

# Developing XML Web Services Using Microsoft<sup>®</sup> ASP.NET

## Delivery Guide

Course Number: 2524B

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, places or events 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, trademarks, 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.

© 2001-2002 Microsoft Corporation. All rights reserved.

Microsoft MS-DOS, Windows, Windows NT Active Directory, Authenticode, IntelliSense, FrontPage, Jscript, MSDN, PowerPoint, Visual C#, Visual Studio, Visual Basic, Windows NT, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

# Contents

## Introduction

Course Materials .....	2
Prerequisites .....	3
Course Outline .....	4
Setup .....	8
Microsoft Official Curriculum .....	9
Microsoft Certified Professional Program .....	11
Facilities .....	13

## Module 1: The Need for XML Web Services

Overview .....	1
Evolution of Distributed Applications .....	2
Problems with Traditional Distributed Applications .....	4
Introducing XML Web Services .....	14
The Web Technology Stack and .NET .....	16
The .NET Alternatives to XML Web Services .....	18
Common XML Web Service Scenarios .....	20
Review .....	22

## Module 2: XML Web Service Architectures

Overview .....	1
Service-Oriented Architecture .....	2
XML Web Services Architectures and Service-Oriented Architecture .....	4
Roles in an XML Web Services Architecture .....	8
The XML Web Services Programming Model .....	16
Review .....	18

## Module 3: The Underlying Technologies of XML Web Services

Overview .....	1
HTTP Fundamentals .....	2
Using HTTP with the .NET Framework .....	8
XML Essentials .....	17
XML Serialization in the .NET Framework .....	26
SOAP Fundamentals .....	29
Using SOAP with the .NET Framework .....	36
Lab 3.1: Issuing HTTP and SOAP Requests Using the .NET Framework .....	45
Review .....	54

## Module 4: Consuming XML Web Services

Overview .....	1
WSDL Documents .....	2
XML Web Service Discovery .....	8
XML Web Service Proxies .....	19
Implementing an XML Web Service Consumer Using Visual Studio .NET .....	27
Lab 4.1: Implementing an XML Web Service Consumer Using Visual Studio .NET .....	34
Review .....	43

**Module 5: Implementing a Simple XML Web Service**

Overview .....	1
Creating an XML Web Service Project .....	2
Implementing XML Web Service Methods .....	11
Managing State in an ASP.NET XML Web Service .....	33
Debugging XML Web Services .....	42
Lab 5.1: Implementing a Simple XML Web Service .....	61
Review .....	77

**Module 6: Publishing and Deploying XML Web Services**

Overview .....	1
Overview of UDDI .....	2
Publishing an XML Web Service .....	16
Finding an XML Web Service .....	21
Publishing an XML Web Service on an Intranet .....	24
Configuring an XML Web Service .....	26
Lab 6.1: Publishing and Finding Web Services in a UDDI Registry .....	29
Review .....	39

**Module 7: Securing XML Web Services**

Overview .....	1
Overview of Security .....	2
Built-In Authentication .....	10
Custom Authentication: SOAP Headers .....	18
Authorization: Role-Based Security .....	25
Authentication and Authorization with HttpModules .....	34
Authorization: Code Access Security .....	39
Encryption .....	46
Lab 7.1: Securing XML Web Services .....	54
Review .....	70
Course Evaluation .....	72

**Module 8: Designing XML Web Services**

Overview .....	1
Data Type Constraints .....	2
Performance .....	11
Lab 8.1: Implementing Caching in an XML Web Service .....	28
Reliability .....	33
Versioning .....	37
HTML Screen Scraping XML Web Services .....	39
Aggregating XML Web Services .....	47
Demonstration: Example of an Aggregated XML Web Service .....	52
Lab 8.2: Implementing an Aggregated XML Web Service .....	53
Review .....	67

**Module 9: Global XML Web Services Architecture**

Overview .....	1
Introduction to GXA .....	2
Routing and Referral .....	8
Security and License .....	16
Review .....	19
Course Evaluation .....	20

## About This Course

This section provides you with a brief description of the course, audience, suggested prerequisites, and course objectives.

### Description

This three-day instructor-led course teaches experienced software developers how to use XML Web services in solving common problems in the distributed application domain. This course teaches developers how to build, deploy, locate, and consume XML Web services.

### Audience

This course is designed for experienced software developers who have previously built component-based applications.

### Student Prerequisites

This course requires that students meet the requirements listed in the following knowledge and skills matrix.

Prerequisites	Understand Concepts		Written simple applications		Written real-world applications	
	Preferred	Required	Preferred	Required	Preferred	Required
Familiarity with C#				✓		
Programming in C++, Java, or Microsoft® Visual Basic®						✓
Familiarity with Extensible Markup Language (XML)		✓				

### Course Objectives

After completing this course, the student will be able to:

- Explain how XML Web services emerged as a solution to the problems with traditional approaches to designing distributed applications.
- Describe the architecture of an XML Web services-based solution.
- Explain how to use the Microsoft .NET Framework to implement XML Web services.
- Implement an XML Web service consumer by using Microsoft Visual Studio® .NET.
- Implement a simple XML Web service by using Microsoft Visual Studio .NET.
- Publish and deploy an XML Web service.
- Secure a XML Web service.
- Evaluate the trade-offs and issues that are involved in designing a real-world XML Web service.
- Implement nonstandard XML Web services such as Hypertext Markup Language (HTML) screen scraping and aggregating XML Web services.
- Describe the Global XML Architecture and explain how to design XML Web services to anticipate the new services.

## Course Timing

The following schedule is an estimate of the course timing. Your timing may vary.

### Day 1

Start	End	Module
9:00	9:20	Introduction
9:20	10:20	Module 1: The Need for XML Web Services
10:20	10:30	Break
10:30	11:30	Module 2: XML Web Service Architecture
11:30	12:15	Lunch
12:15	2:15	Module 3: The Underlying Technologies of XML Web Services
2:15	2:25	Break
2:25	3:10	Lab 3.1: Issuing HTTP and SOAP Requests Using the .NET Framework
3:10	5:10	Module 4: Consuming XML Web Services

### Day 2

Start	End	Module
9:00	10:15	Lab 4.1: Implementing an XML Web Service Consumer Using Visual Studio .NET
10:15	10:25	Break
10:25	12:00	Module 5: Implementing a Simple XML Web Service
12:00	12:45	Lunch
12:45	1:40	Module 5: Implementing a Simple XML Web Service ( <i>continued</i> )
1:40	2:30	Lab 5.1: Implementing a Simple XML Web Service
2:30	2:40	Break
2:40	3:05	Lab 5.1: Implementing a Simple XML Web Service ( <i>continued</i> )
3:05	4:35	Module 6: Publishing and Deploying XML Web Services
4:35	5:05	Lab 6.1: Publishing and Finding XML Web Services in a UDDI Registry

**Day 3**

Start	End	Module
9:00	9:30	Lab 6.1: Publishing and Finding XML Web Services in a UDDI Registry ( <i>continued</i> )
9:30	10:30	Module 7: Securing XML Web Services
10:30	10:40	Break
10:40	11:40	Module 7: Securing XML Web Services ( <i>continued</i> )
11:40	12:25	Lunch
12:25	1:25	Lab 7.1: Securing XML Web Services
1:25	2:25	Module 8: Designing XML Web Services
2:25	2:35	Break
2:35	3:15	Lab 8.1: Implementing Caching in an XML Web Service
3:15	3:45	Module 8: Designing XML Web Services ( <i>continued</i> )
3:45	5:15	Lab 8.2: Implementing an Aggregated XML Web Service
5:15	5:45	Module 9: Global XML Web Services Architecture

## Trainer Materials Compact Disc Contents

The Trainer Materials compact disc contains the following files and folders:

- *Autorun.exe*. When the compact disc is inserted into the CD-ROM drive, or when you double-click the **Autorun.exe** file, this file opens the compact disc and allows you to browse the Student Materials or Trainer Materials compact disc.
- *Autorun.inf*. When the compact disc is inserted into the compact disc drive, this file opens **Autorun.exe**.
- *Default.htm*. This file opens the Trainer Materials Web page.
- *Readme.txt*. This file explains how to install the software for viewing the Trainer Materials compact disc and its contents and how to open the Trainer Materials Web page.
- *2524B\_ms.doc*. This file is the Manual Classroom Setup Guide. It contains the steps for manually installing the classroom computers.
- *2524B\_sg.doc*. This file is the Automated Classroom Setup Guide. It contains a description of classroom requirements, classroom configuration, instructions for using the automated classroom setup scripts, and the Classroom Setup Checklist.
- *Powerpnt*. This folder contains the Microsoft PowerPoint® slides that are used in this course.
- *Pptview*. This folder contains the Microsoft PowerPoint Viewer 97, which can be used to display the PowerPoint slides if Microsoft PowerPoint 2002 is not available. Do not use this version in the classroom.
- *Setup*. This folder contains the files that install the course and related software to computers in a classroom setting.
- *StudentCD*. This folder contains the Web page that provides students with links to resources pertaining to this course, including additional reading, review and lab answers, lab files, multimedia presentations, and course-related Web sites.
- *Tools*. This folder contains files and utilities used to complete the setup of the instructor computer.
- *Webfiles*. This folder contains the files that are required to view the course Web page. To open the Web page, open Microsoft Windows® Explorer, and in the root directory of the compact disc, double-click **Default.htm** or **Autorun.exe**.



## Student Materials Compact Disc Contents

The Student Materials compact disc contains the following files and folders:

- *Autorun.exe*. When the compact disc is inserted into the CD-ROM drive, or when you double-click the **Autorun.exe** file, this file opens the compact disc and allows you to browse the Student Materials compact disc or install Internet Explorer.
- *Autorun.inf*. When the compact disc is inserted into the compact disc drive, this file opens **Autorun.exe**.
- *Default.htm*. This file opens the Student Materials Web page. It provides resources pertaining to this course, including additional reading, review and lab answers, lab files, multimedia presentations, and course-related Web sites.
- *Readme.txt*. This file explains how to install the software for viewing the Student Materials compact disc and its contents and how to open the Student Materials Web page.
- *2524B\_ms.doc*. This file is the Manual Classroom Setup Guide. It contains a description of classroom requirements, classroom setup instructions, and the classroom configuration.
- *Database*. This folder contains databases used in the course.
- *Democode*. This folder contains demonstration code.
- *Flash*. This folder contains the installer for the Macromedia Flash 5.0 browser plug-in.
- *Fonts*. This folder contains fonts that are required to view the Microsoft PowerPoint presentation and Web-based materials.
- *Labfiles*. This folder contains files that are used in the hands-on labs. These files may be used to prepare the student computers for the hands-on labs.
- *Mplayer*. This folder contains the setup file to install Microsoft Windows Media™ Player.
- *UDDI*. This folder contains files that are used to initialize the Universal Description, Discovery, and Integration (UDDI) registry. The folder also contains files that are used to install the UDDI Services and the UDDI SDK, and to register the XML Web services that UDDI registry uses in this course.
- *Webfiles*. This folder contains the files that are required to view the course Web page. To open the Web page, open Windows Explorer, and in the root directory of the compact disc, double-click **Default.htm** or **Autorun.exe**.
- *Wordview*. This folder contains the Word Viewer that is used to view any Word document (.doc) files that are included on the compact disc.

## Document Conventions

The following conventions are used in course materials to distinguish elements of the text.

Convention	Use
<b>Bold</b>	Represents commands, command options, and syntax that must be typed exactly as shown. It also indicates commands on menus and buttons, dialog box titles and options, and icon and menu names.
<i>Italic</i>	In syntax statements or descriptive text, indicates argument names or placeholders for variable information. Italic is also used for introducing new terms, for book titles, and for emphasis in the text.
Title Capitals	Indicate domain names, user names, computer names, directory names, and folder and file names, except when specifically referring to case-sensitive names. Unless otherwise indicated, you can use lowercase letters when you type a directory name or file name in a dialog box or at a command prompt.
ALL CAPITALS	Indicate the names of keys, key sequences, and key combinations—for example, ALT+SPACEBAR.
Monospace	Represents code samples or examples of screen text.
[ ]	In syntax statements, enclose optional items. For example, <i>[filename]</i> in command syntax indicates that you can choose to type a file name with the command. Type only the information within the brackets, not the brackets themselves.
{ }	In syntax statements, enclose required items. Type only the information within the braces, not the braces themselves.
	In syntax statements, separates an either/or choice.
►	Indicates a procedure with sequential steps.
...	In syntax statements, specifies that the preceding item may be repeated.
.	Represents an omitted portion of a code sample.
.	
.	

## Introduction

### Contents

Introduction	1
Course Materials	2
Prerequisites	3
Course Outline	4
Setup	8
Microsoft Official Curriculum	9
Microsoft Certified Professional Program	11
Facilities	13



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place 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, trademarks, 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.

© 2001–2002 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Active Directory, Authenticode, IntelliSense, FrontPage, Jscript, MSDN, PowerPoint, Visual C#, Visual Studio, Visual Basic, Windows NT, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

# Instructor Notes

**Presentation:**  
**20 Minutes**

The Introduction module provides students with an overview of the course content, materials, and logistics for Course 2524B, *Developing XML Web Services Using Microsoft® ASP.NET*.

**Required Materials**

To teach this course, you need the following materials:

- Delivery Guide
- Trainer Materials compact disc

**Preparation Tasks**

To prepare for this course, you must:

- Complete the Course Preparation Checklist that is included with the trainer course materials.
- Review all contents on the Trainer Materials compact disc.
- Review the Trainer Preparation Presentation on the Trainer Materials compact disc.
- Read the Trainer Delivery Guide for the course.
- Read the Instructor Notes that precede each module. The Instructor Notes contain preparation suggestions for each module.
- Read any recommended documents that are mentioned in the Instructor Notes section for each module.
- Practice using the Microsoft products and tools that are used in this course.
- Practice presenting each module and demonstration.
- Review the Classroom Setup Guide.
- Practice each lab. Anticipate the questions that students may have.
- Identify the key points for each topic, demonstration, and lab.
- Identify how each demonstration and lab supports the module topics and reinforces the module objectives.
- Identify examples, analogies, demonstrations, and additional delivery strategies that will help to clarify module topics for students.
- Identify the information that students need to complete each lab successfully.
- Note any problems that you may encounter during a demonstration or lab and determine a course of action for resolving them in the classroom.
- Identify additional preparation that is required to ensure the success of each demonstration and lab.
- Identify ways to customize a demonstration or lab to provide a more meaningful learning experience for your specific audience.

## How to Teach This Module

	<p>This section contains information that will help you to teach this module.</p>
<b>Introduction</b>	<p>Welcome students to the course and introduce yourself. Provide a brief overview of your background to establish credibility.</p> <p>Ask students to introduce themselves and provide their background, product experience, and expectations of the course.</p> <p>Record student expectations on a whiteboard or flip chart that you can reference later in class.</p>
<b>Course Materials</b>	<p>Tell students that everything they will need for this course is provided at their desk.</p> <p>Have students write their names on both sides of the name card.</p> <p>Describe the contents of the student workbook and the Student Materials compact disc.</p> <p>Tell students where they can send comments and feedback on this course.</p> <p>Demonstrate how to open the Web page that is provided on the Student Materials compact disc by double-clicking <b>Autorun.exe</b> or <b>Default.htm</b> in the StudentCD folder on the Trainer Materials compact disc.</p>
<b>Prerequisites</b>	<p>Describe the prerequisites for this course. This is an opportunity for you to identify students who may not have the appropriate background or experience to attend this course.</p>
<b>Course Outline</b>	<p>Briefly describe each module and what students will learn. Be careful not to go into too much detail because the course is introduced in detail in Module 1.</p> <p>Explain how this course will meet students' expectations by relating the information that is covered in individual modules to their expectations.</p>
<b>Setup</b>	<p>Describe any necessary setup information for the course, including course files and classroom configuration.</p>
<b>Microsoft Official Curriculum</b>	<p>Explain the Microsoft Official Curriculum (MOC) program and present the list of additional recommended courses.</p> <p>Refer students to the Microsoft Official Curriculum Web page at <a href="http://www.microsoft.com/traincert/training/">http://www.microsoft.com/traincert/training/</a> for information about curriculum paths.</p>
<b>Microsoft Certified Professional Program</b>	<p>Inform students about the Microsoft Certified Professional (MCP) program, any certification exams that are related to this course, and the various certification options.</p>
<b>Facilities</b>	<p>Explain the class hours, extended building hours for labs, parking, restroom location, meals, phones, message posting, and where smoking is or is not allowed.</p> <p>Let students know if your facility has Internet access that is available for them to use during class breaks.</p> <p>Also, make sure that the students are aware of the recycling program if one is available.</p>

# Introduction

- Name
- Company affiliation
- Title/function
- Job responsibility
- Distributed application/component-based application development experience
- Expectations for the course

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Course Materials

- Name card
- Student workbook
- Student Materials compact disc
- Course evaluation

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

The following materials are included with your kit:

- *Name card.* Write your name on both sides of the name card.
- *Student workbook.* The student workbook contains the material covered in class, in addition to the hands-on lab exercises.
- *Student Materials compact disc.* The Student Materials compact disc contains the Web page that provides you with links to resources pertaining to this course, including additional readings, review and lab answers, lab files, multimedia presentations, and course-related Web sites.

---

**Note** To open the Web page, insert the Student Materials compact disc into the CD-ROM drive, and then in the root directory of the compact disc, double-click **Autorun.exe** or **Default.htm**.

---

- *Course evaluation.* To provide feedback on the course, training facility, and instructor, you will have the opportunity to complete an online evaluation near the end of the course.

To provide additional comments or inquire about the Microsoft® Certified Professional program, send e-mail to [mcphelp@microsoft.com](mailto:mcphelp@microsoft.com).



# Prerequisites

Prerequisites	Understand Concepts		Written simple applications		Written real-world applications	
	Preferred	Required	Preferred	Required	Preferred	Required
Familiarity with C# or Microsoft Visual Basic®.NET				✓		
Programming in C++, Java, or Visual Basic						✓
Familiarity with XML		✓				

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

This course requires that you meet the requirements listed in the following knowledge and skills matrix.

Prerequisites	Understand Concepts		Written simple applications		Written real-world applications	
	Preferred	Required	Preferred	Required	Preferred	Required
Familiarity with C# or Microsoft Visual Basic® .NET				✓		
Programming in C++, Java, or Microsoft Visual Basic						✓
Familiarity with Extensible Markup Language (XML)		✓				

## Course Outline

- **Module 1: The Need for XML Web Services**
- **Module 2: XML Web Service Architectures**
- **Module 3: The Underlying Technologies of XML Web Services**
- **Module 4: Consuming XML Web Services**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Module 1, “The Need for XML Web Services,” introduces XML Web services, and discusses the problem space that they address. In this context, the evolution of distributed applications and the limitations of existing distributed application architectures are covered. After completing this module, you will be able to explain how XML Web services emerged as a solution to the problems with traditional approaches to designing distributed applications.

Module 2, “XML Web Service Architectures,” describes the architecture of an XML Web services-based solution. Service-oriented architecture is a conceptual architecture for distributed applications and this module explains how the XML Web service architecture is a type of service-oriented architecture. The roles of the various elements in the XML Web service architecture are also covered. After completing this module, you will be able to describe the architecture of an XML Web services-based solution.

Module 3, “The Underlying Technologies of XML Web Services,” discusses the three foundation technologies of XML Web services: Hypertext Transfer Protocol (HTTP), the Extensible Markup Language (XML), and the Simple Object Access Protocol (SOAP). This module also discusses the support that the Microsoft .NET Framework provides for using these technologies. The module provides hands-on experience with each of these technologies. After completing this module, you will be able to describe the underlying technologies of XML Web services and explain how to use the .NET Framework to communicate with XML Web services using these technologies.

Module 4, “Consuming XML Web Services,” is the first of the modules that discusses the implementation details of an XML Web service-based solution. This module specifically focuses on how to implement an XML Web service consumer to consume (use) XML Web services. Web Service consumers are implemented based on the service description documents of XML Web services. In this context, this module discusses the structure of a Web Service Description Language (WSDL) document and how to find XML Web services and their service descriptions at known endpoints by using Disco.exe. After completing this module, you will be able to implement an XML Web service consumer by using Microsoft Visual Studio® .NET.

## Course Outline (*continued*)

- **Module 5: Implementing a Simple XML Web Service**
- **Module 6: Publishing and Deploying XML Web Services**
- **Module 7: Securing XML Web Services**
- **Module 8: Designing XML Web Services**
- **Module 9: Global XML Web Services Architecture**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Module 5, “Implementing a Simple XML Web Service,” explains how to implement a simple XML Web service by using Microsoft ASP.NET. The module also explains how to manage state in ASP.NET XML Web services. Some of the techniques for debugging XML Web services are also covered. After completing this module, you will be able to implement a simple XML Web service by using Microsoft Visual Studio .NET.

Module 6, “Publishing and Deploying XML Web Services,” explains how to publish an XML Web service in a Universal Description, Discovery, and Integration (UDDI) registry to facilitate XML Web services discovery at unknown endpoints. This module covers both publishing and finding an XML Web service in a UDDI registry. The options for publishing an XML Web service on an intranet and the options for modifying the default configuration of an XML Web service are also discussed. After completing this module, you will be able to publish and deploy an XML Web service.

Module 7, “Securing XML Web Services,” describes how to secure XML Web services, specifically, how to provide authentication, authorization, and secure communication in XML Web services. In the context of authentication, this module covers the authentication mechanisms in Microsoft Internet Information Services (IIS) in addition to custom authentication mechanisms that use SOAP headers. In the context of authorization, the .NET Framework’s support for role-based security and code access security are covered. In the context of secure communication, this module covers how to encrypt the communications between an XML Web service and an XML Web service consumer by using SOAP extensions. After completing this module, you will be able to secure an XML Web service.

