in the practices or labs of this course.

Lesson agenda

This lesson includes the following topics:

- The C# Preprocessor
- Structs
- C# Threads and Threading
- Attributes and Reflection
- .NET Framework Security
- Interoperability
- .NET Remoting

The C# Preprocessor



- You can use the preprocessing directives to provide instructions to the compiler
- Examples of preprocessing directives are # if, #else, #define

```
#define DEBUG
...
public void Calculate() {
#if DEBUG
   // write debug messages
   WriteToLogfile("Entering method");
#endif
   . . .
}
```

Introduction

Before your code is actually compiled, a *preprocessor* runs, examines, and prepares your program for the compiler. You can use the preprocessor to control the code that the compiler receives. In some circumstances, such as debugging, you may want to compile extra debugging statements into your program. However, you do not want these statements in your finished application. You can use preprocessing directives to cause the preprocessor to compile sections of code only under certain circumstances. This is called *conditional compilation*.

Preprocessing directives

Preprocessing directives provide the ability to conditionally skip sections of source files, to report error and warning conditions, and to delineate distinct regions of source code.

Preprocessing in C#

C# is designed to avoid the need for include files, because classes do not require a separate definition file. For this reason, the C# preprocessor is less important than it is in C or C++. Items that typically occur in header files, such as macros, are also eliminated to provide for simpler code maintenance and speedy compilation.

A main function of the preprocessor in C# is to enable conditional compilation.

Directive syntax

A preprocessing directive always occupies a separate line of source code and always begins with a pound sign (#) character and a preprocessing directive name. White space may occur before the # character and between the # character and the directive name.

Commonly used directives

The following table describes some of the preprocessing directives that are available for C#:

Directive	Description
#if...#endif	The **#if** directive allows you to begin a conditional directive, test a symbol or symbols to see if they evaluate to **true**. If they do evaluate to **true**, the compiler evaluates all of the code between the **#if** and the next directive. The **#endif** directive terminates the scope of the **#if** directive.
#else	The **#else** directive allows you to create a compound conditional directive, so that if none of the expressions in the preceding **#if** or (optional) **#elif** directives evaluate to **true**, the compiler evaluates all code between **#else** and the subsequent **#endif**.
#define	The **#define** directive allows you to define a symbol, such that, by using the symbol as the expression passed to the **#if** directive, the expression evaluates to **true**.
#region...#endregion	The #**region** and #**endregion** directives allow you to define areas of code that can be collapsed and hidden under a label. The development environment uses this directive to hide the code that it generates automatically.

Example

In the following example, the call to the **WriteToLogfile** method is compiled only when "#define DEBUG" appears as a line in the code.

```
#define DEBUG
...
public void Calculate() {
#if DEBUG
  // write debug messages
  WriteToLogfile("Entering method");
#endif
  . . .
}
```

Resources

For further information about the C# preprocessor, see "2.5 Preprocessing Directives" in the *Microsoft .NET Framework Software Development Kit (SDK)*.

Structs

- **A *struct* is a value type that can contain constructors, constants, fields, methods, properties, indexers, operators, and nested types**
- **Data structures suitable for use with structs**
    - Contain a small number of data members
    - Do not require use of inheritance
    - Can be implemented using value semantics
- **Structs vs classes**
    - *Classes* are reference types
    - *Structs* are value types

Introduction

When you create a class, you create a new reference type. You can also create new value types called structs, which are similar to classes but do not support class functionality such as inheritance, although they can implement interfaces.

Definition

A *struct* type is a value type that can contain constructors, constants, fields, methods, properties, indexers, operators, and nested types. Examples of structs are complex numbers, points in a coordinate system, or key-value pairs in a dictionary.

Data structures suitable for use with structs

Structs are particularly useful for small data structures that have value semantics. You should use structs for types that:

- Contain a small number of data members.
- Do not require the use of inheritance.
- Use value semantics, for example, where you expect an assignment to copy a value instead of a reference.

Structs vs. classes

Structs are similar to classes in that they are types that can contain properties and methods. Unlike classes, structs are value types and do not require heap allocation. An instance of a struct directly contains the data of the struct, whereas an instance of a class contains a reference to the data.

- Structs are value types.
- Classes are reference types.