Module 8, “Designing XML Web Services,” examines some of the important issues that you need to consider when designing a real-world XML Web service. The issues discussed are related to datatype constraints, performance, reliability, versioning, deployment in Internet Service Provider (ISP) and Application Service Provider (ASP) scenarios, and aggregating XML Web services. The module also discusses Hypertext Markup Language (HTML) screen scraping as a pseudo-XML Web service. After completing this module, you will be able to evaluate the trade-offs and issues that are involved in designing a real-world XML Web Service.

Module 9, “Global XML Web Services Architecture,” describes the limitation of the current specifications that determine how XML Web services are built. This module describes some of the Global XML Web services Architecture (GXA) specifications and how to design XML Web services today that will anticipate the services that GXA will offer. After completing this module, you will be able to describe limitations inherent to the specifications with which today’s XML Web services are built, describe the upcoming GXA specifications and understand how to design XML Web services that anticipate and can leverage the features that GXA will offer when released.

# Setup

- Windows XP Professional
- Microsoft Windows .NET Server, Beta 3
- UDDI Services for .NET Server Beta 3
- Visual Studio .NET Enterprise Developer Edition
- Windows Component Update
- SQL Server 2000 Developer Edition
- UDDI SDK version 1.76
- Course Files
  - Labs
  - Demonstrations and code walkthroughs

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

The following software will be used in the classroom:

- Microsoft Windows® XP Professional
- Microsoft Windows .NET Server, Beta 3
- UDDI Services for .NET Server Beta 3
- Microsoft Visual Studio .NET Enterprise Developer Edition
- Microsoft Windows Component Update compact disc set
- Microsoft SQL Server™ 2000 Developer Edition
- Microsoft UDDI SDK version 1.76

## Labs

There are starter and solution files associated with the labs in this course. The starter files are located in the *<install folder>\Labfiles\<language>\Lab0x\Starter* folder and the solution files are in the *<install folder>\Labfiles\<language>\Lab0x\Solution* folder, where Lab0x reflects the current lab.

---

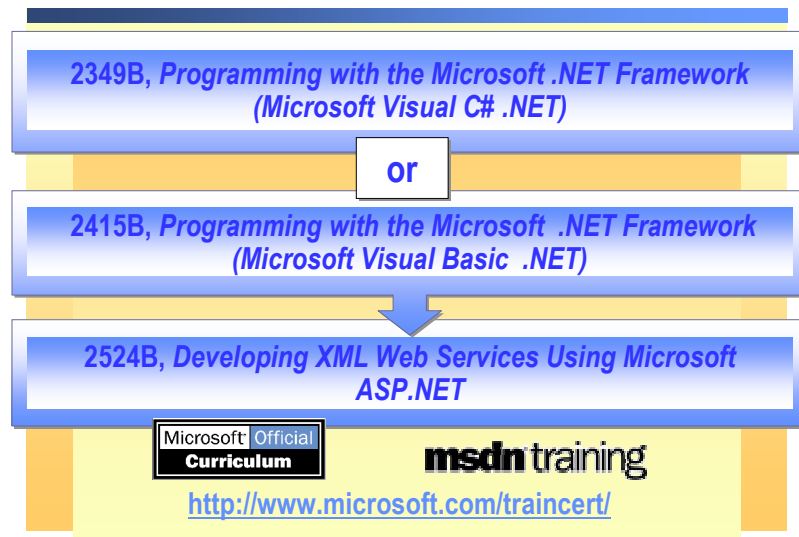
**Note** The labs in this course are based on a banking services scenario. A complete working solution of this scenario is located in the *<install folder>\Labfiles\<language>\WebServicesSolution* folder.

---

## Demonstrations and code walkthroughs

There are code files associated with the demonstrations and code walkthroughs in this course. These files are located in the *<install folder>\Democode\<language>\Mod0x* folder, where Mod0x reflects the current module.

## Microsoft Official Curriculum



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

Microsoft Training and Certification develops Microsoft Official Curriculum (MOC), including Microsoft MSDN® Training, for computer professionals who design, develop, support, implement, or manage solutions using Microsoft products and technologies. These courses provide comprehensive skills-based training in instructor-led and online formats.

### Additional recommended courses

Each course relates in some way to another course. A related course may be a prerequisite, a follow-up course in a recommended series, or a course that offers additional training.

In preparation for taking Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*, you can take either of the following courses:

- Course 2349B, Programming with the Microsoft .NET Framework (Microsoft Visual C#™ .NET)
- Course 2415B, Programming with the Microsoft .NET Framework (Microsoft Visual Basic .NET)

Course	Title and description
2349B	<p><i>Programming with the Microsoft .NET Framework (Microsoft Visual C# .NET)</i></p> <p>This course provides a hands-on tour of the Microsoft .NET Framework for C# developers. An overview of key concepts is followed by an in-depth tutorial on areas including the common type system, base class libraries, assemblies, delegates, and events, memory management, file and network I/O, serialization, and remoting. Examples and labs reinforce the knowledge that is needed to develop, deploy, and version Microsoft .NET Components.</p>
2415B	<p><i>Programming with the Microsoft .NET Framework (Microsoft Visual Basic .NET)</i></p> <p>This course provides developers with a hands-on tour of the Microsoft .NET Framework and tutorials about working with assemblies, versioning, the common type system, memory management, file and network I/O, serialization, remoting, and XML Web services.</p>

Other related courses may become available in the future, so for up-to-date information about recommended courses, visit the Training and Certification Web site.

**Microsoft Training and Certification information**

For more information, visit the Microsoft Training and Certification Web site at <http://www.microsoft.com/traincert/>.



# Microsoft Certified Professional Program



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

Microsoft Training and Certification offers a variety of certification credentials for developers and IT professionals. The Microsoft Certified Professional program is the leading certification program for validating your experience and skills, keeping you competitive in today's changing business environment.

## Related certification exams MCP certifications

The Microsoft Certified Professional program includes the following certifications.

### ■ MCSA on Microsoft Windows 2000

The Microsoft Certified Systems Administrator (MCSA) certification is designed for professionals who implement, manage, and troubleshoot existing network and system environments based on Microsoft Windows 2000 platforms, including the Windows .NET Server family.

Implementation responsibilities include installing and configuring parts of the systems. Management responsibilities include administering and supporting the systems.

### ■ MCSE on Microsoft Windows 2000

The Microsoft Certified Systems Engineer (MCSE) credential is the premier certification for professionals who analyze the business requirements and design and implement the infrastructure for business solutions based on the Microsoft Windows 2000 platform and Microsoft server software, including the Windows .NET Server family. Implementation responsibilities include installing, configuring, and troubleshooting network systems.

### ■ MCSD

The Microsoft Certified Solution Developer (MCSD) credential is the premier certification for professionals who design and develop leading-edge business solutions with Microsoft development tools, technologies, platforms, and the Microsoft Windows DNA architecture. The types of applications MCSDs can develop include desktop applications and multi-user, Web-based, N-tier, and transaction-based applications. The credential covers job tasks ranging from analyzing business requirements to maintaining solutions.

- **MCDBA on Microsoft SQL Server 2000**

The Microsoft Certified Database Administrator (MCDBA) credential is the premier certification for professionals who implement and administer Microsoft SQL Server databases. The certification is appropriate for individuals who derive physical database designs, develop logical data models, create physical databases, create data services by using Transact-SQL, manage and maintain databases, configure and manage security, monitor and optimize databases, and install and configure SQL Server.

- **MCP**

The Microsoft Certified Professional (MCP) credential is for individuals who have the skills to successfully implement a Microsoft product or technology as part of a business solution in an organization. Hands-on experience with the product is necessary to successfully achieve certification.

- **MCT**

Microsoft Certified Trainers (MCTs) demonstrate the instructional and technical skills that qualify them to deliver Microsoft Official Curriculum through Microsoft Certified Technical Education Centers (Microsoft CTECs).

**Certification requirements**

The certification requirements differ for each certification category and are specific to the products and job functions addressed by the certification. To become a Microsoft Certified Professional, you must pass rigorous certification exams that provide a valid and reliable measure of technical proficiency and expertise.

---

**For More Information** See the Microsoft Training and Certification Web site at <http://www.microsoft.com/traincert/>.

You can also send e-mail to [mcphelp@microsoft.com](mailto:mcphelp@microsoft.com) if you have specific certification questions.

---

**Acquiring the skills tested by an MCP exam**

Microsoft Official Curriculum (MOC) and MSDN Training Curriculum can help you develop the skills that you need to do your job. They also complement the experience that you gain while working with Microsoft products and technologies. However, no one-to-one correlation exists between MOC and MSDN Training courses and MCP exams. Microsoft does not expect or intend for the courses to be the sole preparation method for passing MCP exams. Practical product knowledge and experience is also necessary to pass the MCP exams.

To help prepare for the MCP exams, use the preparation guides that are available for each exam. Each Exam Preparation Guide contains exam-specific information, such as a list of the topics on which you will be tested. These guides are available on the Microsoft Training and Certification Web site at <http://www.microsoft.com/traincert/>.

# Facilities



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*



---

## Module 1: The Need for XML Web Services

### Contents

Overview	1
Evolution of Distributed Applications	2
Problems with Traditional Distributed Applications	4
Introducing XML Web Services	14
The Web Technology Stack and .NET	16
The .NET Alternatives to XML Web Services	18
Common XML Web Service Scenarios	20
Review	22



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, places or events 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, trademarks, 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.

© 2001–2002 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Active Directory, Authenticode, IntelliSense, FrontPage, Jscript, MSDN, PowerPoint, Visual C#, Visual Studio, Visual Basic, Windows NT, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

# Instructor Notes

**Presentation:**  
**60 minutes**

**Lab:**  
**00 minutes**

This module provides students with an understanding of the problem space that XML (Extensible Markup Language) Web services address. The module compares various approaches to implementing distributed applications. XML Web services are shown to be the natural evolution of distributed application architectures on the Internet. Because the XML Web services in this course are implemented by using Microsoft® ASP.NET and the Microsoft .NET Framework, alternate options for implementing distributed applications by using the .NET Framework are discussed to better define what kinds of solutions XML Web services are appropriate for.

After completing this module, students will be able to:

- Describe the evolution of distributed applications.
- Identify the problems with traditional distributed application architectures and technologies.
- Describe XML Web services and briefly explain how they address the design problems in traditional distributed applications.
- List the alternate options for distributed application development.
- Identify the kinds of scenarios where XML Web services are an appropriate solution.

**Required materials**

To teach this module, you need the Microsoft PowerPoint® file 2524B\_01.ppt.

**Preparation tasks**

To prepare for this module:

- Read all of the materials for this module.
- Read the topic “Designing Distributed Applications” in the Microsoft Visual Studio® .NET documentation in Microsoft MSDN®. Also, read all of the linked topics.

## How to Teach This Module

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

- **Evolution of Distributed Applications**

The students must understand how distributed applications have evolved from being islands of functionality into being service providers and building blocks for larger systems. Students also need to understand the importance of distributed applications.

- **Problems with Traditional Distributed Applications**

Begin this section by explaining some of the design considerations that are unique to distributed applications. Compare and contrast the remote procedure call (RPC) and message-based architectures for building distributed applications. Acknowledge that there are other distributed application architectures, but explain that the intent of this section is to understand the architectural issues, and not the specific pros and cons of each architectural pattern. Explain how the Web has provided a new environment in which distributed applications can be developed and define what some of the benefits and challenges of the Web are.

- **Introducing XML Web Services**

Briefly describe what XML Web services are. Emphasize the fact that the underlying technologies for XML Web services are Internet technologies. Explain how XML Web services are an evolution of existing distributed application architectures. Avoid an extensive discussion of the features of XML Web services because this will be covered throughout the rest of this course.

- **The Web Technology Stack and .NET**

Explain that the .NET Framework provides classes that map to each level in the technology stack. Explain the trade-offs in implementing a solution at various levels of the technology stack. Use the explanation of the trade-offs to guide the students to the conclusion that only in limited circumstances should they consider reimplementing higher levels of the technology stack. Tell the students that in most circumstances, they should take advantage of the productivity gains and robustness of the infrastructure that the .NET Framework provides.

- **The .NET Alternatives to XML Web Services**

Explain that the .NET Framework supports many patterns for building distributed applications, with XML Web services being just one type. Many students will be interested in .NET remoting solutions. Be sure to contrast the tighter coupling of .NET remoting solutions vs. the loose coupling of XML Web service solutions.

- **Common XML Web Service Scenarios**

Describe some common scenarios where XML Web services might be an appropriate solution. You are encouraged to share other scenarios as appropriate with your students.



# Overview

- Evolution of Distributed Applications
- Problems with Traditional Distributed Applications
- Introducing XML Web Services
- The Web Technology Stack and .NET
- The .NET Alternatives to XML Web Services
- Common XML Web Service Scenarios

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

To understand the importance of XML Web services, you need to understand the problem space that they address. Specifically, you need to be familiar with the evolution of distributed applications and the limitations of existing distributed application architectures.

This module begins by examining how the architecture and technologies related to distributed applications evolved. You will study the problems inherent in each of the existing distributed application architectures. Next, XML Web services and the role they play in the context of distributed application architectures are described. The module then goes on to describe the Web technology stack and the support that the Microsoft® .NET Framework provides for each of the technologies in the stack. Also, some of the .NET alternatives to XML Web services are briefly described. The module concludes with a discussion about some of the common scenarios in which it is appropriate to use XML Web services.

## Objectives

After completing this module, you will be able to:

- Describe the evolution of distributed applications.
- Identify the problems with traditional distributed application architectures and technologies.
- Describe XML Web services and briefly explain how they address the design problems in traditional distributed applications.
- List the alternate options for distributed application development.
- Identify the kinds of scenarios where XML Web services are an appropriate solution.

# Evolution of Distributed Applications

- What is a distributed application?
- Why do we need distributed applications?
- Distributed applications as service providers
- Distributed applications and the Web

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

Before the advent of the personal computer, it could be argued that the notion of distributed applications did not exist. Until that point, using a computer involved sitting in front of a terminal and interacting with a mainframe. Although the terminals could be spread across multiple buildings or even physically located off-site, there was a central computer that performed all of the processing and stored all of the data.

## What is a distributed application?

With the advent of the mini-computer and the personal computer, decentralizing both processing and data storage became desirable. However, even though the data processing and storage were no longer centralized, the application logically could still be a single application, by designing the application as a distributed application. A distributed application is an application whose processing requirements may be satisfied by multiple physical computers, and whose data may be stored in many physical locations, but whose logical function is not determined by the physical topology that is used to implement the application.

**Why do we need distributed applications?**

The driving forces behind the move to decentralize processing and data storage include:

- Cost of mainframes

One of the primary driving forces was the cost of mainframes. Not only was the initial investment cost beyond the reach of most companies, but having a single point of failure was a risk that most companies could not afford.

- Data ownership

An important factor behind decentralization was the politics of data ownership. Departments, divisions, geographic locations, or sites that owned the data did not like to delegate the responsibility of managing their data to some other central location.

- Security

Another important factor was security. For an organization, typically most of its data needs to be easily accessible. However, sensitive corporate data still must be secured. Catering to these two competing security requirements was much easier if the data could be physically segmented.

The preceding factors led to the emergence of a new application design pattern, which is known as distributed applications.

**Distributed applications as service providers**

With the emergence of the design pattern for the distributed application came the realization that the computer industry had not yet achieved its goal of reuse. Instead of viewing distributed applications as logically monolithic, it became useful to view the distributed components of an application as providers of services to a logical application. The concept of distributing functionality held the promise of reuse. Developers could use each of the distributed sets of functionality as a building block for much larger applications. There are significant problems in achieving this type of reuse. Some of these problems are covered later in this module, when the various architectures and technologies that are used to implement distributed applications are explained.

**Distributed applications and the Web**

Although the Internet had existed for more than twenty years, it was only in the mid-1990s that the possibility of the Internet providing significant infrastructure for building distributed applications was realized. Simple text-based protocols were developed as a primary means for communicating service requests and sending data on the Internet. The widespread adoption of such protocols made the Internet a viable platform for distributed applications. Instead of relying on competing and often proprietary technologies, Web standards would form the foundation for distributed applications for the Web.

# Problems with Traditional Distributed Applications

- Design Considerations for Distributed Applications
- RPC-Based Architectures
- Message-Based Architectures
- Web Standards

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

The development of distributed applications required new design techniques and models. They also resulted in new kinds of problems. In this section, you will look at the issues that you must consider when designing distributed applications. You will also look at two kinds of architectures that were developed to enable distributed application development:

- Remote Procedure Call-based (RPC-based) architectures.
- Message-based architectures.

The problems with the preceding architectures will also be discussed. Finally, you will look at the effect of the Web standards on distributed application development.

## Design Considerations for Distributed Applications

- Data types that are not compatible across different systems
- Server failures or loss of server response
- Client failures
- Retrying a call
- Security
- Synchronizing clocks between multiple computers

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

There are several common problems that you need to consider when designing a distributed application. These problems are not unique to any particular distributed application design.

### Different data types

Different operating systems support different data types. Sometimes, there is not a 100 percent compatibility of data types across different operating systems. Therefore, you must consider how to handle data types that are not compatible across different systems.

### Server failures

Because components of distributed applications are often remote, there are more single points of failure. Failure of any one point can cause the entire distributed application to fail. Therefore, you must consider how to handle server failures and loss of server response.

### Client failures

If a server is storing state on behalf of a client, and the client fails, then you must consider how the server will be notified. You also must consider if it is necessary to reclaim resources on the server that were in use by the client.

### Retrying calls

If a remote method is called and there is no response from the server, it may not be acceptable to retry calling the method. For example, if a method is called to purchase a large order of stock, and if the server received the request to place the order but the response was lost, then it would not be acceptable to resubmit the purchase order.

**Security**

In distributed applications, there are more opportunities for security threats. Not only must you consider authentication and authorization, but you also must consider how to secure the communication between a client and a server, and how to guard against man-in-the-middle attacks, denial-of-service attacks, replay attacks, and so on.

**Synchronizing clocks**

Many operations rely on time stamping. For example, it is not acceptable for a server to acknowledge that it received a purchase order before the purchase order was placed. This kind of a problem can arise if the clocks on the client and server computers are not synchronized. Therefore, you must decide how you will ensure the synchronization of the clocks on the various computers that communicate in a distributed application.

## RPC-Based Architectures

- **What is an RPC?**
  - RPCs are calls made to procedures or functions that reside on a remote system
- **Synchronous function calls**
- **Problems with RPC-based architectures**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

RPC-based architectures were among the first candidates to be considered as a solution to the design problems of distributed applications.

### What is an RPC?

A *remote procedure call (RPC)* is a call made to a procedure or function that resides on a remote system. An RPC looks like an ordinary procedure call or a function call within the code that uses it. An RPC provides both:

- Location transparency

The programmer does not need to know the physical location of the service provider.
- A familiar programming model

Most programmers are accustomed to using some form of procedure call.

The RPC infrastructure generates a stub, which acts as a representative of the remote procedure code and marshals any procedure arguments to a buffer, which may be transmitted over a network to the RPC server. On the RPC server, the stub unpacks the arguments in the server process and the arguments are passed to the actual function being called. Any return value is returned to the caller in a similar way.

### Synchronous function calls

In an RPC model, an application establishes a conversation with an appropriate RPC server. The RPC function calls look very similar to local procedure calls; also, the blocking semantics of RPCs are the same as those of local procedure calls. The fact that the blocking semantics are the same means that calls are synchronous, that is, the thread of execution is blocked until the function returns. For most developers, this is a very comfortable programming model. However, layering a synchronous model on top of a distributed architecture introduces some problems.

<b>Building redundancy</b>	The first problem that is inherent in RPC-based architectures is discovery. How can the application discover the information that is needed to connect to an endpoint that could supply the required services? The simple solution that is used in most applications is to hard-code the endpoint information. This is not an optimal solution because it makes building redundancy and failover capabilities into an application very difficult.
<b>Aggregate availability</b>	As an application begins to rely on multiple distributed services, it becomes more susceptible to the possibility of some critical service being unavailable. Therefore, the aggregate availability of a distributed application would be negatively affected by the brittleness of typical implementations. Typical implementations are brittle because they do not tolerate changes to their deployment environment very well without failure.
<b>Load balancing and failover</b>	Hard-coding the endpoints in an application results in another problem. Specifically, there is no simple way for an RPC-based application to do any form of dynamic load balancing. Neither can the application respond to server unavailability by dynamically failing over to an alternate server.
<b>Prioritization</b>	<p>Prioritization of requests is almost impossible, because all requests by default are handled on a first-come, first-serve basis. If a particular server is heavily loaded, the higher priority clients might be subjected to unacceptable delays.</p> <p>Consider an investment brokerage house. Most brokerage clients are small accounts. However, the brokerage would also have a number of large accounts that require special service because of their transaction volumes. In a volatile market, large clients must be given precedence over smaller customers. The brokerage house cannot afford to have the transactions of large clients queued behind transactions of smaller clients, at the risk of losing business from large clients.</p>
<b>Load spikes</b>	<p>Another significant problem with RPC-based applications is the inability to handle load spikes. Load spikes can have the following consequences:</p> <ul style="list-style-type: none"><li>■ Temporary server outages due to server failure.</li><li>■ Failure of an action because a required resource (for example, database connections) had been exhausted.</li><li>■ The need for more hardware than is required for typical loads, to handle the infrequent load spikes.</li></ul>



## Message-Based Architectures

- **Asynchronous Messaging**
- **Problems with Message-Based Architectures**
  - Message payload processing
  - Interoperability
  - Workflows and message sequencing

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

Another candidate architecture that you can use to build distributed applications is a message-based architecture. Message-Oriented Middleware provides applications with interprocess communication services by using message queuing technology as the basis for a guaranteed service level for critical applications. The queuing technology tracks a message along each leg of its route, much like delivery services for a commercial package company performs package tracking. This queuing technology ensures that any problem can be quickly detected, and possibly even corrected, without user intervention.