Resources

For further information about structs, see the following resources:

- "Structs Tutorial" in the *C# Programmers Reference* in the online Help.
- "**System.Data.SqlTypes** namespace" in the *.NET Framework SDK*, which includes an excellent example of data types that are implemented as structs in the .NET Framework.

C# Threads and Threading

- ***Threads*** **are the basic unit to which an operating system allocates processor time**
- **Advantages**
  - Multiple threads increase responsiveness to the user and simultaneously process the data necessary to complete the task
- **Disadvantages**
  - Use as few threads as possible to minimize the use of operating-system resources and improve performance
- **Threading features**
  - The C# language provides the **lock** statement
  - The .NET Framework provides classes in the **System.Threading** namespace

Introduction

Sometimes you may want to run parts of your application in a different process, or thread.

Definition

A *thread* is the basic unit to which an operating system allocates processor time. More than one thread at a time can execute code inside that process.

Advantages and disadvantages

There are both advantages and disadvantages to using threading:

- Advantages

  Using more than one thread is the most powerful technique available to simultaneously increase responsiveness to the user and process the data necessary to complete the job.

- Disadvantages

  It is recommended that you use as few threads as possible, thereby minimizing the use of operating-system resources and improving performance. Threading also has resource requirements and potential conflicts that you must consider when you design your application.

Example

For example, when you print a large document, you do not want your application to stop responding to the user as it processes the print job. So, you create a new thread that handles the printing work while your main application continues to be available to the user.

Threading features

C# and the .NET Framework provide features that allow different threads in a program to start and to communicate with each other.

- The C# language provides the **lock** statement that allows you to manage resource conflicts.
- The .NET Framework provides classes in the **System.Threading** namespace that support threading and interthread communication.

Resources

For more information about threads and threading, see the following:

Books

If you are new to threads and threading, it is recommended that you complete an advanced course or read an appropriate book.

- "Assemblies, Threads and AppDomains" in *C# and the .NET Platform*, by Andrew Troelsen, Apress, 2001.
- "Multithreaded Programming" in *Inside C#*, by Tom Archer, Microsoft Press®, 2001.

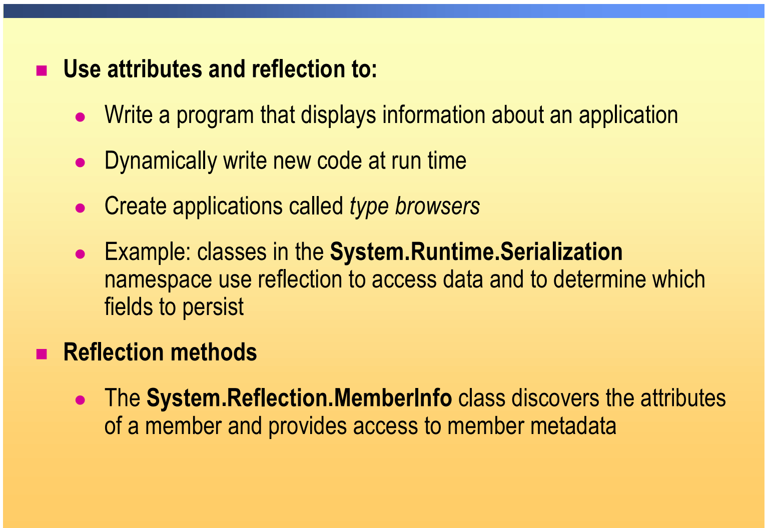
Online seminars

- "How to Build Multi-Threaded Application in .NET" online seminars, which are available under **Developer Resources** at http://www.microsoft.com/net/develop/. Eight seminars are available.

SDK

- The *.NET Framework SDK*. The SDK is a useful source of information and tutorials about threads and threading.

Attributes and Reflection



- **Use attributes and reflection to:**
  - Write a program that displays information about an application
  - Dynamically write new code at run time
  - Create applications called *type browsers*
  - Example: classes in the **System.Runtime.Serialization** namespace use reflection to access data and to determine which fields to persist
- **Reflection methods**
  - The **System.Reflection.MemberInfo** class discovers the attributes of a member and provides access to member metadata

Introduction

You use attributes to add metadata to your application, and you use reflection to read the metadata. Metadata in this context means data about the application. You can think of attributes as annotations that you insert into your application and reflection as the process that reads these annotations at run time.

Uses for attributes and reflection

You can use attributes and reflection to:

- Write a program that displays information about other applications, for example, for purposes of documenting those applications.