Message-based architectures have usually been built around message-queuing products such as Microsoft Message Queuing (formerly known as MSMQ).

### Asynchronous messaging

The most evident features of message-based architectures are that they are asynchronous and that they are based on the exchange of messages rather than function calls. Both of these features have some advantages, such as:

- Messages can be routed based on load and priority.
- Asynchronous calls allow clients to do productive work while waiting for a time-consuming operation.

However, these features introduce problems.

### Message payload processing

Because message-based systems transfer messages, one of the first tasks that the application programmer is responsible for is adding the functionality for packing and unpacking of the message contents. After unpacking the message contents, the application must still validate the contents. As the complexity and flexibility of the message payload increases, unpacking and validating messages becomes more difficult.

**Interoperability**

Most message-based systems are implemented by using proprietary message-queuing products. Using proprietary message-queuing products has at least two requirements in implementing interoperable messaging-based systems. All of the organizations participating in the distributed operation must have:

- Message queuing software.
- Bridging software to operate between the disparate messaging environments.

Even if the preceding requirements are met, the resulting solution tends to be difficult to implement and expensive. Therefore, message-based solutions are not viable as a standard way to implement distributed applications.

**Workflows and message sequencing**

Many distributed application scenarios involve workflows that are defined as a sequence of messages being exchanged between multiple computers. Because messages are sent asynchronously, it is possible that messages may arrive out of order. In some scenarios, it would be fatal if messages were processed in an incorrect sequence. For example, if a stock broker received orders to buy and sell, out of sequence, this could significantly affect the prices paid in each transaction. This means that the application developer has the additional burden of creating a high-level protocol layer on top of the messaging protocol to track the sequence of messages.

## Web Standards

- **Problems with binary protocols**
- **Web protocols and data formats**
  - HTML
  - HTTP
  - XML
- **Problems with the Web**
  - Security
  - Performance

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

Both RPC- and message-based architectures have been successfully implemented by many organizations, but these architectures suffer from a number of problems. In this topic, you will look at some of the problems inherent in current and legacy distributed object models, and how the adoption of Web standards in designing distributed applications alleviates many of these problems.

### Problems with binary protocols

Distributed object models such as Distributed Component Object Model (DCOM), Java Remote Method Invocation (RMI), and Common Object Request Broker Architecture (CORBA), suffer from the limitation of relying on binary protocols. Some of the problems inherent in using binary protocols include:

#### ■ Firewalls

The first problem is that binary protocols are point-to-point. As a result, any communication with an endpoint that is inside a firewall requires firewall administrators to open up a range of ports to allow communication. For most organizations, this is an unacceptable security risk.

#### ■ Interoperability

Another problem is interoperability between disparate object models. Each object model uses its own proprietary protocol. It is possible to provide software to translate the packets that are passed between the different object models. However, a loss of information always results because of the translation. The result is that most organizations use a single object model to implement all of their systems within an organization. Consequently, the environment of distributed applications is divided into different groups that are identified by the object model that each group has adopted. If a potential trading partner chooses a competing object model, this can cause significant problems.

- Data formats

Another problem with binary protocols is the encoding of data that is transmitted by using these protocols. Every protocol encodes data differently, which places a huge overhead on organizations when they have to consume data that was encoded in multiple, incompatible ways.

Also, the difficulty in translating data from one format to another leads to the segregation of organizations based on the data formats that they can handle.

Because of the problems in using binary protocols, there was a need for a ubiquitous protocol, and an easily parsable and transformable data encoding. It turned out that the emergence of the World Wide Web (WWW) provided the universal solution that everyone could easily use.

### The Internet and the Web

Transmission Control Protocol (TCP) and Internet Protocol (IP) were originally developed to connect different networks that different designers designed into a network of networks. Ultimately, this network of networks came to be known as the Internet.

Then, in late 1990, Tim Berners-Lee, a computer scientist at CERN invented the World Wide Web, which is also known as the Web. The *Web* is a globally interconnected network of hypertext documents. Emerging from this effort were two revolutionary technologies: Hypertext Markup Language (HTML) and Hypertext Transfer Protocol (HTTP).

HTML is a language that defines how to add markup (in the form of tags) to text documents to provide information to a Web browser on how to lay out the text in the document. The documents with HTML tags are known as hypertext documents.

### Advantages of HTTP

HTTP is the protocol that is used for requesting and receiving hypertext documents on the Web. A very important point to be noted about HTTP is that it is not restricted to work with just HTML documents. An example of this fact is that XML Web services and their clients can exchange XML documents by using HTTP.

As the popularity of the Web increased, HTTP as a protocol has been almost universally adopted. Using HTTP overcomes one of the major obstacles for the interoperation of distributed object models, specifically the lack of a ubiquitous, trusted protocol.

### XML - A Universal data format

Developers soon realized that although HTML allowed a document author to define presentation structure, it did not provide any way to define the structure of the data or the relationship between the data in a document. In 1996, this limitation led to the birth of a language for marking up text to describe the structure of the data in a document. This language is known as Extensible Markup Language (XML). Some of the goals of XML documents are that they must be:

- Easily usable over the Internet.
- Unambiguous.
- Easy to create.
- Easy to parse and process.
- Extensible, platform independent, and able to support localization.

The rapid adoption of XML is evidence of its suitability as a universal data format.

### Firewall friendly

The final contribution (in the context of this topic) of the Web is the Web server. Web servers typically communicate by using HTTP, which is a trusted, widely adopted protocol. An equally important aspect of a Web server is its role as a gateway to an organization. Web servers need not merely serve HTML content. Through the HTTP extensibility mechanisms, Web servers can also forward requests to an appropriate request handler. The Web server does not concern itself with how the handler interprets the payload of an HTTP request. This is because it is the responsibility of the handler to process the forwarded request and generate an HTTP response. The Web server sends the response back to the client.

Web servers can forward requests for any kind of service that an HTTP request describes and whose results can be packaged in an HTTP response. And all of this can be done without requiring any reconfiguration or loosening of firewall policy.

### Problems with the Web

In spite of the benefits that the Web has provided, there are some concerns in the areas of security and performance.

### Security

Because the Internet is a public infrastructure, it also means that any communication is potentially vulnerable to interception, modification, spoofing (a technique that is used to gain unauthorized access to a computer), etc.

---

**Note** A discussion of the various security mechanisms that different technologies use in the distributed applications domain is beyond the scope of this course. However, in Module 7, “Securing XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*, you will see how you can secure XML Web services that were built by using the Microsoft .NET Framework.

---

### Performance

The majority of Internet users still only have dial-up access to the Internet. This introduces significant performance problems and severely constrains the type and complexity of application that can be delivered over the Web. For example, some interactive applications require significant interaction with the server. The bandwidth limitations of dial-up connections severely limit the kinds of interactivity an application could support.

Performance issues combined with security concerns and reliability problems (even the largest Web sites are not immune to server outages and service unavailability) make designing applications for a private network a better solution, in some scenarios.

# Introducing XML Web Services

- **What Are XML Web Services?**
  - URL-addressable set of functionality exposed over a network
- **Based on Internet Technologies**
- **Building Blocks**
- **The Future of Distributed Applications**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

The problems with existing object models for distributed applications forced developers to look for alternatives. With the rapid adoption of Web standards, it is natural that solutions based on Web standards would be considered. This led to the evolution of XML Web services.

## What Are XML Web Services?

A Web service is an URL addressable set of functionality that is exposed over a network to serve as a building block for creating distributed applications. An early example of a Web Service is Microsoft Passport. Passport provides authentication services and all of its functionality is accessible through HTTP requests.

---

**Note** In this course, any mention of Web services specifically refers to XML-based Web services. Although other kinds of Web services are possible (for example, custom HTTP listeners), it is unlikely that they will be as popular and highly used as XML-based Web services.

---

## Foundations for XML Web Services

The foundations for XML Web services are HTTP, XML, and Simple Object Access Protocol (SOAP, a lightweight HTTP- and XML-based protocol that is used for information exchange). The development of these technologies is governed by the World Wide Web Consortium (W3C). You will learn about these technologies in greater detail later in Module 3, “The Underlying Technologies of XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

## Building Blocks

Like components, XML Web services are black boxes. They encapsulate the implementation and provide an interface for communicating with the XML Web service. Therefore, you can use XML Web services as building blocks for applications.

## No granularity restriction

There is no restriction on the granularity of an XML Web service. It can range from simple components such as an order-tracking component published by a shipping company to large applications such as hosted financial applications. You can apply XML Web services at many different levels of a solution.

**Static resources or interactive applications**

XML Web services can provide convenient access to a static set of information. For example, an XML Web service can allow a customer to request demographic information for a specified city.

Alternatively, developers might use XML Web services to implement highly interactive applications. For example, a travel Web site might make it possible to build an entire vacation itinerary online by using multiple XML Web services. The user can use XML Web services for making hotel and rental car reservations, planning flight itineraries, and booking flights, etc.

**Aggregating XML Web Services**

An XML Web service can aggregate other XML Web services to provide a sophisticated set of services. For example, an XML Web service for a real-estate agency might make use of an XML Web service for a credit verification to facilitate approval of online loan applications. In the future, more and more distributed applications will be built from XML Web services. In such applications, XML Web services will often be selected at run time based on different metrics, such as availability, cost, performance, and quality. This level of choice will be invaluable in designing redundant systems with failover capabilities.

---

**Note** Aggregating XML Web services are also known as federated XML Web services.

---

**The future of distributed applications**

Why should XML Web services succeed where all other technologies have failed? Let us look at the key characteristics of XML Web services that can enable its success.

**Interoperability**

XML Web services are intended to be invoked by using SOAP. Because SOAP is platform neutral, developers must no longer figure out how to build bridges between DCOM, CORBA, and other disparate protocols. Any XML Web service can interoperate with any other XML Web service.

Also, because XML Web services communicate by using HTTP and XML, any network node, which supports these technologies, can both host and access XML Web services.

**Multilanguage support**

Developers can write XML Web services in any language. Consequently, developers need not learn new languages or standardize on a single language to create or consume XML Web services.

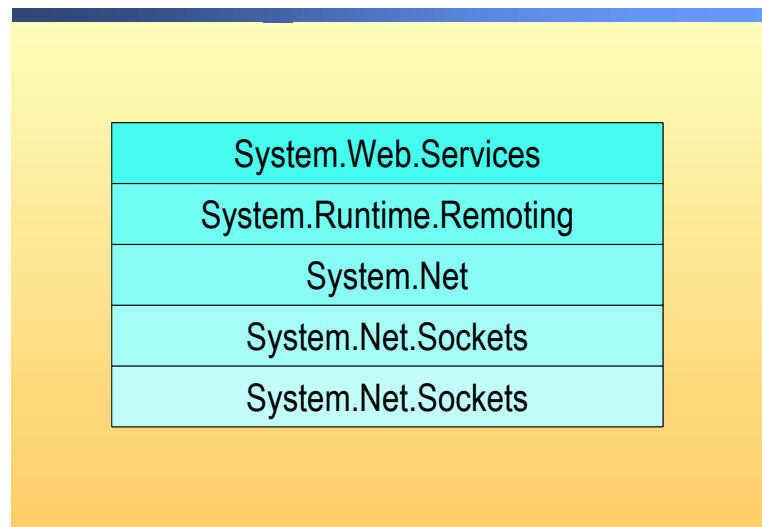
**Reusing existing applications**

It is very easy to expose existing components and libraries as XML Web services. Vendors, like Microsoft, provide tools to make the task of exposing components and libraries even easier. Most companies have a large number of existing components, libraries, and applications. It may be more cost effective to reuse the functionality in these software resources than to reimplement them.

**Use of industry-supported standards**

All of the major vendors are supporting technologies that are related to XML Web services, specifically, HTTP, XML, and SOAP. The universal support for these standards is unprecedented. This kind of support will make it very easy for heterogeneous systems to communicate. For example, a component that is written in C# and exported as an XML Web service can easily be used by a Common Gateway Interface (CGI) application that is written in C++, if that application were to make a SOAP request and process the result appropriately.

# The Web Technology Stack and .NET



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

When you consider implementing distributed applications, the wide array of implementation options might force you to make some trade-offs. For example, you might choose to trade ease of implementation for performance, or trade richness of services for complexity of communication.

The trade-offs involved in implementing a solution that is based on a specific level in the Web technology stack are outlined in this topic.

## TCP/IP

This is the lowest level in the technology stack. In this level, you can choose to allow the distributed elements of an application to communicate by using TCP/IP. The .NET Framework supports this type of application through the classes contained in the **System.Net.Sockets** namespace.

## Sockets

If you want to include session support in your application, then you can use sockets. The .NET Framework supports this type of application through the classes in the **System.Net.Sockets** namespace.

## HTTP

If you want to interact with Web servers or allow communication through corporate firewalls, then you can use HTTP. The .NET Framework supports this type of application through the classes in the **System.Net** namespace.

## XML or binary formats

You can implement a distributed application that is based on an object remoting solution. However, there are a number of problems related to object identity and the wire format of the remoted object. The wire format of the remoted object can be in binary format or perhaps an XML serialization of the object. The .NET Framework supports this type of application through the classes provided in the **System.Runtime.Remoting** namespace.



**SOAP**

If you want to implement distributed services that have a very loose coupling with the service consumers and are based completely on Web standards, then you can implement an XML Web service. The protocol of choice for this kind of application is the Simple Object Access Protocol (SOAP). A discussion on SOAP is provided in Module 3, “The Underlying Technologies of XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

The .NET Framework supports implementing XML Web services through the **System.Web.Services** namespace.

## The .NET Alternatives to XML Web Services

- Stand-alone listeners
- Custom protocol handlers
- .NET Remoting
  - .NET remoting vs. XML Web Services

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

XML Web services fall under the highest level in the technology stack. Depending upon the level of control that your application needs (for example, you might not have the option to compromise on performance), you might decide to implement your application at a lower level in the technology stack.

In theory, implementing a solution at a lower level in the technology stack increases the performance of the solution. This is because such a solution need not incur all of the overhead of the other technologies that are higher up in the stack. However, in practice, most developers do not have the skills to implement a scalable, robust, and maintainable solution, by using a low-level technology. Even if the developers have the skills, most of them do not have the time to implement such a solution.

.NET has several different implementation options, other than XML Web services, that are available for distributed application development. The following options are not trivial options. It is advisable to further investigate these options if you think that they may be a better solution to your requirements in comparison to XML Web services.

**Stand-alone listeners**

The first alternative to an XML Web service is a stand-alone listener. A stand-alone listener is an application that monitors a well-known port and responds to the messages it receives at that port. As a developer, you can implement stand-alone listeners in different ways:

- You can use a prewritten Internet Server Application Programming Interface (ISAPI) filter to handle all of the low-level socket communication and protocol implementation on behalf of your listener. Some examples are SOAP, Microsoft Active Server Page (ASP), and Microsoft Active Template Library (ATL) Server filters. You could then implement the listener functionality in a C++ class or an ASP page.
- You can implement an ISAPI filter that handles requests for documents with a specific extension and then decodes the contents of an HTTP request.
- You can write a server application that monitors a well-known port. You will then not be restricted to using HTTP or SOAP protocols. To write such an application, you can use the classes in the **System.Net** namespace of the .NET Framework.

The preceding list is not exhaustive. However, it introduces you to the options that are available for implementing stand-alone listeners.

**Custom protocol handlers**

If HTTP does not fit your requirements, then you can implement a custom protocol handler by deriving it from the **WebRequest** and **WebResponse** classes, which are found in the .NET Framework. You can still make use of the .NET serialization support when using your custom protocol, but the general object-remoting capabilities are not available.

**.NET Remoting**

If you need a remote component infrastructure, but do not need the level of interoperability that XML Web services provides, then you can use .NET Remoting. The **System.Runtime.Remoting** namespace provides classes to activate remote objects, marshal arguments by value and by reference, and make asynchronous calls, etc.

**.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, it is recommended that you consider the following when choosing which technology is more appropriate for the problem you are trying to solve:

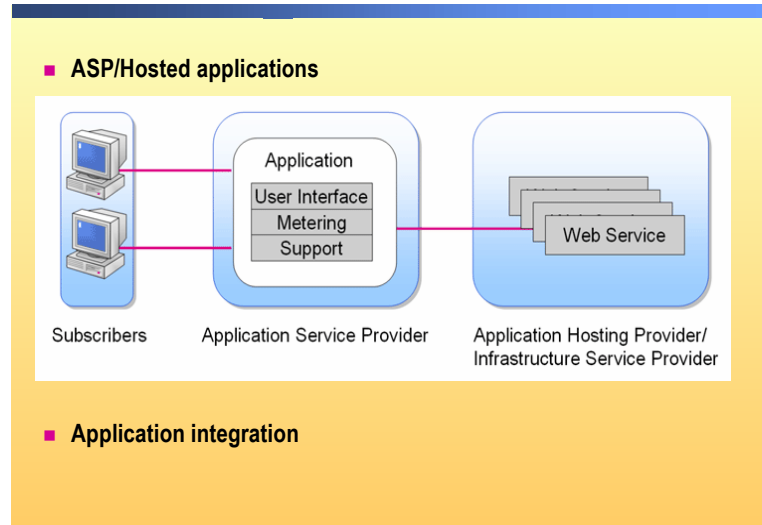
- .NET Remoting tends to be more appropriate for applications where the implementation of the applications at both endpoints is under the control of the same organization.
- XML Web services are more appropriate for applications where the client side of the service is likely to be outside the control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

---

**Note** See Course 2349B: *Programming the Microsoft® .NET Framework (Microsoft Visual C# .NET)* and Course 2415B: *Programming the Microsoft® .NET Framework (Microsoft Visual Basic .NET)*, for a detailed discussion on .NET Remoting.

---

## Common XML Web Service Scenarios



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

There are a number of scenarios where XML Web services are an appropriate solution.

### Application Service Providers/Hosted Applications

Application Service Providers (ASP) host applications that they then rent to subscribers. From a subscriber's perspective, the following are the characteristics of hosted applications:

- The application that the ASP hosts is viewed as a portal.
- The application that the ASP hosts exists in an isolated environment.
- Each subscriber has their own instance of the application.
- Subscribers do not share data with other subscribers.

From an ASP's perspective, all hosted applications must meet at least the following criteria:

- Application instances must be separately configurable for each subscriber, even on a shared hardware. This includes security settings.
- Applications must have mechanisms for measuring the duration of use of an application for billing purposes.

It is also useful if both an ASP and an application provide standard interfaces for mutual interaction.

ASPs do not have to host the applications at their own premises. In such cases, the physical application host often is the hosting provider. Having the physical application host act as the hosting provider allows the ASPs the flexibility in acquiring applications to offer to subscribers.

Considering the requirements that ASPs have for hosted applications, it is obvious that XML Web services are potentially a good solution for designing applications that are meant for hosting.

**Application integration**

Another potential use of XML Web services is in the area of application integration. Scenarios for application integration are generally characterized by a loose coupling with a published communication contract among the various applications that need to be integrated.

XML Web services provide useful capabilities in both of these aspects. By design, XML Web services are URL addressable, which provides for very loose coupling. Also, by using Web Services Description Language (WSDL), individual XML Web services can provide a contract that describes the XML Web service interface.

---

**Note** You will learn more about WSDL in Module 4, “Consuming XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

---

## Review

- Evolution of Distributed Applications
- Problems with Traditional Distributed Applications
- Introducing XML Web Services
- The Web Technology Stack and .NET
- The .NET Alternatives to XML Web Services
- Common XML Web Service Scenarios

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

1. What were some of the driving forces behind the development of distributed applications?
  - **The cost of mainframes**
  - **The issue of data ownership**
  
2. What are some of the problems that are associated with traditional distributed application architectures?
  - **Use of binary protocols**
  - **Use of proprietary data formats**
  - **Tightly-coupled solutions**
  - **Complexity of development**

3. What is an XML Web service?

**An XML Web service is a URL addressable set of functionality that is exposed over a network to serve as building blocks for creating distributed applications.**

4. What is the main difference between .NET remoting and XML Web services?

**.NET Remoting provides the infrastructure to support object remoting solutions, including functionality. XML Web services support only the transport of data, and not functionality.**





---

## Module 2: XML Web Service Architectures

### Contents

Overview	1
Service-Oriented Architecture	2
XML Web Services Architectures and Service-Oriented Architecture	4
Roles in an XML Web Services Architecture	8
The XML Web Services Programming Model	16
Review	18



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, places or events 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, trademarks, 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.

© 2001–2002 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Active Directory, Authenticode, IntelliSense, FrontPage, Jscript, MSDN, PowerPoint, Visual C#, Visual Studio, Visual Basic, Windows NT, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

# Instructor Notes

**Presentation:**  
**60 Minutes**

This module broadly describes the service-oriented architecture, which is a conceptual architecture. Then, the module explains how XML Web service architectures are a type of service-oriented architecture. It also describes the various roles within the XML Web service architecture.

**Lab:**  
**00 Minutes**

After completing this module, students will be able to:

- Identify how XML Web service architectures are a type of service-oriented architecture.
- Describe the elements of an XML Web service architecture and explain their roles.
- Describe the XML Web service programming model.

**Required Materials**

To teach this module, you need the Microsoft® PowerPoint® file 2524B\_02.ppt.

**Preparation Tasks**

To prepare for this module:

- Read all of the materials for this module.
- Try out the demonstration.

## Demonstration

### An Electronic Funds Transfer XML Web Service

This section provides demonstration procedures that are not appropriate for the student notes.

#### ► To demonstrate the NorthwindClient application

1. Start the application NorthwindClient.exe, which can be found in the folder `<installroot>\Labfiles\CS\Lab08_2\Solution\NorthwindClient\bin\Debug` or `<installroot>\Labfiles\VB\Lab08_2\Solution\NorthwindClient\bin`.
2. In the **From** list, click **Woodgrove Online Bank**.
3. In the **To** list, click **Contoso Micropayments**.
4. Click **Transfer**.
5. Explain that \$100 has been transferred from an account at the Woodgrove Bank to an account at the micropayment service, named Contoso.
6. Explain that the Northwind Traders XML Web service took care of all the details of managing the transfer, including retrieving routing numbers, and so on.

#### ► To show the Service Description pages for the Northwind, Woodgrove, and Contoso XML Web services

1. Open three separate browser windows.
2. In the first browser window, navigate to the following URL:  
`http://Localhost/Northwind/Traders.asmx`
3. In the second browser window, navigate to the following URL:  
`http://Localhost/Woodgrove/Bank.asmx`
4. In the third browser window, navigate to the following URL:  
`http://Localhost/Contoso/Micropayment.asmx`
5. Describe the relationship between the methods that are listed on each of the Service Description pages. Emphasize that the Northwind XML Web service is a client of the other two XML Web services.

#### ► To show that money is transferred between the accounts

1. Click the **GetAccount** link on the **Service Description** page to open the **Service Method Description** page for the **GetAccount** method of the Woodgrove XML Web service.
2. In the **acctID** box, type the account number by using the value of the **AccountID** field for the **From** account in NorthwindClient.exe.
3. Click **Invoke**.  
An XML document that contains the results of the method call is displayed.
4. Point out the value in the **balance** element in the XML document.
5. Click the **Transfer** button in the client application, NorthwindClient.exe.
6. Click the **Refresh** button on the browser window that displays the XML document.
7. Point out that the balance has been reduced by \$100.

## Module Strategy

Use the following strategy to present this module:

- Service-Oriented Architecture

Explain what a service-oriented architecture is. This topic is intended to provide the students with a conceptual framework to be able to understand the architecture of XML Web service-based solutions.

- XML Web Service Architectures and Service-Oriented Architecture

Explain the relationship between the conceptual service-oriented architecture and XML Web services architectures. Use the demonstration of the solution of the final lab in the course to show each of the XML Web service architectural elements as concrete implementations.

- Roles in an XML Web Service Architecture

This topic examines the specific roles in XML Web service architecture and explains that the Microsoft .NET Framework can provide assistance in implementing the functionality for each of the entities that plays a role.

- The XML Web Services Programming Model

Describe the features of the XML Web services programming model. Emphasize how this model is different than the traditional stateful, monolithic programming model. However, defer any in-depth discussion on how the XML Web services programming model affects the design of XML Web services until Module 8, “Designing XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.



# Overview

- Service-Oriented Architecture
- XML Web services architectures and Service-Oriented Architecture
- Roles in an XML Web services architecture
- The XML Web services programming model

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

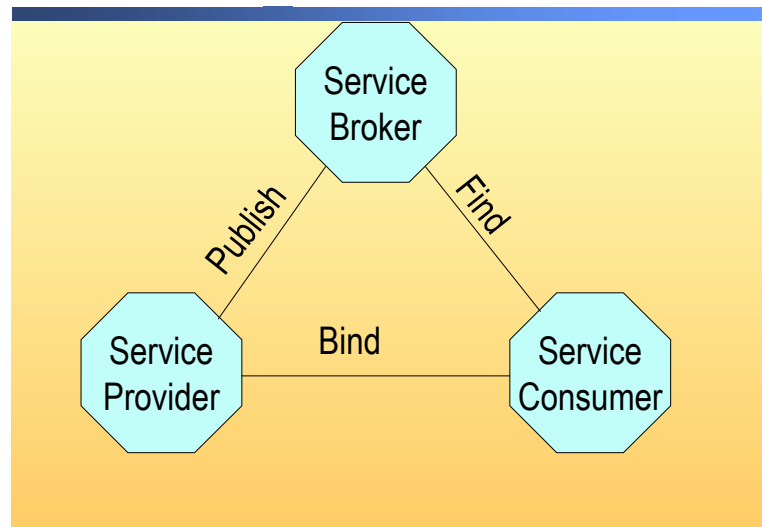
In this module, you will begin by looking at service-oriented architecture as a conceptual architecture for distributed applications. Next, you will examine how solution architectures that are based on XML (Extensible Markup Language) Web services are a type of service-oriented architecture. Then, you will examine each of the roles in an XML Web service architecture. Finally, you will look at the kind of programming model that an XML Web service architecture imposes.

## Objectives

After completing this module, you will be able to:

- Identify how XML Web services architectures are a type of service-oriented architecture.
- Describe the elements of an XML Web services architecture and explain their roles.
- Describe the XML Web service programming model.

# Service-Oriented Architecture



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

To build flexible, robust distributed applications, there are a number of requirements that must be met:

- When integrating software resources, the resources must be loosely coupled; that is, resources must be distinct and separate.
- Interprogram communication must be compliant with Internet standards.
- The service interfaces of software resources must be published for public use, and the interface definitions and documentation must be publicly accessible.

Building applications that meet the preceding requirements can result in the following advantages:

- You can construct applications by integrating core business processes with outsourced software services and resources.
- You can create more granular software resources.
- Reusable third-party software resources can provide cost and productivity benefits.
- The sale of software as services can become widespread. For example, a company could sell a shared calendar service as a Web accessible service instead of selling a stand-alone calendar application.

## Elements of a Service-Oriented Architecture

A service-oriented architecture is ideal for implementing such distributed applications. It is a conceptual architecture for implementing dynamic, loosely coupled, distributed applications.



Today, most business systems and applications are made up of tightly coupled applications and subsystems. When applications and subsystems are tightly coupled, a change to any one subsystem can cause many dependent components or applications to fail. This brittleness of existing systems is one of the primary reasons for the high cost of maintaining them; it also limits how easily the applications can be modified to satisfy changing business requirements.

A service-oriented architecture consists of three primary roles: service provider, service consumer, and service broker. A diagram of this architecture is shown on the preceding slide.

**Service provider**

A service provider is a node on the network (intranet or Internet) that provides access to the interface to a software service that performs a specific set of operations. A service provider node provides access to the services of a business system, a subsystem, or a component.

**Service consumer**

A service consumer is a node on the network that binds to a service from a service provider and uses the service to implement a business solution. In the service-oriented architecture model, service consumers are not applications, but nodes. However, for the purpose of this course, we will view a service consumer as a client application on a node.

**Service broker**

A service broker is a node on the network that is a repository of service descriptions and can be used like an address book to find the location of services. Service consumers can interrogate a service broker to locate a required service provider and service. Service brokers will often also act as service providers in cases where the requested service is service brokering.

**Interaction between the roles**

The preceding three service-oriented architecture roles interact to perform three basic operations:

- Publish services

Service providers publish their services to a service broker. The information published includes the service interface definition, location of service providers, and possibly other supporting information or documentation.

- Find services

Service consumers find required/desired services by using a service broker.

- Bind to services

Service consumers bind to specific services that are provided by a service provider. The binding process includes authentication of consumers.

Both finding and binding to services can be done dynamically to allow applications to configure themselves dynamically. For example, if an application finds that the response time from a service provider has become unacceptable, then it might decide to switch to another service provider at run time.

# XML Web Services Architectures and Service-Oriented Architecture

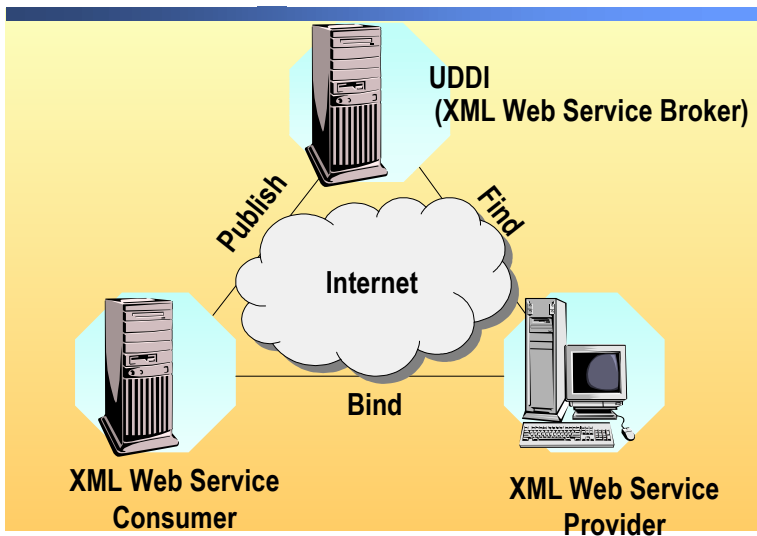
- Overview of XML Web Service Architectures
- XML Web services as an implementation of a Service-Oriented Architecture
- Demonstration: An Electronic Funds Transfer Web Service

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Before delving into the details of implementing and using XML Web services, it is important to understand how an XML Web service architecture is a type of service-oriented architecture.

First, you will examine XML Web service architectures. Next, you will examine the mapping between elements of the XML Web service architecture and elements of the service-oriented architecture. Finally, you will view a demonstration of a working example of an XML Web service solution, specifically the components of the solution architecture.

## Overview of XML Web Service Architectures



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

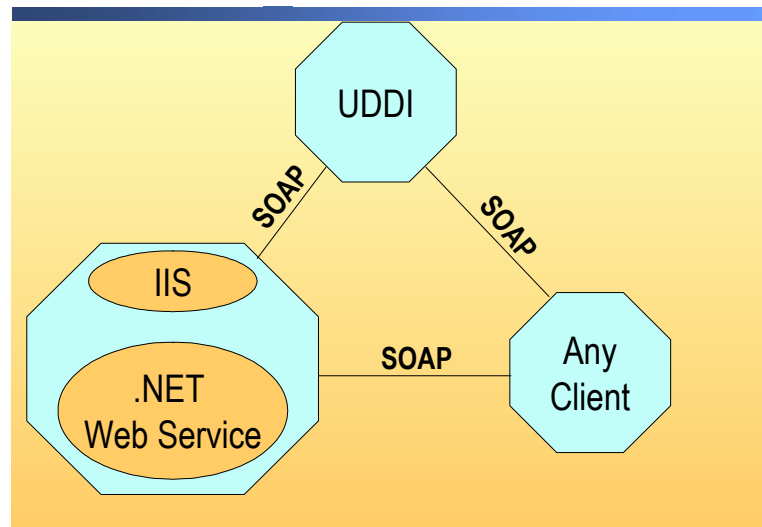
The basic elements in an XML Web service architecture are:

- The XML Web service provider, which is a network node hosting an XML Web service.
- The XML Web service consumer, which is a network node hosting any client that can communicate by using Hypertext Transfer Protocol (HTTP). The clients include browsers, console applications, and traditional graphical user interface (GUI) applications.
- The XML Web service broker, which is a network node hosting a global registry of available XML Web services much like a global address book.

All of these network nodes should be able to communicate with each other typically through a Transmission Control Protocol/Internet Protocol (TCP/IP) based network.

The diagram on the slide shows the relationship among the various elements of an XML Web service architecture. The rest of this module focuses on how the elements of an XML Web services architecture correspond to service-oriented architecture and then focuses on the various elements of the architecture.

## Web Services As an Implementation of a Service-Oriented Architecture



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

In a solution based on XML Web services, the three network nodes defined in a service-oriented architecture correspond to the elements of the XML Web services solution.

### The service broker in XML Web services

The role of a service broker is primarily fulfilled by a node that hosts a Universal Description, Discovery, and Integration (UDDI) registry. You will be introduced to UDDI later in this module. For more complete coverage of programming XML Web services and XML Web service consumers by using UDDI, see Module 6, “Publishing and Deploying XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

### The service provider in XML Web services

The role of a service provider is fulfilled by nodes that expose XML Web services through ASP.NET pages with the extension .asmx. For more information about the implementation details, see Module 5, “Implementing a Simple XML Web Service,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

---

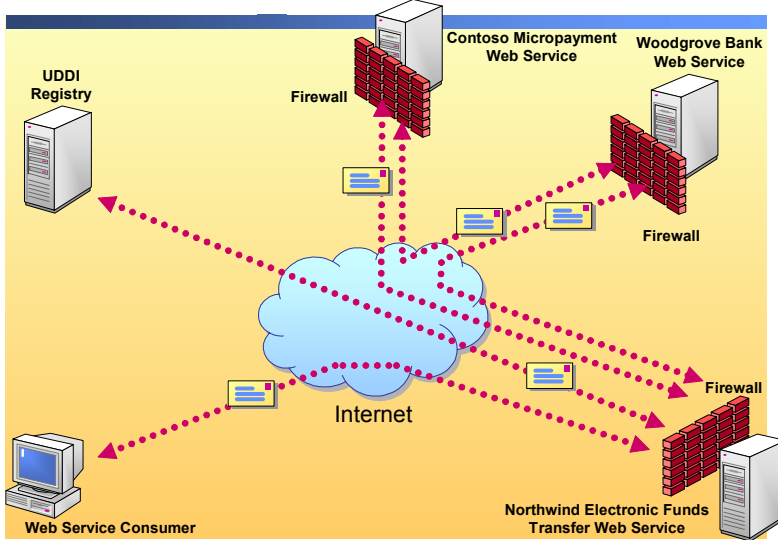
**Note** The entry points to XML Web services that are implemented by using ASP.NET are Web pages with the extension .asmx.

---

### The service consumer in XML Web services

The role of a service consumer is fulfilled by any node that can communicate by using Simple Object Access Protocol (SOAP) or HTTP, understands the service interface that is being used, and can supply the necessary authentication credentials.

## Demonstration: An Electronic Funds Transfer Web Service



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this demonstration, you will see an actual implementation of the concepts that you learned in the previous two topics. The demonstration will focus on how to create an XML Web services-based solution for a sample electronic funds transfer. You will build your own version of this solution in the labs for Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

## Roles in an XML Web Services Architecture

- The XML Web service provider
- The XML Web service consumer
- The XML Web service broker

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Earlier in this module you saw that an XML Web service architecture consists of the following elements: XML Web service provider, XML Web service consumer, and XML Web service broker. We will now briefly examine each of these roles.

## The XML Web Service Provider

- Web servers
- The .NET Common Language Runtime
- Examples of XML Web service providers

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

One of the important roles in an XML Web service architecture is that of the XML Web service provider. In this topic, you will examine the infrastructure that an XML Web service provider makes available to support and host XML Web services.

Some examples of the infrastructure that an XML Web service provider (a network node) must provide to an XML Web service are HTTP protocol handling and authentication services. If an XML Web service provider cannot offer such infrastructure, then the XML Web service must support this infrastructure. However, this would make developing XML Web services much more difficult.

You will examine some of the infrastructure that is provided when using Microsoft Internet Information Services (IIS) and Microsoft ASP.NET on a computer running Microsoft Windows® as the XML Web service provider.

### Web servers

At a minimum, an XML Web service provider must include a protocol listener. For XML Web services that are developed by using the Microsoft .NET Framework and Microsoft Visual Studio® .NET, the protocol listener must be an HTTP listener.

Because an XML Web service provider might be hosting multiple XML Web services, it must also be able to direct the request to an appropriate XML Web service. This is analogous to the Remote Procedure Call Subsystem (RPCSS) service that is responsible for handling incoming Distributed Component Object Model (DCOM) requests and directing them to an appropriate Component Object Model (COM) server.

Unknown XML Web service consumers can access an XML Web service provider. Therefore, at a minimum, the Web Server must provide basic security services at the protocol level.

Microsoft Internet Information Services (IIS), which is a Web server, provides all of the services that an XML Web service requires through its features:

- IIS is an HTTP listener.
- IIS can act as a gateway to the implementations of the various XML Web services that it may host, through its pluggable Internet Server Application Programming Interface (ISAPI) architecture.
- IIS provides significant security infrastructure.

You will see how to secure XML Web services by using the security capabilities of IIS in Module 7, “Securing XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

### **IIS and XML Web services**

A Web server such as IIS can invoke a service on behalf of a client, by using many different options. A Web server can start a Common Gateway Interface (CGI) application; run a script interpreter as done in Microsoft Active Server Pages (ASP), or invoke an ISAPI application.

When IIS works in conjunction with the common language runtime, it uses an ISAPI filter to intercept requests for pages with the extension .asmx, and then start a run-time host. The run-time host then executes the code for an XML Web service that is implemented by using the .NET Framework.

IIS is not restricted to hosting .NET-based XML Web services. It can also host Microsoft Active Template Library (ATL) Server-based XML Web services. ATL Server-based XML Web services are beyond the scope of this course. However, .NET-based XML Web service provides some significant advantages. One of the most important advantages is the flexible security infrastructure that the .NET platform provides. For more information, see Module 7, “Securing XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

### **Examples of XML Web service providers**

If an organization wants to provide XML Web services, it must be capable of providing some kind of electronic service. Because almost any piece of functionality can be classified as a service, it is impossible to enumerate all the possible kinds of XML Web services. However, two common examples of XML Web service providers are independent software vendors and general-purpose business processes.

Independent software vendors own products that perform a variety of tasks. These products can be exposed as individual XML Web services or aggregations of XML Web services. For example, a company that sells a calculation application for personal taxes might want to make that application accessible as an XML Web service.

General-purpose business processes, which are sufficiently generalized for adoption by a wide variety of clients, can also be exposed as XML Web services. For example, a payroll processing service can offer its payroll management services as an XML Web service.



## The XML Web Service Consumer

- Minimum functionality
- Service location
- Proxies
- Asynchronous calls
- Examples of XML Web service consumers

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

In this topic, you will look at the minimum set of functionality required for an XML Web service consumer to use an XML Web service. You will also look at how a consumer locates an XML Web service; the role of proxies in simplifying the implementation of XML Web service consumers; and how to use proxies to make asynchronous calls to XML Web services.

### Minimum functionality

To consume an XML Web service, an XML Web service consumer must call the methods of an XML Web service with the appropriate parameters by using the protocols (for example, SOAP) that the service supports.

It is difficult to correctly format messages before passing them to an XML Web service, and it is also difficult to handle the details of the protocols that the XML Web service supports. The .NET Framework provides classes that encapsulate most of the low-level details. Encapsulating the low-level details frees the developer from having to implement the infrastructure.

### Service location

Before an XML Web service can be used, a consumer must be able to locate it. Locating an XML Web service can be done statically by hard-coding the endpoint in the XML Web service consumer at design time. Alternately, the XML Web service consumer can dynamically discover the location of an XML Web service at run time. This provides an XML Web service consumer with the flexibility of choosing between equivalent, competing XML Web services based on criteria such as price or performance.

The standard mechanism for locating appropriate XML Web services, their service description, and their endpoints is through a UDDI registry. For more information about how an XML Web service consumer can make use of UDDI to locate an XML Web service and how to advertise an XML Web service in a UDDI registry, see Module 6, "Publishing and Deploying XML Web Services," in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

**Proxies**

When implementing an XML Web service consumer, developers can spend their time more productively on other issues, and should not have to concern themselves with the following tasks:

- Working with the underlying protocols.
- Parsing byte streams to extract data.
- Validating the inbound data streams.
- Constructing the outbound data packets.

However, the developer is often forced to handle the preceding tasks because there is no pre-built code available to perform these tasks. A typical approach to handling these tasks is to encapsulate or hide the implementation details in a wrapper class that acts as a proxy for the XML Web service. Not only can the proxy classes hide implementation details, but they also provide the developer with a familiar programming model of calling methods on objects.

The only problem with this technique is that a proxy class must be implemented for every XML Web service interface that an XML Web service consumer wants to interact with.

Microsoft provides a tool called Wsdl.exe to implement XML Web service proxy classes. However, there are some pitfalls inherent in making the programming interface to an XML Web service look like a local procedure call. For more information, see Module 4, “Consuming XML Web Services,” and Module 8, “Designing XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

Because an XML Web service interface is defined by using XML, it is also fairly straightforward to write tools that can automatically generate the proxy wrapper classes.

**Asynchronous calls**

Because XML Web services are usually accessed over networks that are not as reliable or fast as local area networks (LAN), it is often better to implement XML Web service consumers that make asynchronous calls to XML Web services. The proxies that are generated by using Wsdl.exe allow the caller to make asynchronous calls to a Web server. The proxy class in conjunction with the runtime handles details of thread pool management, the completion of a callback notification method, and so on. For more information about how to make asynchronous calls to an XML Web service, see Module 4, “Consuming XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

**Examples of XML Web service consumers**

Line-of-business applications will probably be the primary users of XML Web services, but there are a number of types of businesses that could be XML Web service consumers. Two examples of these types of businesses are online newspapers and an Application Service Provider (ASP).

An online newspaper might use multiple XML Web service news feeds. The incoming news feeds could be formatted, filtered, catalogued, and made searchable according to customer preferences.

An Application Service Provider (ASP) might host XML Web services, re-brand XML Web services, or do both. Also, an ASP might aggregate multiple XML Web services and offer the composite XML Web service to its customers.

## The XML Web Service Broker

- Interactions between brokers and providers
- Interactions between brokers and consumers
- UDDI registries

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

Just as a service-oriented architecture needs a service broker, an XML Web service architecture also needs a service broker. To facilitate interactions, businesses need a comprehensive solution to publish their information to any customer or business partner around the world. An XML Web service broker interacts with both XML Web service providers and XML Web service consumers to provide this functionality.