  The **Reflection** namespace contains the classes and interfaces that support this functionality.
- Dynamically write new code at run time, and then call that code.

  The classes of the **System.Reflection.Emit** namespace provide a specialized form of reflection that enables you to build types at run time.
- Create applications called *type browsers*, which enable users to select types and then view the information about those types.
- The classes in the **System.Runtime.Serialization** namespace use reflection to access data and to determine which fields to persist.

Reflection methods

The main reflection methods to query attributes are contained in the **System.Reflection.MemberInfo** class, which discovers the attributes of a member and provides access to member metadata.

The following example demonstrates the basic way of using reflection to obtain access to attributes:

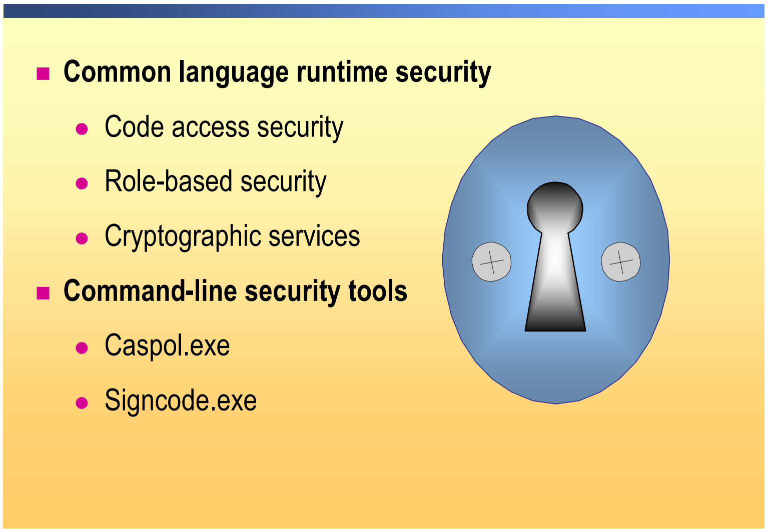
```
class MainClass {
  public static void Main() {
      System.Reflection.MemberInfo info = typeof(SomeClass);
      object[] attributes = info.GetCustomAttributes(true);
      for (int i = 0; i < attributes.Length; i ++) {
          System.Console.WriteLine(attributes[i]);
      }
  }
}
```

Resources

For more information about attributes and reflection, see:

- Any C# language book, most of which discuss this topic. For example, *Inside C#*, by Tom Archer, Microsoft Press, 2001.
- “Attributes Tutorial” in the *.NET Framework SDK*.

.NET Framework Security



- **Common language runtime security**
  - Code access security
  - Role-based security
  - Cryptographic services
- **Command-line security tools**
  - Caspol.exe
  - Signcode.exe

Introduction

As a programmer, you must consider the security of the data that you manage. You are at risk when you open a database, and Web-based applications are a clear target for malicious users. Security checks ensure that a piece of code has the credentials to access certain resources.

Common language runtime security

Although it is unnecessary for most application developers to do any special work to gain the advantages of the .NET Framework security system, it is useful to have knowledge of the classes and services that the .NET Framework provides.

The classes and services that the common language runtime and .NET Framework provide also enable system administrators to customize the access that code has to protected resources. Additionally, the runtime and the .NET Framework provide classes and services that facilitate the use of cryptography and role-based security.

Code access security

The .NET Framework provides a security mechanism called *code access security*. All managed code that targets the common language runtime receives the benefits of code access security, even if that code does not make a single code access security call.

The code access security mechanism:

- Protects computer systems from malicious mobile code.
- Allows code from unknown origins to run safely.

Tip For more information about .NET Framework code access security, see “Code Access Security” in the *.NET Framework Developer’s Guide*.

Role-based security

The runtime provides support for role-based authorization based on a Microsoft Windows® account or a custom identity. After you define identity (the user) and principal objects (the security context for that user), you can perform various security checks against them.

Tip For more information about .NET Framework role-based security, see “Role-Based Security” in the *.NET Framework Developer’s Guide*.

Cryptographic services

The classes in the .NET Framework **Cryptography** namespace manage many details of cryptography for you. You do not need to be an expert in cryptography to use these classes. When you create a new instance of one of the encryption algorithm classes, keys are auto-generated for ease of use, and default properties are always as safe and secure as possible.