A common means of publishing information about business services will make it possible for organizations to:

- Quickly discover the correct trading partners out of the millions that are online.
- Define how to conduct business after preferred businesses are discovered.
- Create an industry-wide approach for businesses to quickly and easily integrate with their customers and partners on the Internet. Organizations will be able to share information about their products and services, and how they prefer to be integrated into each other's systems and business processes.

**Interactions between brokers and providers**

Brokers specify to providers what kinds of information needs to be made public, and then publishes this information. The kinds of information published by a broker include:

- Classification information to allow XML Web services to be categorized.
- Contact information for the XML Web service.
- A textual description of the offered services.
- Links to documents providing information about the XML Web services that the provider hosts.
- The location of endpoints for XML Web services.

These locations are usually stored as Uniform Resource Locators (URLs) that denote the location of the advertised XML Web services. Because it is not feasible to specify all of the information in the broker's repository, there may be pointers to URLs or file-based resources that will facilitate further discovery. For example, service-level guarantees or information regarding authentication requirements may be discoverable only at an XML Web service provider's location.

**Interactions between brokers and consumers**

The primary interaction between XML Web service consumers and the XML Web service broker is related to searching. Brokers must facilitate the search of their repository to enable XML Web service consumers to locate the appropriate XML Web service and then discover the information that is required to bind to that XML Web service.

**UDDI registries**

There are many approaches to providing the XML Web service brokering services.

One simple approach is to have all of the potential trading partners communicate binding information to each other by using a specific method created for that purpose. In this approach, you specifically do not require a broker. For example, some organizations using electronic data interchange (EDI) simply publish the list of required EDI document specifications that the trading partners must use on a Web site. The problem with this approach is that there is no easy way to discover which of the external businesses is compatible with your business.

Another approach is to have all of the businesses publish an XML Web services description file on their Web site. Then, Web crawlers can automatically access a registered URL and can index the description files for the XML Web services that are found at each Web site. An XML Web service broker could then provide a portal that gives access to the indexes that the Web crawlers build. Relying on Web crawlers to provide indexes for XML Web services has similar problems to the problems encountered today with standard Web search engines and catalogs that we have today. The problem is that there is no mechanism to ensure consistency in service description formats and for the easy tracking of changes whenever they occur. Just as Web search engines return many invalid links, such a mechanism for XML Web services would also result in out-of-date service descriptions and binding information.

The brokering approach that has been chosen for XML Web services relies on a distributed registry of businesses and their service descriptions that are implemented in a common XML format. The solution that implements this approach to solving the discovery problem is known as Universal Description, Discovery, and Integration (UDDI).

UDDI is a specification for distributed Web-based information registries of XML Web services. UDDI is also a public set of implementations of the specification that allow businesses to register information about the XML Web services that they offer so that other businesses can find them.

The core component of a UDDI-compliant registry is a business registration element, which is an XML file that describes a business entity and its XML Web services. Conceptually, the information specified in a business registration has three parts:

- Business addresses, contact information, known identifiers, and so on.
- Lists of industrial categorizations that are based on standard taxonomies.
- The technical information about the XML Web services that the business exposes.

This information includes references to XML Web service interface specifications, and potentially, pointers to various file- and URL-based discovery mechanisms.

For more information about how to publish an XML Web service in a UDDI-compliant registry and how to search a UDDI-compliant registry to locate XML Web services, see Module 6, “Publishing and Deploying XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

# The XML Web Services Programming Model

- Web protocols
- Stateless
- Loosely coupled
- Universal data format

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

To successfully implement or consume an XML Web service, it is important to understand the key features of the XML Web services programming model. It is also important to understand some of the ramifications of the programming model.

## Web protocols

The first feature of the XML Web services programming model is that the communications protocol will typically be HTTP. However, HTTP does not intrinsically support the concept of a method invocation. Because of this constraint, XML Web service consumers often use the XML-based SOAP over HTTP for invoking the XML Web service methods. Therefore, it is essential for a developer to have at least a working knowledge of both HTTP and SOAP. For more information about HTTP, XML, and SOAP, see Module 3, “The Underlying Technologies of XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

## Stateless

Most developers are familiar with a stateful object model. In other words, an instance of a class is created and then various operations are performed on the object. Between each method invocation, the object maintains its state. In a stateless environment, the object retains no state between calls. Any state that needs to be persisted between calls can be stored in a database or a cookie.

XML Web services are not objects in the traditional sense. When using ASP.NET to implement an XML Web service, you can use a C# class to implement it. This class is referenced by an ASP.NET page with the extension .asmx. When the page is processed, an instance of this class is created. The lifetime of the .asmx page binds the lifetime of the resulting object, which means that a different object instance will handle every method invocation. As a result, the classes that implement an XML Web service are stateless. Although designing stateless systems can be initially more difficult, stateless systems can easily scale-out as the load on the system increases.

For more information about how to design stateless XML Web services and how to manage state in stateless XML Web services, see Module 8, “Designing XML Web Services,” in Course 2524B, *Building Web Services Using Microsoft ASP.NET*.

### Loosely coupled

In a non-distributed application, if any of the required software resources, such as a function library in a dynamic-link library (DLL), are available when an application is launched, they will continue to be available for the lifetime of the application. Usually, they will also be available on each successive use of the application. For distributed applications, especially distributed applications that make use of software resources over the Internet, there is an increased likelihood that the required software resources will not always be available. Therefore distributed applications that are implemented by using XML Web services must be more resilient to software resources becoming unavailable, even at run time.

As a consequence, XML Web service-based solutions must be loosely coupled so that they can dynamically reconfigure themselves if a resource becomes unavailable. Loosely coupled applications also have the advantage of allowing failover because the consumers will not have affinity with any particular instance of an XML Web service.

For more information about how to design XML Web services to facilitate loose-coupling, and also learn how to implement loosely coupled XML Web services, see Module 8, “Designing XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

### Universal data format

The universal data format that is used in XML Web services is XML. A complete coverage of XML is beyond the scope of this course, but a working knowledge of XML is imperative to implement and consume XML Web services.

The following are a few of the areas where XML is used in XML Web services:

- The SOAP protocol is XML-based.
- XML Web service descriptions are XML documents.
- Data returned from an XML Web service is in an XML document.
- XML Web services are registered with a UDDI registry by using XML documents that are business service descriptions.
- ASP.NET applications are configured by using XML configuration files.

## Review

- **Service-Oriented Architecture**
- **XML Web services architectures and Service-Oriented Architecture**
- **Roles in an XML Web services architecture**
- **The XML Web services programming model**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

1. What are the three main components of a service-oriented architecture?
  - **Service provider**
  - **Service consumer**
  - **Service broker**
2. What service-oriented architecture role does a network node with IIS and the runtime have in an XML Web service architecture?

**XML Web service provider**



- 
3. Which wire format is used by an XML Web service and an XML Web service consumer to communicate with each other?

**SOAP**

4. Name two of the characteristics of the XML Web services programming model.

**The answers can be any two of the following:**

- **Use Web protocols**
- **Are stateless**
- **Are loosely coupled**
- **Use XML as the universal data format**



---

## Module 3: The Underlying Technologies of XML Web Services

### Contents

Overview	1
HTTP Fundamentals	2
Using HTTP with the .NET Framework	8
XML Essentials	17
XML Serialization in the .NET Framework	26
SOAP Fundamentals	29
Using SOAP with the .NET Framework	36
Lab 3.1: Issuing HTTP and SOAP Requests Using the .NET Framework	45
Review	54



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, places or events 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, trademarks, 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.

© 2001–2002 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Active Directory, Authenticode, IntelliSense, FrontPage, Jscript, MSDN, PowerPoint, Visual C#, Visual Studio, Visual Basic, Windows NT, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

## Instructor Notes

**Presentation:**  
**120 Minutes**

This module provides students with an overview of the technologies that form the foundation of Extensible Markup Language (XML)-based Web services.

**Lab:**  
**45 Minutes**

After completing this module, students will be able to:

- Describe the structures of a Hypertext Transfer Protocol (HTTP) request and response.
- Issue HTTP **POST** and **GET** requests and process the responses by using the Microsoft® .NET Framework.
- Describe data types by using the XML Schema Definition language (XSD).
- Explain how to control the way a .NET Framework object is serialized to XML.
- Describe the structures of a Simple Object Access Protocol (SOAP) request and response.
- Issue a SOAP request and process the response by using the .NET Framework.

**Required Materials**

To teach this module, you need the Microsoft PowerPoint® file 2524B\_03.ppt.

**Preparation Tasks**

To prepare for this module:

- Read all of the materials for this module.
- Try the walkthroughs and demonstrations in this module.
- Complete the lab.

## Module Strategy

This module is intended to explain the technologies underlying XML Web services. Throughout this module, you should emphasize to the students the simplicity of the technologies that are covered.

Use the following strategy to present this module:

- HTTP Fundamentals

This section is intended to provide students with a basic understanding of the HTTP protocol and explain how to issue HTTP requests by using the .NET Framework. Explain that HTTP is a simple protocol that is designed for interoperability and not performance. Emphasize how simple HTTP is to understand.

- Using HTTP with the .NET Framework

Use the code walkthroughs in this section to show the use of the base classes that the .NET Framework provides to access data from the Internet. Emphasize that these classes encapsulate the HTTP-specific operations when communicating with a Web server, and show how synchronous and asynchronous operations are supported.

- XML Essentials

Explain that XML is fundamental to XML Web services. Do not spend much time on the basics of XML. Briefly review the important XML concepts. Cover the topics on XSD as a progressive tutorial, rather than a list of concepts.

- XML Serialization in the .NET Framework

Explain how you can modify the default serialization behavior for .NET Framework data types. Explain the importance of the ability to modify the default serialization behavior of data types.

- SOAP Fundamentals

This topic is intended to provide students with a basic understanding of the SOAP protocol and explain how to issue SOAP requests by using the .NET Framework. Emphasize that SOAP is the preferred wire format for XML Web services.

- Using SOAP with the .NET Framework

Explain to the students that the .NET Framework handles most of the details of communication between an XML Web service and an XML Web service consumer when using SOAP in XML Web services that are implemented by using the .NET Framework.

# Overview

- HTTP Fundamentals
- Using HTTP with the .NET Framework
- XML Essentials
- XML Serialization in the .NET Framework
- SOAP Fundamentals
- Using SOAP with the .NET Framework

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

XML Web services are built on Web technologies. The three core technologies that form the foundation for XML Web services are the Hypertext Transfer Protocol (HTTP), the Extensible Markup Language (XML), and the Simple Object Access Protocol (SOAP). It is important to understand how the three technologies work and how the Microsoft® .NET Framework provides support for these three technologies so that developers can use them in XML Web services.

## Lesson objectives

After completing this module, you will be able to:

- Describe the structures of an HTTP request and response.
- Issue HTTP **POST** and **GET** requests and process the responses by using the Microsoft .NET Framework.
- Describe data types by using the XML Schema Definition language (XSD).
- Explain how to control the way a .NET Framework object is serialized to XML.
- Describe the structures of a SOAP request and response.
- Issue a SOAP request and process the response by using the .NET Framework.

# HTTP Fundamentals

- Overview of HTTP
- Structures of HTTP Requests and Responses
- The GET and POST Methods

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

The HTTP is a World Wide Web Consortium (W3C) standard protocol for transferring documents on the Internet. XML Web services use HTTP for communication. It is a generic, stateless, protocol, which can be used for many tasks in addition its original use for hypertext.



## Overview of HTTP

### ■ Structure of an URL

```
http://host[:port][path[?querystring]]
```

### ■ Example

```
http://www.woodgrovebank.com/accts.asp?AccNo=23
```

### ■ Stateless protocol

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

#### Introduction

A resource location is specified in HTTP through a mechanism known as a Uniform Resource Locator (URL). Strictly speaking, the mechanism that is used in HTTP is a Uniform Resource Identifier (URI), but we can also think of it as a URL.

URIs are a slightly more general scheme for locating resources on the Internet that focuses more on the resource and less on the location. In theory, a URI could find the closest copy of a mirrored document or locate a document that has moved from one site to another. URLs are the set of URI schemes that have named the resource and contain explicit instructions on how to access the resource.

#### Structure of an URL

The syntax of a URL is as follows:

```
http://host[:port][path[?querystring]]
```

The following is an example of a URL:

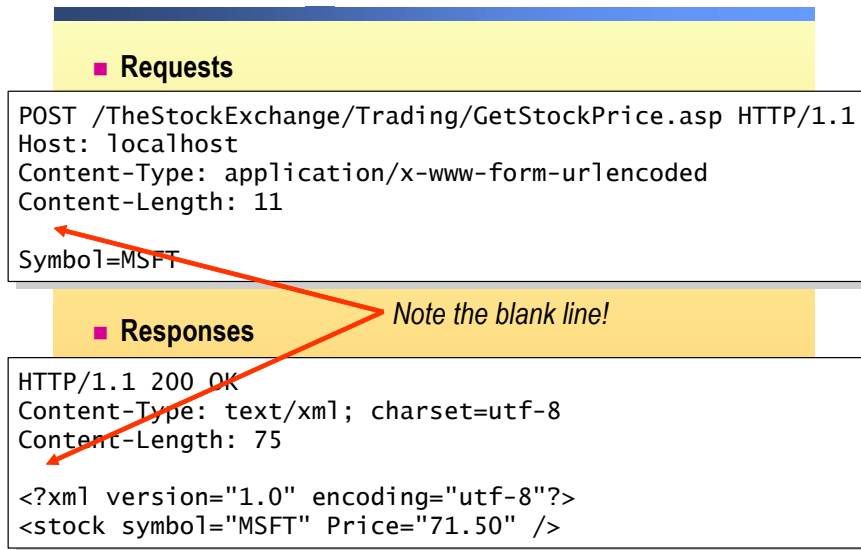
```
http://www.woodgrovebank.com/accts.asp?AccNo=23
```

In the preceding example, *www.woodgrovebank.com* is the host, *accts.asp* is the path and *AccNo=23* is the query string. If the port number is not specified (as in the preceding example), the default port for HTTP, which is port 80, is used.

#### Stateless protocol

HTTP is a *stateless* protocol. This means that whenever the client makes a request, the connection to the server is closed after the client receives the response. Therefore, if any state must be maintained between the client and the server, the server must pass on state information with the response to the client. This will enable the server to recover this information from the client when it receives the next request. For example, if you implement a Web site that displays user-specific content, you would have to implement a mechanism that retains information about the current user to display personalized content.

## Structures of HTTP Requests and Responses



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

HTTP requests and responses have a simple structure.

### Structure of an HTTP request

An HTTP request has the following format:

```

method URL Version
headers
a blank line
message body
  
```

### Example

The following code shows an example of an HTTP request:

```

POST /TheStockExchange/Trading/GetStockPrice.asp HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 11
  
```

```

<?xml version="1.0" encoding="utf-8"?>
<Symbol=MSFT />
  
```

The first line in an HTTP request is known as the request line, and the methods that a request supports are as follows:

- **OPTIONS**
- **GET**
- **HEAD**
- **POST**
- **DELETE**
- **TRACE**
- **CONNECT**
- **extension-method**

**Structure of an HTTP response**

An HTTP response has the following format:

*Version Status-Code Description*  
*headers*  
*a blank line*  
*message body*

**Example**

The following code shows an example of an HTTP response:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 75

<?xml version="1.0" encoding="utf-8"?>
<stock symbol="MSFT" Price="71.50" />
```

## The GET and POST Methods

### ■ HTTP-GET

```
GET /Trading/GetStockPrice.asp?Symbol=MSFT HTTP/1.1
Host: localhost
```

### ■ HTTP-POST

```
POST /Trading/GetStockPrice.asp HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 11

Symbol=MSFT
```

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

The **GET** and **POST** request methods are ideal for communicating with an XML Web service. These methods are designed specifically for submitting data to a Web server and retrieving a specified resource from a Web server. This makes it possible to layer a function call model on top of these methods, which is exactly the model that XML Web services requires.

### HTTP-GET request

Consider the following HTTP-GET request:

```
GET /Trading/GetStockPrice.asp?Symbol=MSFT HTTP/1.1
Host: localhost
```

The most important feature of the request line is the querystring. The querystring is the portion of the URI that follows the question mark, and consists of a set of URL-encoded name/value pairs.

In an HTTP-GET request, there is typically no message body. The response for an HTTP-GET request is just a standard HTTP response, which is described in the preceding topic.

### HTTP-POST request

Consider the following HTTP-POST request:

```
POST /Trading/GetStockPrice.asp HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 11
```

```
Symbol=MSFT
```

In the preceding code, notice that there is no querystring as part of the URI. This is because the information about the request is contained in the message body. This feature of an HTTP-POST request makes it a very convenient way of passing larger sets of data to the server in contrast to an HTTP-GET where the size of the querystring is restricted to 1024 bytes. Also, transmitting the data as part of the message body imposes fewer restrictions on the kind of data that is sent to the server.

In Module 5, “Implementing a Simple XML Web Service,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*, you will see how the choice of an HTTP request method affects the kinds of interfaces that the XML Web services can expose.

## Using HTTP with the .NET Framework

- .NET Classes for working with HTTP
- Code walkthrough: issuing a synchronous HTTP request
- Code walkthrough: issuing an asynchronous HTTP request

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

The preceding section described the fundamentals of the HTTP protocol and the basic request and response model that is used to access data on the Internet.

In this section, you will learn about the specific classes that the .NET Framework provides to access data by using the HTTP protocol. You will also learn how to issue synchronous and asynchronous HTTP requests.

## .NET Classes for Working with HTTP

- **HttpRequest and HttpResponse**
- **StreamReader and StreamWriter**
- **Support for synchronous and asynchronous operations**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Issuing an HTTP request and receiving a response is easy using the .NET Framework. The following classes in the .NET Framework provide all of the required basic functionality:

- **HttpRequest** and **HttpResponse** classes in the **System.Web** namespace
- **StreamReader** and **StreamWriter** classes in the **System.IO** namespace

### **HttpRequest and HttpResponse**

**WebRequest** and **WebResponse** are abstract base classes in the .NET Framework for accessing data from the Internet in a protocol-neutral way. The **HttpRequest** and **HttpResponse** classes, which are derived from **WebRequest** and **WebResponse** respectively, encapsulate the HTTP-specific aspects of the communications with a Web server. Most importantly, they provide easy access to the HTTP headers, and the underlying request and response streams.

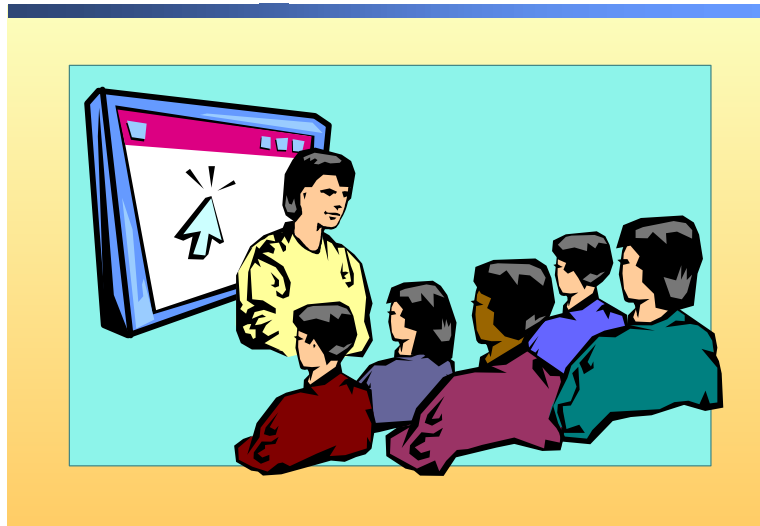
### **StreamReader and StreamWriter**

The **StreamReader** and **StreamWriter** classes are two utility classes that are used to read and write streams by using a specific encoding (UTF-8/UTF-16, etc.).

### **Support for synchronous and asynchronous operations**

The **HttpRequest** class supports both synchronous and asynchronous requests. In the next two topics, you will look at code samples of synchronous and asynchronous operations.

## Code Walkthrough: Issuing a Synchronous HTTP Request



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this code walkthrough, you will look at how a synchronous HTTP request is issued by using the .NET Framework.

This topic examines the functionality that the following sample code for a synchronous request implements. (A C# and a Microsoft Visual Basic® .NET example are provided.)



**C# code example**

```

1.      HttpRequest req = (HttpRequest )↵
           WebRequest.Create(url);
2.      req.ContentType=contentType;
3.      req.Method = method;
4.      req.ContentLength=content.Length;
5.      Stream s;
6.      s = req.GetRequestStream();
7.      StreamWriter sw = new StreamWriter(s,Encoding.ASCII);
8.      sw.Write(content);
9.      sw.Close();
10.
11.     HttpResponse res = (HttpResponse)↵
           req.GetResponse();
12.     s = res.GetResponseStream();
13.
14.     StreamReader sr = new StreamReader(s,Encoding.ASCII);
15.     StringBuilder sb = new StringBuilder();
16.     char [] data = new char[1024];
17.     int nBytes;
18.     do {
19.         nBytes = sr.Read(data,0,(int)1024);
20.         sb.Append(data);
21.     } while (nBytes == 1024);

```

**Visual Basic .NET code example**