Tip For more information about .NET Framework cryptographic services, see “Cryptographic Services” in *the .NET Framework Developer’s Guide*.

Command-line security tools

The .NET Framework SDK supplies command-line tools that help you perform security-related tasks and test your components and applications before you deploy them. Some of those tools include:

- *Caspol.exe*. The policy tool for code access security enables you to view and configure security policy.
- *Signcode.exe*. This file-signing tool signs a portable executable (PE) file with requested permissions, giving you more control over the security restrictions that are placed on your components.

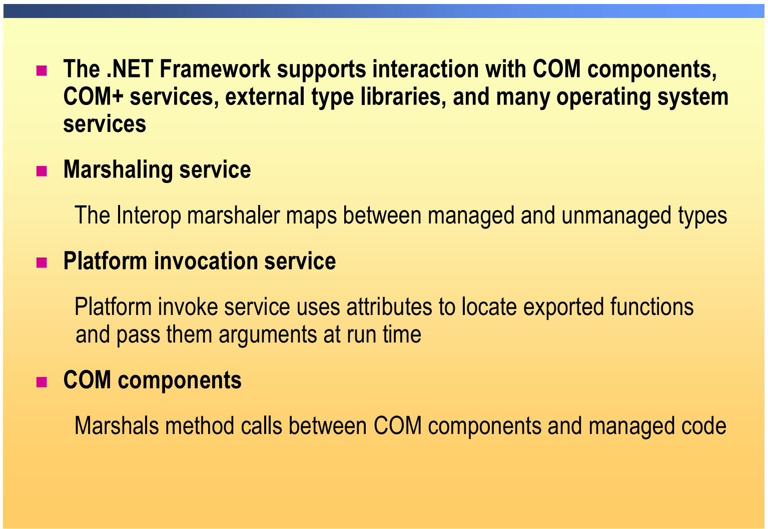
Tip For more information about .NET Framework security tools, see “Security Tools” in the *.NET Framework Developer’s Guide*.

Resources

For more information about making your code secure, see the following:

- Course 2350, *Securing and Deploying Microsoft .NET Assemblies.*
- *Writing Secure Code*, by Michael Howard and David LeBlanc. Microsoft Press, 2001.
- The *.NET Framework SDK*, which provides a tutorial on security. For further information, see the C# Programmer’s reference section for the security tutorial.

Interoperability



- **The .NET Framework supports interaction with COM components, COM+ services, external type libraries, and many operating system services**
- **Marshaling service**

  The Interop marshaler maps between managed and unmanaged types
- **Platform invocation service**

  Platform invoke service uses attributes to locate exported functions and pass them arguments at run time
- **COM components**

  Marshals method calls between COM components and managed code

Introduction

The .NET Framework supports interaction with unmanaged code, such as COM components, COM+ services, external type libraries, and many operating system services. Data types, method signatures, and error-handling mechanisms vary between managed and unmanaged object models. To simplify interoperation between the .NET Framework components and unmanaged code, the common language runtime conceals the differences in these object models from both clients and servers.

Marshaling service

Most data types have common representations in both managed and unmanaged memory. The interop marshaler handles these types for you. Other types can be ambiguous or not represented at all in managed memory. Marshaling occurs whenever the caller and recipient cannot operate on the same instance of data. The interop marshaler makes it possible for both the caller and recipient to appear to be operating on the same data even though the caller and recipient have their own copy of the data. You can supply explicit instructions to the interop marshaler about how it is to marshal an ambiguous type.

Platform invocation service

You can call C functions in dynamic-link libraries (DLLs) through a feature called Platform invoke. *Platform invoke* is a service that uses attributes to locate exported functions and pass arguments to them at run time. This service enables managed code to call unmanaged functions that are implemented in DLLs, such as those in the Microsoft Win32® application programming interface (API). The Platform invoke feature uses interop marshaling to pass method parameters and return values between managed code and the unmanaged library.

COM components

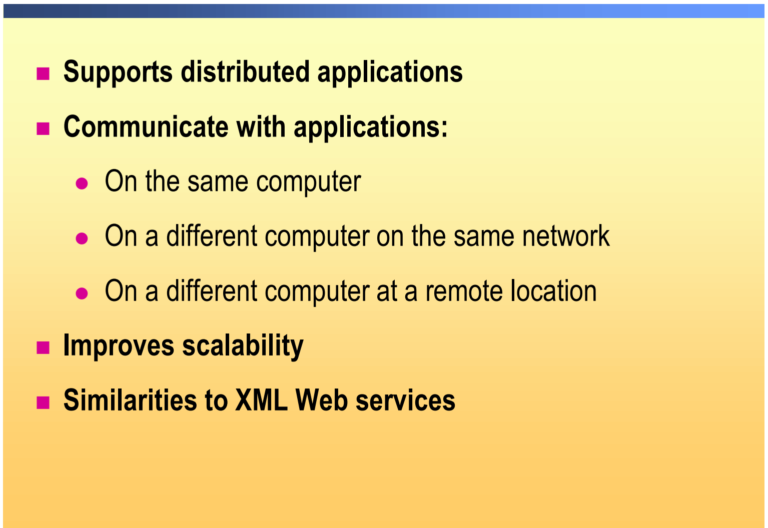
A .NET Framework application that must support a COM component cannot directly consume the functionality that is exposed by that component. Instead, it must access the functionality by using a proxy class, sometimes called a *wrapper*. Although a utility (Tlbimp.exe) is provided to help create this class, this utility does not remove the requirement for manual coding when you require detailed control of the component.

Resources

For more information about interoperability, see the following:

- Course 2571: *Application Upgrade and Interoperability Using Microsoft Visual Studio® .NET.*
- Any C# language book that discusses this topic. For example, *Inside C#,* by Tom Archer, Microsoft Press, 2001.

.NET Remoting



- **Supports distributed applications**
- **Communicate with applications:**
  - On the same computer
  - On a different computer on the same network
  - On a different computer at a remote location
- **Improves scalability**
- **Similarities to XML Web services**

Introduction

You can use .NET Remoting to enable different applications to communicate with one another, whether those applications reside on the same computer, on different computers in the same local area network, or across the world in very different networks; even if the computers run different operating systems.

The **System.Runtime.Remoting** namespace provides classes to activate remote objects, send messages to and receive messages from remote objects, and much more.

You can use remoting to create distributed applications and to make your applications more scalable.

.NET Remoting vs. XML Web services

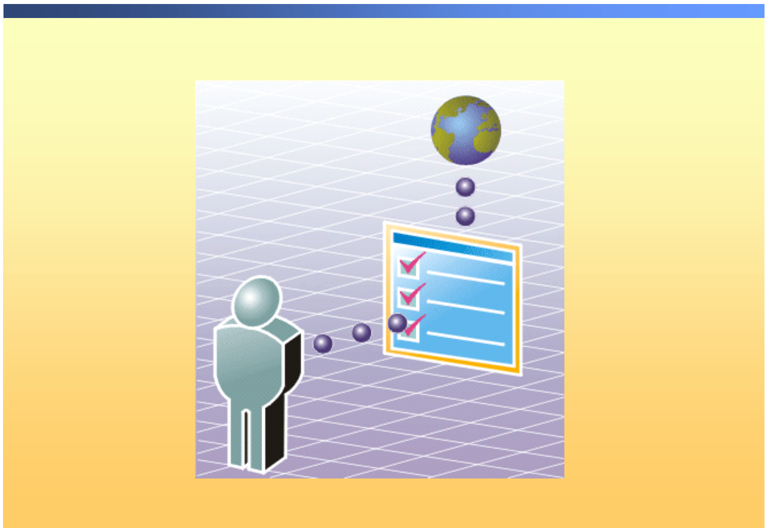
On the surface, .NET Remoting and XML Web services appear very similar to each other. In fact, XML Web services are built on the .NET Remoting infrastructure. However, as a general rule:

- .NET Remoting tends to be more appropriate for applications where the implementation of the applications at both ends of the conversation is under the control the same organization.
- XML Web services are more appropriate for applications where the client side of the service is likely to be outside the control of a particular organization, for example, a trading partner.

Resources

For more information about .NET Remoting, see Course 2349, *Programming the Microsoft .NET Framework with C#.*

Course Evaluation



Your evaluation of this course will help Microsoft understand the quality of your learning experience.

To complete a course evaluation, go to http://www.microsoft.com/traincert/coursesurvey.

Microsoft will keep your evaluation strictly confidential and will use your responses to improve your future learning experience.