```

1.      Dim req As HttpRequest = CType(WebRequest.Create(url), ↵
           HttpRequest)
2.      req.ContentType = contentType
3.      req.Method = method
4.      req.ContentLength = content.Length
5.      Dim s As Stream
6.      s = req.GetRequestStream()
7.      Dim sw As New StreamWriter(s, Encoding.ASCII)
8.      sw.Write(content)
9.      sw.Close()
10.
11.     Dim res As HttpResponse = CType(req.GetResponse(), ↵
           HttpResponse)
12.     s = res.GetResponseStream()
13.
14.     Dim sr As New StreamReader(s, Encoding.ASCII)
15.     Dim sb As New StringBuilder()
16.     Dim data(1024) As Char
17.     Dim nBytes As Integer
18.     Do
19.         nBytes = sr.Read(data, 0, CInt(1024))
20.         sb.Append(data)
21.     Loop While nBytes = 1024

```

**Code explanation**

The functionality that the preceding code implements is described in the following list. Please note that each line reference to the code applies to both the C# and Visual Basic .NET examples.

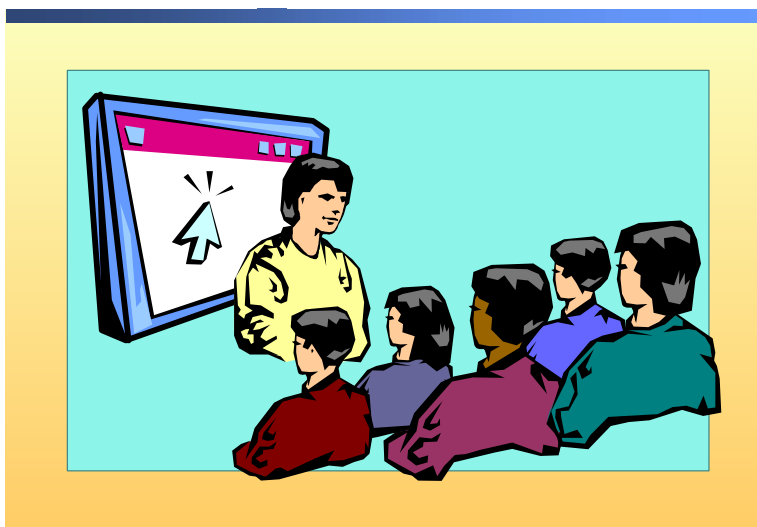
- In line 1, the return value of the **WebRequest.Create** call is converted to **HttpWebRequest**.

In most cases, the **WebRequest** and **WebResponse** classes provide all of the functionality that you need to perform an HTTP request. However, if you need to access HTTP-specific features such as HTTP headers, you need a protocol-specific derived class of **WebRequest**.

- In lines 2 through 4, HTTP-specific properties are set.
- In lines 6 through 9, the content for the request is written to a stream.  
Note in line 7 that the type of encoding is specified for the stream.
- In line 11, the response from the server is retrieved.
- In lines 12 through 21, the content of the response message is read.

Because the response stream is not seekable, the total amount of data to be read cannot be determined at the start of the content retrieval. As a result, the content is retrieved in blocks.

## Code Walkthrough: Issuing an Asynchronous HTTP Request



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this code walkthrough, you will look at how an asynchronous HTTP request is issued by using the .NET Framework.

This topic examines the functionality that the following sample code for an asynchronous request implements. (A C# and a Visual Basic .NET example are provided.)

**C# code example**

```
1.      Stream s;
2.      HttpWebRequest req = (HttpWebRequest )↵
           WebRequest.Create(url);
3.      req.ContentType=contentType;
4.      req.Method = method;
5.      req.ContentLength=content.Length;
6.      s = req.GetRequestStream();
7.      StreamWriter sw = new StreamWriter(s);
8.      sw.Write(content);
9.      sw.Close();
10.
11.     Handler h = new Handler();
12.     AsyncCallback callback = new ↵
           AsyncCallback(h.Callback)
13.     // Pass the request object as the state object
14.     req.BeginGetResponse(callback, req);
15.
16.     ...
17.
18.     public class Handler
19.     {
20.         public void Callback(IAsyncResult ar) {
21.             // Get the WebRequest from RequestState.
22.             HttpWebRequest req = (HttpWebRequest)↵
                 ar.AsyncState;
23.             // Get the response object associated
24.             // with the request.
25.             HttpWebResponse res = (HttpWebResponse)↵
                 req.EndGetResponse(ar);
26.             // Start reading data from the response stream.
27.             Stream s = res.GetResponseStream();
28.
29.             StreamReader sr = new ↵
                 StreamReader(s,Encoding.ASCII);
30.             StringBuilder sb = new StringBuilder();
31.             char [] data = new char[1024];
32.             int nBytes;
33.             do {
34.                 nBytes = sr.Read(data,0,(int)1024);
35.                 sb.Append(data);
36.             } while (nBytes == 1024);
37.             ...
38.             // continue processing
39.         }
40.     }
```

**Visual Basic .NET code example**

```

1.      Dim s As Stream
2.      Dim req As HttpWebRequest = CType(WebRequest.Create(url), ↵
                                         HttpWebRequest)
3.      req.ContentType = contentType
4.      req.Method = method
5.      req.ContentLength = content.Length
6.      s = req.GetRequestStream()
7.      Dim sw As New StreamWriter(s)
8.      sw.Write(content)
9.      sw.Close()
10.
11.     Dim h As New Handler()
12.     Dim callback As New AsyncCallback(h.Callback)
13.     ' Pass the request object as the state object
14.     req.BeginGetResponse(callback, req)
15.
16.     ...
17.
18. Public Class Handler
19.
20.     Public Sub Callback(ar As IAsyncResult)
21.         ' Get the WebRequest from RequestState.
22.         Dim req As HttpWebRequest = CType(ar.AsyncState, ↵
                                             HttpWebRequest)
23.         ' Get the response object associated
24.         ' with the request.
25.         Dim res As HttpWebResponse = ↵
                                             CType(req.EndGetResponse(ar), HttpWebResponse)
26.         ' Start reading data from the response stream.
27.         Dim s As Stream = res.GetResponseStream()
28.
29.         Dim sr As New StreamReader(s, Encoding.ASCII)
30.         Dim sb As New StringBuilder()
31.         Dim data(1024) As Char
32.         Dim nBytes As Integer
33.         Do
34.             nBytes = sr.Read(data, 0, CInt(1024))
35.             sb.Append(data)
36.         Loop While nBytes = 1024
37.         ...
38.         ' continue processing
39.     End Sub 'Callback
40. End Class 'Handler

```

The functionality that the preceding code implements is described in the following list. Please note that each line reference to the code applies to both the C# and Visual Basic .NET examples.

- In lines 1 through 9, an HTTP request is set up exactly the way it is done in a synchronous operation.
- In line 11, an instance of a custom class named **Handler** is created.  
This class will be used to handle the asynchronous completion of the HTTP request.
- In line 12, an instance of a delegate of type **AsyncCallback** is created, and a reference to the **Callback** method of the **Handler** class is passed to the constructor of the delegate.
- In line 14, an asynchronous request is initiated for a response by using the **BeginGetResponse** method.  
A reference to the delegate and a reference to an object that contains any state that might be needed by the method that handles the completion of the request are passed as parameters. In line 14, the request object is passed.
- In line 20, the **Callback** function receives a reference to an **IAsyncResult** interface as a parameter.
- In line 26, the asynchronous request is completed.
- In lines 29 through 39, the response content is retrieved exactly in the way it is done in a synchronous operation.

# XML Essentials

- Overview of XML
- XSD Fundamentals

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

XML is used to implement XML Web services in a number of ways. This includes the use of XML as the wire format between an XML Web service consumer and the XML Web service, and the use of XML to describe the XML Web service interface, etc. It is recommended that the XML Web service developer have a solid understanding of XML.

This topic does not attempt to teach fundamental XML skills. Instead the topic focuses on how you can describe data types by using XML schemas, and how you can control XML serialization in the .NET Framework.

## Overview of XML

- Elements and attributes
- Well-formed documents
- Schemas

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Considering the central role that XML plays in XML Web services, it is useful to review some of its important concepts.

### Elements and attributes

After the document prologue, all XML documents have a root element with child elements. Any of the elements may have attributes that provide further information about a particular element. A common source of confusion is when to use elements versus when to use attributes. There are no absolute rules for this choice. However, the following table summarizes and contrasts some of the most important characteristics of elements and attributes.

Characteristics	Elements	Attributes
May have child nodes	✓	✗
Are ordered	✓	✗
May be repeated	✓	✗
May be indexed	✓	✗
May be typed	✓	✓
May have a default value	✗	✓

When describing the data that your XML Web service consumes or returns, it is important to know the differences between elements and attributes so that you can use them appropriately in your XML documents.



**Well-formed documents**

All XML documents must be well-formed. For a document to be well-formed, it must adhere to the following rules:

- There must be a single root element.  
XML documents are trees, and not forests.
- All elements must be closed, unlike HTML, where many elements (example: `<BR>`) are not required to be closed.
- Capitalization of opening and closing tags of elements must be consistent. Many browsers allow inconsistent casing when using HTML elements (example: `<table>...</TABLE>`), but inconsistent casing is not allowed in XML.
- Elements must be nested correctly.
- Attribute values must be enclosed in quotes. Many browsers allow attribute values to be unquoted, but unquoted attribute values are not allowed in XML.
- An attribute cannot be repeated in an element.

Now that you have reviewed some of the important concepts of XML, the remainder of this topic will focus on how XML is used in XML Web services.

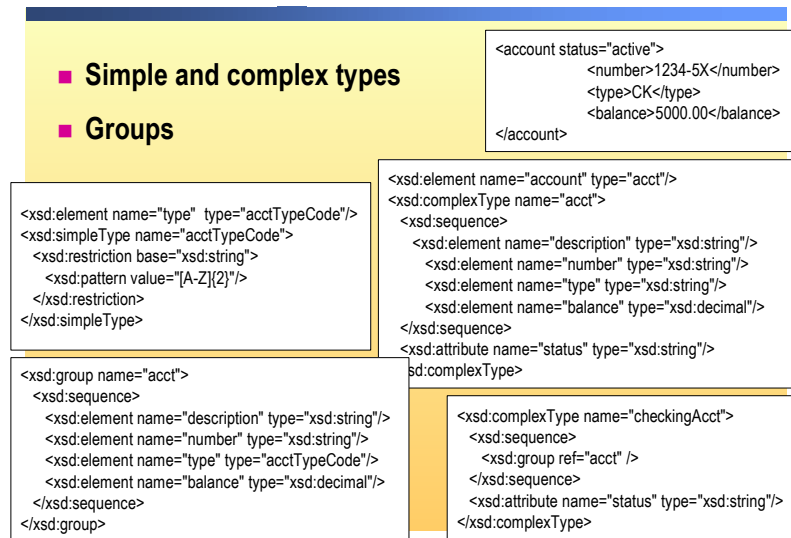
**Schemas**

To successfully use an XML Web service, you need to know the operations that the XML Web service supports and the structure of the documents (or messages) that each operation consumes and produces. This information is defined in a document, known as a service description, which describes an XML Web service. The service description is created by using the Web Service Description Language (WSDL), which is an XML-based language.

Within the WSDL documents, you define XSD schemas that describe the data types and document structures that are allowed in XML documents. XSD schemas validate XML documents in a mechanical way. This frees the programmer from the error-prone task of correctly parsing and validating a complex document structure.

You will learn the basics of XSD later in this module. For more information about WSDL, see Module 4, “Consuming XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

## XSD Fundamentals



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

One of the most important activities involved in designing and implementing XML Web services is specifying data types that are passed to and returned by an XML Web service. You must define data types unambiguously in the specifications. The XSD language is best suited for defining such document specifications. This topic will focus on some of the key features of XSD.

**Note** This topic is intended only to provide a brief introduction of some of the major features of XSD. The complete specifications can be found at <http://www.w3c.org/XML/Schema>.

### Simple and complex types

An XML schema can consist of elements that are simple types or complex types. A complex type can contain child elements in addition to attributes in its content. A simple type can contain neither child elements nor attributes in its content.

Consider the following XML code:

```
<account status="active">
  <number>1234-5X</number>
  <type>CK</type>
  <balance>5000.00</balance>
</account>
```

The preceding code can be represented as a complex type in an XML schema, as shown in the following code:

```
1. <xsd:element name="account" type="acct"/>
2.
3. <xsd:complexType name="acct">
4.   <xsd:sequence>
5.     <xsd:element name="description" type="xsd:string"/>
6.     <xsd:element name="number" type="xsd:string"/>
7.     <xsd:element name="type" type="xsd:string"/>
8.     <xsd:element name="balance" type="xsd:decimal"/>
9.   </xsd:sequence>
10.   <xsd:attribute name="status" type="xsd:string"/>
11. </xsd:complexType>
```

In the preceding example, you can further constrain the element named **type** to restrict it to a 2-character code that is made up of only upper-case letters. You can do this by defining a simple type and redefining the **type** element as follows:

```
1. <xsd:element name="type" type="acctTypeCode"/>
2.
3. <xsd:simpleType name="acctTypeCode">
4.   <xsd:restriction base="xsd:string">
5.     <xsd:pattern value="[A-Z]{2}"/>
6.   </xsd:restriction>
7. </xsd:simpleType>
```

## Groups

When designing the structure of a document, it can be useful to define groups of elements or attributes that can be used in the definition of many different complex types. For example, you might want to define different types of accounts such as checking, savings, credit card, etc. It would be inconvenient to repeatedly list the common elements in each account type in the type definition. In such situations, XSD groups are useful.

In continuation with the preceding example, you can define a group of common elements for all types of accounts as follows:

```
1. <xsd:group name="acct">
2.   <xsd:sequence>
3.     <xsd:element name="description" type="xsd:string"/>
4.     <xsd:element name="number" type="xsd:string"/>
5.     <xsd:element name="type" type="acctTypeCode"/>
6.     <xsd:element name="balance" type="xsd:decimal"/>
7.   </xsd:sequence>
8. </xsd:group>
```

You can use the preceding XSD group to define different account types as shown in the following code:

```
1.      <xsd:complexType name="checkingAcct">
2.          <xsd:sequence>
3.              <xsd:group ref="acct" />
4.          </xsd:sequence>
5.          <xsd:attribute name="status" type="xsd:string"/>
6.      </xsd:complexType>
7.
8.      <xsd:complexType name="savingsAcct">
9.          <xsd:sequence>
10.             <xsd:group ref="acct" />
11.             <xsd:element name="minimumBalance" type="xsd:decimal" />
12.          </xsd:sequence>
13.          <xsd:attribute name="status" type="xsd:string"/>
14.      </xsd:complexType>
```

## XSD Fundamentals (*continued*)

### ■ Compositors

- xsd:sequence
- xsd:choice
- xsd:all

### ■ Derivation

- restriction
- extension

```
<xsd:complexType name="customerName">
</xsd:complexType>

<xsd:complexType name="account">
  <xsd:sequence>
    <xsd:element name="description" type="xsd:string"/>
    <xsd:element name="number" type="xsd:string"/>
    <xsd:element name="type" type="acctTypeCode"/>
    <xsd:element name="balance" type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="status" type="xsd:string" />
</xsd:complexType>

<xsd:complexType name="savingsAcct">
  <xsd:complexContent>
    <xsd:extension base="account" >
      <xsd:sequence>
        <xsd:element name="minimumBalance" type="xsd:decimal" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In addition to supporting simple and complex types and groups, XSD also supports two features: compositors and derivation.

## Compositors

Compositors are an element that specifies a sequential (**xsd:sequence**), disjunctive (**xsd:choice**), or conjunctive (**xsd:all**) interpretation of the contents of the compositor. XSD provides the following compositors:

### ■ **xsd:sequence**

You use the **xsd:sequence** compositor to define an ordered sequence of child elements. In the previous example, in the topic **Groups**, **xsd:sequence** defines an ordered sequence of child elements in a complex type.

### ■ **xsd:choice**

You use the **xsd:choice** compositor to define a list of choices, which can be a set of elements, groups, or compositors. You can use the **xsd:choice** compositor in a schema to specify that the XML document that the schema validates can contain one of the choices from a given list.

For example, the **xsd:choice** compositor in the following schema code specifies that the XML document that the schema validates can contain either a `fullname` element or a sequence of elements that describes a customer name:

```
1. <xsd:complexType name="customerName">
2.   <xsd:choice>
3.     <xsd:element name="fullname" type="xsd:string" />
4.     <xsd:sequence>
5.       <xsd:element name="firstname" type="xsd:string" />
6.       <xsd:element name="middleinitial"
7.         type="xsd:string" minOccurs="0" />
8.       <xsd:element name="lastname" type="xsd:string" />
9.     </xsd:sequence>
10.   </xsd:choice>
11. </xsd:complexType>
```

### ■ **xsd:all**

The **xsd:all** compositor defines an unordered list of elements, groups, or compositors. For example, the **xsd:all** compositor in the following schema code specifies that the XML document that the schema validates can contain the checking, savings, and credit card elements in any order:

```
1. <xsd:complexType name="listOfAccts">
2.   <xsd:all minOccurs="0" maxOccurs="unbounded">
3.     <xsd:element ref="checking" minOccurs="0" />
4.     <xsd:element ref="savings" minOccurs="0" />
5.     <xsd:element ref="creditcard" minOccurs="0" />
6.   </xsd:all>
7. </xsd:complexType>
```

## Derivation

Another powerful feature of XSD is the ability to define new types by derivation. There are two ways of using derivation to define new types:

### ■ restriction

Defining new types by using restriction involves further constraining elements and attributes of the base type. For example, numeric values might be restricted to a smaller range than in the base type.

### ■ extension

Defining new types by using extension involves adding new elements to the derived type. The derived type now has all of the elements that are added in the derived type definition in addition to the elements that are defined in the base type.

In the earlier example on simple types, you saw a new simple type named **accTypeCode** that was derived by using restriction. In the following example, a new type named **savingsAcct** is derived by extending the **account** type:

```
1. <xsd:complexType name="account">
2.   <xsd:sequence>
3.     <xsd:element name="description" type="xsd:string"/>
4.     <xsd:element name="number" type="xsd:string"/>
5.     <xsd:element name="type" type="accTypeCode"/>
6.     <xsd:element name="balance" type="xsd:decimal"/>
7.   </xsd:sequence>
8.   <xsd:attribute name="status" type="xsd:string" />
9. </xsd:complexType>
10.
11. <xsd:complexType name="savingsAcct">
12.   <xsd:complexContent>
13.     <xsd:extension base="account" >
14.       <xsd:sequence>
15.         <xsd:element name="minimumBalance"
16.                       type="xsd:decimal" />
17.       </xsd:sequence>
18.     </xsd:extension>
19.   </xsd:complexContent>
20. </xsd:complexType>
```

## XML Serialization in the .NET Framework

- **XmlRootAttribute**
- **XmlElementAttribute**
- **XmlAttributeAttribute**
- **XmlArrayAttribute**
- **XmlArrayItemAttribute**
- **Caveats**
  - POST/GET vs. SOAP
  - Property serialization

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

When implementing XML Web services by using Microsoft Visual Studio® .NET and the .NET Framework, it is convenient to define the service interface (the methods to be exposed) in terms of the native .NET data types. The native serialization format for the .NET data types is XML. However, sometimes the default mapping of objects to elements is not what you require. Therefore you must instruct the XML serializer on how to convert an object graph into the XML document structure that you require. The **System.Xml.Serialization** namespace in the .NET Framework provides classes that you can use to modify the way objects are serialized. For more information, see the topic “XmlSerializer Class” in the .NET Framework Class Library on Microsoft MSDN®.

### Code example of attribute classes

When you implement an XML Web service by using ASP.NET, you typically do not directly control the serialization of your objects. Instead, you rely on the attribute classes found in the **System.Xml.Serialization** namespace to control the way objects are serialized.



Consider the following code:

```

1. [XmlRoot("account")]
2. public class Acct
3. {
4.     [XmlElement("description")]
5.     public string Description;
6.     [XmlElement("number")]
7.     public string Number;
8.     [XmlElement("type")]
9.     public string Type;
10.    [XmlElement("balance")]
11.    public decimal Balance;
12.    [XmlAttribute("status")]
13.    public string Status;
14. }
15. ...
16. [return:XmlArray("AccountList")]
17. [return:XmlArrayItem("Account")]
18. public Acct[] GetAllAccounts()
19. ...

```

The attribute classes that are used in the preceding code are explained in the following paragraphs.

---

**Note** In the .NET Framework, attribute class names have the format *XXXXAttribute*. However using the .NET compilers makes it unnecessary for developers to use the **Attribute** suffix in their code. Therefore, the .NET developer can refer to “the *XXXX* attribute” instead of the *XXXXAttribute* class without ambiguity.

---

#### The **XmlRootAttribute** class

Every XML document must have a single root element. The **XmlRoot** attribute allows you to control how the root element is generated by setting certain properties. For example, you can specify the name of the generated XML element by setting the **ElementName** property. You can apply the **XmlRoot** attribute to classes only.

#### The **XmlElementAttribute** class

You can apply the **XmlElement** attribute to public fields or public properties to control the characteristics of XML elements, such as the element name and namespace.

If you apply the **XmlElement** attribute to a field or property that returns an array, the items in the array are generated as a sequence of XML elements. However, if you do not apply the **XmlElement** attribute to such a field or property, the items in the array are generated as a sequence of child elements, nested under an element, which is named after the field or property.

#### The **XmlAttributeAttribute** class

By default, the XML serializer serializes public fields and properties as XML elements. When you apply the **XmlAttribute** attribute to a public field or property, the XML serializer serializes the member as an XML attribute.

XML attributes can only be simple types. Therefore, you can apply the **XmlAttribute** attribute only to public fields or properties that return a primitive type.

**The XmlArrayAttribute class**

When you apply the **XmlArray** attribute class to a class member, the XML serializer generates a nested sequence of XML elements from that member. For example, if a class that is to be serialized represents a bank's customer, then you can generate an array of accounts that the customer owns by applying the **XmlArray** attribute to a public field that returns an array of objects that represent the accounts. If you apply the **XmlArray** attribute to a field or property that returns an array, then by default, the name of the generated XML element is derived from the member identifier. However, by setting the **ElementName** property of the **XmlArray** attribute, you can change the name of the generated XML element.

**The XmlArrayItemAttribute class**

To more precisely control the XML element generation for the members of an array, you can use the **XmlArrayItem** class. Using **XmlArrayItem** also allows you to ensure that *polymorphic* arrays (arrays containing derived objects of the base array type) are correctly serialized. For example, suppose that a class named **account** exists and two other classes named **checkingAcct** and **savingsAcct** respectively that are derived from **account** also exist. Further, suppose that a class named **bankCustomer** has a field that returns an array of **account** objects. To allow the **XmlSerializer** class to serialize both the **checkingAcct** and **savingsAcct** classes, apply the **XmlArrayItem** to the field twice, once for each of the two acceptable types.

There are many other attributes that you can use to control the format of a serialized object. For more information, see the documentation for the **XmlXXXXAttribute** classes in the **System.XML** namespace.

**Caveats**

In the context of XML, the .NET Framework, and XML Web services, there are a few caveats to keep in mind.

- Use of POST/GET versus SOAP

Currently, when you use **[return:XmlArrayItem]**, the name of the array item is modified when you use SOAP, but not when you use GET or POST. Therefore, the generated XML document will be different depending on whether the XML Web service consumer uses POST/GET or SOAP. The following code shows how to control the names of the XML elements that are emitted when an array is serialized:

```
[return:XmlArrayItem(ElementName="savingsAcct", ↵
                    Type=typeof(SavingsAcct))]
[return:XmlArrayItem(ElementName="creditCardAcct", ↵
                    Type=typeof(CreditCardAcct))]
[WebMethod]
public Acct[] GetAllAccounts()
{
    ...
}
```

- Property serialization

When an object is serialized, only public read/write properties are serialized. In other words, there is no way to serialize a read-only property (a property with only a **get** accessor).

# SOAP Fundamentals

- Overview of SOAP
- Structure of SOAP messages
- Code Walkthrough: Invoking an XML Web service method using SOAP

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Simple Object Access Protocol (SOAP) is a lightweight protocol for the exchange of information in decentralized, distributed environments. It is an XML-based protocol that consists of two parts:

- An envelope for handling extensibility and modularity.
- An encoding mechanism for representing types within an envelope.

You can potentially use SOAP in combination with a variety of other protocols. However, the only protocol bindings currently defined are for HTTP and HTTP Extension Framework (HTTP-EF).

## Overview of SOAP

- SOAP messages
- Parts of a SOAP message
  - SOAP envelope
  - SOAP encoding rules
  - SOAP RPC representation
  - Protocol bindings for HTTP and HTTP-EF

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

Exchanging SOAP messages provides a useful way for communicating with XML Web services. However, before you can implement and consume XML Web services that communicate through SOAP, it is important that you understand the structure of SOAP messages and the basic operation of the protocol.

### SOAP messages

SOAP messages are fundamentally one-way transmissions from a sender to a receiver. SOAP does not define any application semantics such as a programming model or implementation-specific semantics. However, XML Web services require a request/response model. A solution is to send SOAP messages in the body of an HTTP request and response. This solution provides the required model for XML Web services.

You can optimize SOAP implementations to exploit the unique characteristics of particular network systems. For example, the HTTP binding provides for SOAP request messages to be sent out as part of an HTTP request, and the SOAP response messages to be delivered as HTTP responses, using the same connection as the outbound request.

**Parts of a SOAP message**

SOAP consists of four parts:

- The SOAP envelope, which defines what is in a message, who should process the message, and whether the message is optional or mandatory.
- The SOAP encoding rules, which define a serialization mechanism for exchanging instances of application-defined data types.
- The SOAP RPC representation, which defines a convention for representing remote procedure calls and responses.
- Protocol bindings that describe how to use SOAP in HTTP either with or without the HTTP-EF.

As a developer, you should be familiar with the details of a SOAP envelope. However, you can ignore the encoding and RPC details of the SOAP message because the .NET Framework handles these details. You will briefly examine protocol bindings when you look at the .NET Framework support for SOAP later in this module.

## Structure of SOAP Messages

- SOAP Envelope
- SOAP Header
- SOAP Body
- SOAP Fault

```

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <WoodgroveAuthInfo xmlns="http://tempuri.org/">
      <Username>string</Username>
      <Password>string</Password>
    </WoodgroveAuthInfo>
  </soap:Header>
  <soap:Body>
    <GetAccount xmlns="http://tempuri.org/">
      <acctID>int</acctID>
    </GetAccount>
  </soap:Body>
</soap:Envelope>

```

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

A SOAP message consists of the **Envelope** element at the root, which in turn consists of a mandatory **Body** element and an optional **Header** element.

### SOAP Envelope

The **Envelope** element is the root node of an XML document that represents the SOAP message. It contains the **Header** and **Body** elements and is mandatory.

### SOAP Header

The **Header** element is the first immediate child element of the SOAP **Envelope** element. All immediate child elements of the **Header** element are known as header entries.

The **Header** element provides a generic means for adding features to a SOAP message in a decentralized manner without prior agreement between the communicating parties. Headers allow us to provide extended information about a message. Typical uses of header entries are authentication, transaction management, payment, etc.

### SOAP Body

If the SOAP **Envelope** element does not contain the **Header** element, then the **Body** element must be the first immediate child element of the **Envelope**. If the **Header** element is present, then the **Body** element must immediately follow the **Header** element.

All immediate child elements of the **Body** element are called body entries and each body entry is a separate element within the SOAP **Body** element.

In the context of XML Web services, the **Body** element comprises the data specific to the actual method call, such as the XML Web service method name and parameters and/or return values for the method invocation.

### SOAP Fault

The SOAP **Fault** element is used to transport error or status information or both, within a SOAP message. If the SOAP **Fault** element is present, it must be a body entry and may not appear more than once within the **Body** element.

The SOAP **Fault** element defines the following four child elements:

■ **faultcode**

The **faultcode** element is intended for use by the XML Web service consumer to identify the fault. This element must be present within the SOAP **Fault** element. SOAP defines a small set of SOAP fault codes that cover the basic SOAP faults.

■ **faultstring**

The purpose of the **faultstring** element is to provide a human-readable explanation of the fault. It must be present within the SOAP **Fault** element and must provide information explaining the nature of the fault.

■ **faultactor**

The purpose of the **faultactor** element is to provide information about who caused the fault to happen within the message path. It indicates the source of the fault. The value of the **faultactor** element is a URI that identifies the source. Applications that are not the ultimate destination of the SOAP message must include the **faultactor** element in the SOAP **Fault** element.

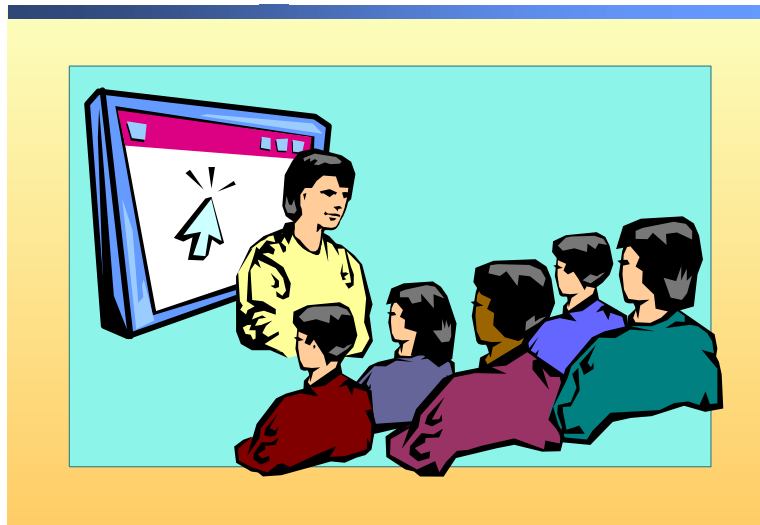
■ **detail**

The **detail** element is for holding application-specific error information related to the **Body** element. It is included if the contents of the **Body** element could not be successfully processed. The absence of the **detail** element within the **Fault** element indicates that the fault is not related to processing of the **Body** element.

The following example code shows a SOAP fault message that might be generated when you attempt to withdraw money from a bank account:

```
1. HTTP/1.0 500 Internal Server Error
2. Content-Length: 460
3. Content-Type: text/xml; charset="utf-8"
4.
5. <soap:Envelope
6.   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
7.   <soap:Body>
8.     <soap:Fault>
9.       <faultcode>123XYZ</faultcode>
10.      <faultstring>Server Error</faultstring>
11.      <detail>
12.        <bank:faultdetails xmlns:bank="urn:OnlineBank">
13.          <message>Your account is overdrawn</message>
14.          <errorcode>1234</errorcode>
15.        </bank:faultdetails>
16.      </detail>
17.    </soap:Fault>
18.  </soap:Body>
  </soap:Envelope>
```

## Code Walkthrough: Invoking an XML Web Service Method Using SOAP



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this code walkthrough, you will look at how to invoke an XML Web service method by using SOAP and the .NET Framework.

### Code example: invoking the **GetAccount** method

This topic will examine the functionality that the following sample code implements for using SOAP to invoke a method named **GetAccount**.

```

1. POST /dummy/service1.asmx HTTP/1.1
2. Host: 192.168.0.80
3. Content-Type: text/xml; charset=utf-8
4. Content-Length: 215
5. SOAPAction: "http://woodgrovebank.com/GetAccount"
6.
7. <?xml version="1.0" encoding="utf-8"?>
8. <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9.   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
10.   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
11.   <soap:Body>
12.     <GetAccount xmlns="http://woodgrovebank.com">
13.       <acctNumber>1234</acctNumber>
14.     </GetAccount>
15.   </soap:Body>
16. </soap:Envelope>
17.

```

In the preceding code, the endpoint of the **GetAccount** method is specified in the **SOAPAction** header, and the method and its parameters are contained in the **soap:Body** element.



**Code example: HTTP response**

The response to the preceding method invocation is returned in an HTTP response. The response is in the form of an XML document in the **soap:Body** element.

```
1. HTTP/1.1 200 OK
2. Content-Type: text/xml; charset=utf-8
3. Content-Length: 247
4.
5. <?xml version="1.0" encoding="utf-8"?>
6. <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7.     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
8.     xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
9.   <soap:Body>
10.    <GetAccountResponse xmlns="http://woodgrovebank.com">
11.      <savingsAcct>
12.        <balance>5250.00</balance>
13.      </savingsAcct>
14.    </GetAccountResponse>
15.  </soap:Body>
16. </soap:Envelope>
17.
```

## Using SOAP with the .NET Framework

- Controlling the SOAP message format
- Code Walkthrough: Issuing a SOAP request using the .NET Framework

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

When you implement an XML Web service by using ASP.NET, a sophisticated mechanism is available for controlling the format of the SOAP message that is sent to and returned by the server. As you saw earlier, the contents of the SOAP messages sent to and from an XML Web service are in XML. However, the encoding of the XML is not strictly defined.

In this section, you will learn about the encoding definitions for XML Web services that are part of the SOAP specification. You will also learn about different encoding styles that you can use to format the parameters to an XML Web service method.

## Controlling the SOAP Message Format

- SOAP encoding definitions
- Formatting the SOAP body
- Formatting parameters

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

The .NET Framework provides attributes that can control the format of the XML inside a SOAP message to facilitate working with XML Web services that expect different encoding styles. It is the responsibility of the XML Web service consumer to encode the XML appropriately. For more information about how the consumer can control encoding, see Module 4, “Consuming XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

### SOAP encoding definitions

The SOAP specification states two distinct encoding definitions that are related to XML Web services. You can find these definitions in Section 7 and Section 5 of the SOAP specification (see [http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#\\_Toc478383512](http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383512)). These definitions outline rules on:

- How the SOAP **Body** element must be overall formatted.
- How parameters within a method must be encoded.

Both of the preceding encoding rules are optional. Therefore, ASP.NET Web Services supports SOAP requests and responses that use both of these encoding rules, along with other variants.

### Formatting the SOAP body

You can format an XML Web service method within the **Body** element of a SOAP request or a SOAP response by using either RPC encoding or document encoding. ASP.NET Web Services support both the RPC and document encoding styles, with document encoding being the default style.

- **RPC encoding**

The RPC-encoding style formats the **Body** element according to Section 7 of the SOAP specification. This section explains how to use SOAP for RPC. In the RPC-encoding style, all parameters are wrapped within a single element that is named after the XML Web service method and each element within that element represents a parameter named after its respective parameter name.

- **Document encoding**

The document-encoding style formats the **Body** element as described in an XSD schema. If you use the document-encoding style, the service description for the XML Web service defines the XSD schemas for both SOAP requests and SOAP responses to the XML Web service method. These schemas are part of WSDL documents, which are explained in detail in Module 4, “Consuming an XML Web Service,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

In the document-encoding style, clients must send an XML document to an XML Web service exactly in the format specified in the associated XSD schemas.

### **Formatting parameters**

Because the parameters to an XML Web service method can make up the bulk of the data passed in a SOAP request or response, how the parameters are encoded determines how the XML document will look. There are two encoding styles for parameters:

- **Encoded**

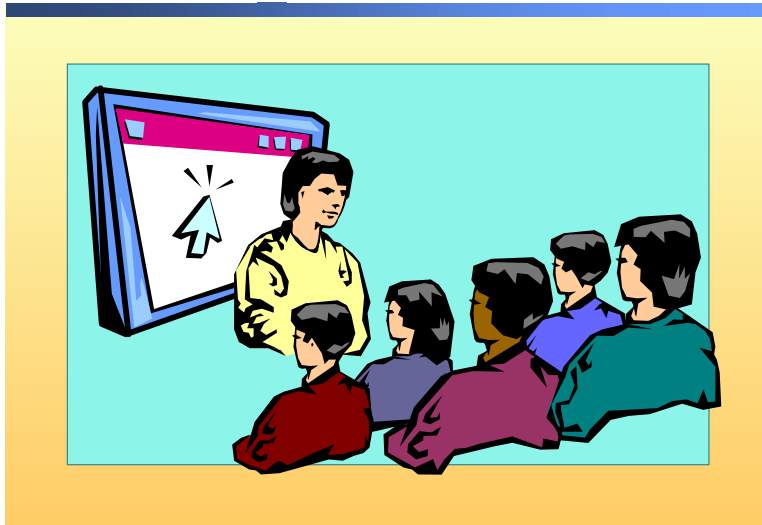
This style encodes the parameters by using the SOAP-encoding rules outlined in Section 5 of the SOAP specification.

- **Literal**

This style encodes each parameter according to a predefined XSD schema.

You will see how to control encoding of the SOAP document for XML Web services and the format of parameters in Module 5, “Implementing a Simple XML Web Service,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

## Code Walkthrough: Issuing a SOAP Request Using the .NET Framework



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this code walkthrough, you will look at how to issue a SOAP request by using the .NET Framework.

The following code shows a sample XML document, which is used to invoke an XML Web service method by using the document-encoding style for the **Body** element and the Literal-encoding style for the parameters.

**Code example:**  
**Encoding using**  
**Document and Literal**  
**styles**

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope namespaces omitted for brevity>
  <soap:Body>
    <GetAccount xmlns="http://tempuri.org/">
      <acctID>int</acctID>
    </GetAccount>
  </soap:Body>
</soap:Envelope>
```

**C# example**

The following code shows how the preceding message can be constructed.

```
1. public static string BuildSOAPMessage()
2. {
3.     MemoryStream st;
4.     st = new MemoryStream(1024);
5.
6.     XmlTextWriter tr = new XmlTextWriter(st, Encoding.UTF8);
7.     tr.WriteStartDocument();
8.     tr.WriteStartElement("soap", "Envelope", ↵
9.         "http://schemas.xmlsoap.org/soap/envelope/");
10.    tr.WriteAttributeString("xmlns", "xsi", null, ↵
11.        "http://www.w3.org/2001/XMLSchema-instance");
12.    tr.WriteAttributeString("xmlns", "xsd", null, ↵
13.        "http://www.w3.org/2001/XMLSchema");
14.    tr.WriteAttributeString("xmlns", "soap", null, ↵
15.        "http://schemas.xmlsoap.org/soap/envelope/");
16.
17.    tr.WriteStartElement("Body", ↵
18.        "http://schemas.xmlsoap.org/soap/envelope/");
19.    tr.WriteStartElement(null, "GetAccount", "http://woodgrovebank
20.        .com");
21.    tr.WriteElementString("acctNumber", "1234");
22.    tr.WriteEndElement();
23.    tr.WriteEndElement();
24.    tr.WriteEndDocument();
25.    tr.Flush();
26.    ...
}
```

**Visual Basic .NET code example**

```
1.      Public Shared Function BuildSOAPMessage() As String
2.
3.      Dim st As MemoryStream
4.      st = New MemoryStream(1024)
5.
6.
7.      Dim tr As New XmlTextWriter(st, Encoding.UTF8)
8.      With tr
9.          .WriteStartDocument()
10.         .WriteStartElement("soap", "Envelope",
            ↳ "http://schemas.xmlsoap.org/soap/envelope/")
11.         .WriteAttributeString("xmlns", "xsi", Nothing,
            ↳ "http://www.w3.org/2001/XMLSchema-instance")
12.         .WriteAttributeString("xmlns", "xsd", Nothing,
            ↳ "http://www.w3.org/2001/XMLSchema")
13.         .WriteAttributeString("xmlns", "soap", Nothing,
            ↳ "http://schemas.xmlsoap.org/soap/envelope/")
14.
15.         .WriteStartElement("Body",
            ↳ "http://schemas.xmlsoap.org/soap/envelope/")
16.         .WriteStartElement(Nothing, "GetAccount",
            ↳ "http://woodgrovebank.com")
17.         .WriteElementString("acctNumber", "1234")
18.         .WriteEndElement()
19.         .WriteEndElement()
20.         .WriteEndDocument()
21.         .Flush()
22.     End With
23.     ...
24. End Function
```

**C# code example**

The following code shows how you can issue the preceding SOAP request by using the .NET Framework:

```
1.      public static string GetSOAPData(string url,
2.                                         ↪string action,string content)
3.      {
4.          Stream s;
5.          HttpWebRequest req = (HttpWebRequest)↪
6.                                  WebRequest.Create(url);
7.          string hdr = "SOAPAction: \"http://sftsrc.com/\"↪
8.                                  + action + "\"";
9.          req.Headers.Add(hdr);
10.         req.ContentType="text/xml; charset=utf-8";
11.         req.Method = "POST";
12.         if (content.Length > 0)
13.         {
14.             req.ContentLength=content.Length;
15.             s = req.GetRequestStream();
16.             StreamWriter sw = new StreamWriter(s);
17.             sw.Write(content);
18.             sw.Close();
19.         }
20.         ...
21.         HttpWebResponse res = (HttpWebResponse)↪
22.                                 req.GetResponse();
23.         return GetResponseAsString(res);
24.         ...
25.     }
```



**Visual Basic .NET code  
example**

```
1.      Public Shared Function GetSOAPData(ByVal url As String,  
                                           ↳ByVal action As String,  
                                           ↳ByVal content As String) As String  
2.      Dim s As Stream  
3.      Dim req As HttpWebRequest =  
           ↳CType(WebRequest.Create(url), HttpWebRequest)  
4.      Dim hdr As String =  
           ↳"SOAPAction: \http://localhost/" + action + "\"  
5.      req.Headers.Add(hdr)  
6.      req.ContentType = "text/xml; charset=utf-8"  
7.      req.Method = "POST"  
8.      If content.Length > 0 Then  
9.          req.ContentLength = content.Length  
10.         s = req.GetRequestStream()  
11.         Dim sw As New StreamWriter(s)  
12.         sw.Write(content)  
13.         sw.Close()  
14.     End If  
15.  
16.     ...  
17.     Dim res As HttpWebResponse =  
           ↳CType(req.GetResponse(), HttpWebResponse)  
18.     Return GetResponseAsString(res)  
19.     ...  
20. End Function
```

**C# code example**

The following code shows how you can convert the preceding response message into a string:

```
1.      public static string GetResponseAsString(↵
2.                                         WebResponse res)
3.      {
4.          Stream s = res.GetResponseStream();
5.          StreamReader sr = new StreamReader(↵
6.                                         s,Encoding.ASCII);
7.          StringBuilder sb = new StringBuilder();
8.          char [] data = new char[1024];
9.          int nBytes;
10.         do
11.         {
12.             nBytes = sr.Read(data,0,(int)1024);
13.             sb.Append(data);
14.         } while (nBytes == 1024);
15.         return sb.ToString();
16.     }
```

**Visual Basic .NET code example**

```
1.      Public Shared Function GetResponseAsString(ByVal res As
2.                                         WebResponse) As String
3.
4.          Dim s As Stream = res.GetResponseStream()
5.          Dim sr As New StreamReader(s, Encoding.ASCII)
6.          Dim sb As New StringBuilder()
7.          Dim data(1024) As Char
8.          Dim nBytes As Integer = 1024
9.          Do
10.             nBytes = sr.Read(data, 0, CType(1024, Integer))
11.             sb.Append(data)
12.         While Bytes = 1024
13.             Return sb.ToString()
14.         End Function
```

## Lab 3.1: Issuing HTTP and SOAP Requests Using the .NET Framework



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Objectives

After completing this lab, you will be able to:

- Issue HTTP **POST** and **GET** request methods and process the responses by using the Microsoft .NET Framework.
- Construct a SOAP message by using the .NET Framework.
- Issue a SOAP request and process the response by using the .NET Framework.

---

**Note** This lab focuses on the concepts in this module and as a result may not comply with Microsoft security recommendations.

---

### Lab Setup

For each lab exercise in this course, you have the option of coding in either C# or VB .NET. In order to avoid unnecessary instructions, the following placeholders will be used throughout the lab exercises:

<install folder>

The install folder for the course content. Usually this will be the folder C:\Program Files\Msdntrain\2524.

<lab root>

The folder <installfolder>\Labfiles\CS or <install folder>\Labfiles\VB, depending of the language you are using for the lab exercises.

It is recommended that you select only one language for the lab exercises.

There are starter and solution files that are associated with this lab. The starter files are in the folder `<lab root>\Lab03\Starter`. The solution files for this lab are in the folder `<lab root>\Lab03\Solution`.

**Scenario**

In this lab, you will issue HTTP and SOAP requests to an XML Web service named Woodgrove Online Bank. Specifically, you will be constructing requests to invoke the **GetAccount** operation of the XML Web service.

**Estimated time to  
complete this lab: 45  
minutes**

## Exercise 0

### Setting up the Woodgrove XML Web Service

In this exercise, you will create the virtual directory for the Woodgrove XML Web service. You will use either the Visual Basic.NET or C# Woodgrove XML Web service depending on your programming language preference.

#### ► Set Up Woodgrove XML Web Service

1. Click Start, click Control Panel, click Performance and Maintenance, click Administrative Tools, and then double-click Internet Information Services.
2. Click the **plus** sign and expand the tree to **Default Web Site**.
3. Right-click **Default Web Site**, point to **New**, and then click **Virtual Directory**.
4. Complete the **Virtual Directory Creation Wizard** by using the information in the following table.

On this wizard page	Do this
<b>Welcome to the Virtual Directory Creation Wizard</b>	Click <b>Next</b> .
<b>Virtual Directory Alias</b>	In the <b>Alias</b> box, type <b>Woodgrove</b> and click <b>Next</b> .
<b>Web Site Content Directory</b>	In the <b>Directory</b> box, browse to <code>&lt;labroot&gt;\WebServicesSolution\Woodgrove</code> and click <b>Next</b> .
<b>Access Permissions</b>	Click <b>Next</b> .
<b>You have successfully completed the Virtual Directory Creation Wizard.</b>	Click <b>Finish</b> .

5. Close Internet Information Services.

## Exercise 1

### Creating the Base Project

In this exercise, you will create a console application named **Technology**. You will also add the required namespace declarations so that you can issue HTTP and SOAP requests. Finally, you will incorporate a helper function (which has been provided for you) that will assist in displaying the responses to your requests.

#### ► Create the **Technology** console application

1. Open Microsoft Visual Studio .NET.
2. Create a console application project named **Technology**.  
Set the Location of the project to be `<lab root>\Lab03`.  
Your project will be created at `<lab root>\Lab03\Technology`.

3. Open **Solution Explorer**.

4. Rename

Visual Basic .NET	C#
Module1.vb to Tester.vb	Class1.cs to Tester.cs

5. In **Solution Explorer**, open the renamed file.

6. Rename

Visual Basic .NET	C#
Module1 to Tester	Class1 to Tester

7. For students completing the lab using Microsoft Visual Basic® .NET, perform the following steps:
  - a. In **Solution Explorer**, right click the project name.
  - b. Click **Properties**.
  - c. In the **Startup object** combobox, select **Sub Main** and click **OK**.

#### ► Add the required namespaces

- Add instructions to import the following namespaces:  
System.Net  
System.IO  
System.Text  
System.Xml  
System.Collections.Specialized  
System.Diagnostics

---

► **Add the helper function**

1. Using Microsoft Windows® Explorer, open *<lab root>\Lab03\Starter\Tracing.txt*.
2. Locate the **Main** method.
3. Copy the code in the **Insert 1** section from Tracing.txt into the **Main** method.

---

**Note** The remainder of the code that you add to the **Main** method in this lab must be added immediately after the code just inserted.

---

4. Copy the functions in the **Insert 2** section from Tracing.txt after the **Main** method.

## Exercise 2

### Implementing a Method to Issue HTTP Requests

In this exercise, implement a method named **GetData** that will be used to issue HTTP requests. This method will be used for HTTP-GET, HTTP-POST, and SOAP requests.

#### ► Implement the **GetData** method

1. Add a method named **GetData**. The method must have the following signature:

Visual Basic .NET	C#
<pre>Public Sub GetData(ByVal url As String,     ↳ ByVal contentType As ↳String,     ↳ ByVal method As String,     ↳ ByVal content As String,     ↳ ByVal ParamArray headers() As String)</pre>	<pre>public static void GetData(string url,     ↳ string contentType,     ↳ string method,     ↳ string content,     ↳ params string[] headers)</pre>

2. Create an instance of the **HttpRequest** class by using the **HttpRequest.Create** method. Use the *url* parameter.
3. If there are any headers in the **headers** parameter, then add the headers to the **HttpRequest.Headers** collection.
4. If the length of the **method** parameter is greater than zero, then assign it to the **HttpRequest.Method** property.
5. If the length of the **contentType** parameter is greater than zero, then assign it to the **HttpRequest.ContentType** property.
6. If the length of the **content** parameter is greater than zero, then add the content to the request.
  - a. Assign the length of the **content** parameter to the **HttpRequest.ContentLength** property.
  - b. Obtain the request stream by using the **GetRequestStream** method of **HttpRequest**.
  - c. Create an instance of **StreamWriter** class that is associated with the request stream.
  - d. Write the content to the stream by using the **StreamWriter**.
  - e. Close the **StreamWriter**.
7. Call the **DisplayRequest** method that was added in exercise 1. Use the **HttpRequest** object created in step 2.
8. Use the **GetResponse** method of **HttpRequest** to issue the request and retrieve an instance of **HttpResponse**.
9. Call the **DisplayResponse** method that was added in exercise 1. Use the **HttpRequest** object retrieved in step 8.



## Exercise 3

### Issuing HTTP-GET and HTTP-POST Requests

In this exercise, you will add code to invoke the **GetAccount** operation of the XML Web service for the Woodgrove Online Bank by using HTTP-GET and HTTP-POST requests.

#### ► Issue an HTTP-GET request

1. Locate the **Main** method.
2. Add a local string variable with the URL that is required to invoke the **GetAccount** operation of the XML Web service for the Woodgrove Online Bank by using the **GET** method.

The endpoint is `http://Localhost/WoodGrove/Bank.asmx/GetAccount` and the querystring is `acctID=1`.

3. Invoke the **GetData** method.
  - a. Use the URL defined in step 2.
  - b. Specify the **GET** method.
  - c. All other parameters must be empty strings.

#### ► Issue an HTTP-POST request

1. Locate the **Main** method.
2. Add a local string variable with the URL required to invoke the **GetAccount** operation of the XML Web service for the Woodgrove Online Bank by using the **POST** method.

The endpoint is `http://Localhost/WoodGrove/Bank.asmx/GetAccount`.

3. Invoke the **GetData** method.
  - a. Use the URL defined in step 2.
  - b. Specify **application/x-www-form-urlencoded** as the content type.
  - c. Specify the **POST** method.
  - d. Set the content to be the string `acctID=1`.

## Exercise 4

### Completing the BuildSOAPMessage Method

In this exercise, you will complete the code for a method named **BuildSOAPMessage**.

► **Add the function stub**

1. Using Windows Explorer, open <lab root>\Lab03\Starter\BuildSOAP.txt.
2. Add the function stub for the **BuildSOAPMessage** method to.

---

**Visual Basic .NET**

**C#**

The Tester module in Technology.vb

The Tester class in Technology.cs

- a. At the insertion point indicated in the function stub for the **BUILDSOAPMESSAGE** method, do the following:
  - b. Use the **WriteStartElement** method to add an element named **GetAccount**.  
  
The namespace to be used is `http://Tempuri.org`. There is no namespace prefix for this element.
3. Use the **WriteElementString** method to add an element named **acctID** with the value **1**.
  4. Use the **WriteEndElement** method to close the **GetAccount** element.

## Exercise 5

### Issuing a SOAP Request

In this exercise, you will add code to invoke the **GetAccount** operation of the XML Web service for the Woodgrove Online Bank by using a SOAP request.

1. Locate the **Main** method.
2. Add a local string variable with the URL that is required to invoke the **GetAccount** operation of the XML Web service for the Woodgrove Online Bank by using the **POST** method.

The endpoint is `http://Localhost/WoodGrove/Bank.asmx`.

3. Add a local string variable with the required SOAPAction.

The action required is `http://tempuri.org/GetAccount`.

4. Invoke the **GetData** method.
  - a. Use the URL defined in step 2.
  - b. Specify `text/xml; charset=utf-8` as the content type.
  - c. Specify the **POST** method.
  - d. Set the content to be the result of calling **BuildSOAPMessage**.
  - e. Provide the SOAPAction header.

#### ► Test the application

1. Build and run the application.
2. Compare the inbound and outbound messages for each of the three request types.
3. Ensure that each of the different method calls returns the same results.

## Review

- HTTP Fundamentals
- Using HTTP with the .NET Framework
- XML Essentials
- XML Serialization in the .NET Framework
- SOAP Fundamentals
- SOAP Using the .NET Framework

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

1. How can parameter information be passed to a Web server when using the HTTP-GET protocol?

**Parameters can be passed by using the querystring part of the URL.**

2. If you need an XML entity that can have a default value, should you use an element or an attribute?

**Attribute**

3. Which XSD compositor defines an ordered list of elements?

**xsd:sequence**

4. How are errors reported to a client when using the SOAP protocol?

**By using SOAP Fault elements**

5. Which SOAP parameter encoding style entails the use of an XSD schema to encode the parameters?

**Literal**



---

## Module 4: Consuming XML Web Services

### Contents

Overview	1
WSDL Documents	2
XML Web Service Discovery	8
XML Web Service Proxies	19
Implementing an XML Web Service Consumer Using Visual Studio .NET	27
Lab 4.1: Implementing an XML Web Service Consumer Using Visual Studio .NET	34
Review	43



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, places or events 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, trademarks, 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.

© 2001–2002 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Active Directory, Authenticode, IntelliSense, FrontPage, Jscript, MSDN, PowerPoint, Visual C#, Visual Studio, Visual Basic, Windows NT, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.



# Instructor Notes

**Presentation:**  
**120 Minutes**

This module teaches students how to implement XML (Extensible Markup Language) Web service consumers by using Microsoft® Visual Studio® .NET.

**Lab:**  
**75 Minutes**

After completing this module, students will be able to:

- Explain the structure of a Web Service Description Language (WSDL) document.
- Explain the XML Web services discovery process.
- Locate service contracts by using Disco.exe.
- Generate XML Web service proxies by using Wsdll.exe.
- Implement an XML Web service consumer by using Visual Studio .NET.
- Invoke an XML Web service synchronously and asynchronously by using an XML Web service proxy.

**Required Materials**

To teach this module, you need the Microsoft PowerPoint® file 2524B\_04.ppt.

**Preparation Tasks**

To prepare for this module:

- Read all of the materials for this module.
- Try the demonstrations and study the code examples in this module.
- Complete the lab.

**Dual-language  
PowerPoint macros**

The PowerPoint file for this module contains macros that allow you to switch the displayed code between C# and Microsoft Visual Basic® .NET. To run the macros, you must install the full version of PowerPoint.

To switch a single slide to C#, perform the following steps:

1. Open the PowerPoint deck in PowerPoint.
2. On the **Slide Show** menu, click **View Show**.
3. Locate the slide that you want to switch to C#.
4. Click **C#** on the slide.

To switch a single slide to Visual Basic .NET, perform the following steps:

1. Open the PowerPoint deck in PowerPoint.
2. On the **Slide Show** menu, click **View Show**.
3. Locate the slide that you want to switch to Visual Basic .NET.
4. Click **Visual Basic .NET** on the slide.

---

**Note** You can switch a slide to C# or Visual Basic .NET at any time while displaying the slides. Just click **C#** or **Visual Basic .NET** to switch between the languages.

---

## Demonstration

This section provides demonstration procedures that will not fit in the margin notes or are not appropriate for the student notes.

### Locating Discovery Documents Using Disco.exe

#### ► To use Disco.exe to locate discovery documents

1. Click the Start menu, point to **All Programs**, point to **Microsoft Visual Studio .NET**, point to **Visual Studio .NET Tools**, and then click **Visual Studio .NET Command Prompt**.
2. At the command prompt, type the following command:  
**md c:\wsdl**
3. At the command prompt, type the following command:  
**disco.exe/out:c:\wsdl http://localhost/woodgrove/bank.asmx**
4. At the command prompt, type the following commands:  
**c:**  
**cd \wsdl**  
**dir**
5. Open the .disco and .wsdl documents in notepad.exe.
6. Show the students the contents of the .disco and .wsdl documents.
7. Close the instances of notepad.exe.
8. Close the console window.

## Module Strategy

Use the following strategy to present this module:

- WSDL Documents

You should approach the topics in this section as a progressive development of an example of a WSDL document. Explain the concepts in the first topic with a simple XML Web service that has only one operation that returns a class, (the code is defined in the student notes). Progressively build upon this example WSDL document when you explain each of the WSDL topics. The intent of teaching WSDL syntax is not for students to write a WSDL document without assistance. The intent is for students to be able to describe the structure of a WSDL document and explain how the definitions in a WSDL document correspond to the code that they will implement in an XML Web service.

- XML Web Service Discovery

Explain that this section focuses on how to find WSDL documents and the endpoints for XML Web services that implement the interfaces that are defined in the WSDL documents. For completeness, it is important that you briefly describe Universal Description, Discovery, and Integration (UDDI) as part of the solution, but defer any in-depth discussion of UDDI until Module 6, “Publishing and Deploying XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*. Explain how students can use Disco.exe to generate local copies of WSDL documents and other discovery documents.

- XML Web Service Proxies

This section is intended to help students understand the benefits of implementing proxies to XML Web services. Remind the students about the lab exercises in Module 3, “The Underlying Technologies of XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*, to emphasize how manual construction of Hypertext Transfer Protocol (HTTP) and Simple Object Access Protocol (SOAP) messages is tedious and error-prone. Point out that Module 3 did not cover the steps that are involved in decoding response messages. Explain that decoding response messages is essential functionality in an XML Web service consumer. Explain that this functionality can be encapsulated in an XML Web service proxy. Discuss the code that is generated for the proxies from a high level. Discuss the properties of a proxy that you can configure. Do not spend too much time on this discussion, because students will be learning about how to implement XML Web service consumers in the next section.

- Implementing an XML Web Service Consumer Using Visual Studio .NET

This section is intended to familiarize students with the steps that are required to allow a managed application to consume an XML Web service. Emphasize that the steps that this section outlines do not apply to unmanaged applications. Emphasize the similarity between the steps involved in implementing an XML Web service consumer that is a console application and an XML Web service consumer that is a Web Forms application. Also, discuss the importance of asynchronous invocation of XML Web service methods. Point out the similarity between invoking XML Web service methods asynchronously and issuing asynchronous HTTP requests by using the **WebRequest** class, which is demonstrated in Module 3, “The Underlying Technologies of XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

# Overview

- WSDL Documents
- XML Web Service Discovery
- XML Web Service Proxies
- Implementing an XML Web Service Consumer Using Visual Studio .NET

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

Developers must understand how an XML (Extensible Markup Language) Web service will be consumed before implementing it. Also, XML Web service developers need to implement XML Web service consumers to test the XML Web services that they implement.

In this module, you will learn how to implement XML Web service consumers by using Microsoft® Visual Studio® .NET.

## Objectives

After completing this module, you will be able to:

- Explain the structure of a Web Service Description Language (WSDL) document.
- Explain the XML Web services discovery process.
- Locate service contracts by using Disco.exe.
- Generate XML Web service proxies by using Wsdll.exe.
- Implement an XML Web service consumer by using Visual Studio .NET.
- Invoke an XML Web service synchronously and asynchronously by using an XML Web service proxy.

# WSDL Documents

- What is WSDL?
- Structure of a WSDL document
  - The **types** element
  - The **message** element
  - The **portType** element
  - The **binding** element
  - The **service** element

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

To use or consume an XML Web service, you first need to know how to interact with it.

## What is WSDL?

Web Services Description Language (WSDL) is an XML grammar that is used for describing an XML Web service in terms of the messages it accepts and generates. In other words, a WSDL file acts as a contract between an XML Web service consumer (client) and an XML Web service.

In a WSDL document, you provide abstract definitions of the types that are used in the operations and the documents that are exchanged for each operation. Then, you associate these definitions with a network protocol and group them into messages to define an endpoint.

WSDL can describe endpoints and their operations without specifying the message formats or the network protocols to which an endpoint is bound. The only protocol bindings that are examined in this section are Simple Object Access Protocol (SOAP) 1.1 and HTTP-GET/POST.

As an XML Web service consumer, it is important that you are familiar with WSDL to understand the contract that is defined in a WSDL document. Also, when you implement an XML Web service, you might not want to use an automatically generated WSDL file. Instead, you might choose to generate the default WSDL file and then modify it. Again, for this purpose you need to know WSDL.

## Structure of a WSDL document

A WSDL document is just a list of definitions. In a WSDL file, the root element is named **definitions**. This element contains five primary child elements that are used to define the XML Web service. The following five elements appear within the **definitions** element in a WSDL file in the order specified:

- The **types** element defines the various data types that are used to exchange messages.
- The **message** element describes the messages to be communicated.
- The **portType** element identifies a set of operations and the messages that are involved with each of those operations.
- The **binding** element specifies the protocol details for various service operations and describes how to translate the abstract content of these messages into a concrete format.
- The **service** element groups a set of related ports together.

The following table describes an XML Web service for which we want to create a WSDL file.

C#	Microsoft Visual Basic .NET
<pre>[XmlRoot("account")] public class Acct {     [XmlElement("description")]     public string Description;     [XmlElement("number")]     public string Number;     [XmlElement("type")]     public string Type;     [XmlElement("balance")]     public decimal Balance;     [XmlAttribute("status")]     public string Status; }  public class TheBank {     [WebMethod]     public Acct     GetAccount(string acctNumber)     {         Acct a = new Acct();         a.Description = "Adam's savings acct";         a.Balance=10000.0M;         a.Number="1234-XX";         a.Status="active";         a.Type="SV";         return a;     } }</pre>	<pre>&lt;XmlRoot("account")&gt; _ Public Class Acct     &lt;XmlElement("description")&gt;     Public Description As String     &lt;XmlElement("number")&gt;     Public Number As String     &lt;XmlElement("type")&gt;     Public Type As String     &lt;XmlElement("balance")&gt;     Public Balance As Decimal     &lt;XmlAttribute("status")&gt;     Public Status As String End Class 'Acct  Public Class TheBank     Public&lt;WebMethod()&gt; _     Function     GetAccount(acctNumber As String) As Acct         Dim a As New Acct()         a.Description = "Adam's savings acct"         a.Balance = 10000D         a.Number = "1234-XX"         a.Status = "active"         a.Type = "SV"         Return a     End Function 'GetAccount End Class 'TheBank</pre>

**The types element**

In this section, you will learn how to define a WSDL document that describes the preceding XML Web service.

First, you define the types that are used in the message exchange. This is mostly just a matter of defining the types by using XML Schema Definition Language (XSD). The **acctNumber** parameter is defined as follows:

```
...
<types>
...
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1"
        name="acctNumber" nillable="true"
        type="s:string" />
    </s:sequence>
  </s:complexType>
...
</types>
```

The type definition for the **Acct** class, which the **GetAccount** method returns, is slightly more complex than the previous definition. The type definition for this class can be as follows:

```
<s:complexType name="Acct">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1"
      name="description" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1"
      name="number" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1"
      name="type" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1"
      name="balance" type="s:decimal" />
  </s:sequence>
  <s:attribute name="status" type="s:string" />
</s:complexType>
```

The preceding type definition represents an XML document with the following structure:

```
<?xml version="1.0" encoding="utf-8"?>
<account status="active">
  <description>Adam's savings acct</description>
  <number>1234-XX</number>
  <type>SV</type>
  <balance>10000</balance>
</account>
```

Next, we define the structure of the messages that are to be exchanged. In this example, the method name is **GetAccount** and we use the following naming convention:

- The inbound message has the same name as the method.
- The outbound message has the name of the method with the word **Response** appended.



One way to create the message type definitions is as follows:

```
<s:element name="GetAccount">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1"
        name="acctNumber" nillable="true"
        type="s:string" />
      <s:element minOccurs="1" maxOccurs="1"
        name="suffix" nillable="true" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetAccountResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1"
        name="account" type="s0:Acct" />
    </s:sequence>
  </s:complexType>
</s:element>
```

All of these definitions are nested inside the **types** element.

### The message element

In addition to defining the data types that are passed back and forth during an invocation for an XML Web service method, you must also define the XML Web service request and response messages. Because messages are protocol independent, you can use a message with HTTP-GET/POST, SOAP, or any other protocol that an XML Web service provider supports. If you use SOAP, the message element corresponds to the payload of the SOAP request or response. It does not include the SOAP **Envelope** or the SOAP **Fault** elements. You can give messages any convenient name because WSDL does not define a naming convention for messages.

The **message** element contains zero or more **part** child elements. A **part** element is similar to a parameter or a return value in a function call.

A request message contains all **in** and **inout** parameters; and a response message contains all **out** and **inout** parameters, and the return value. Each **part** element must have a name and data type that you can match to the data types that are used in the underlying service implementation. To continue with the previous example, you can define the request and response messages as follows:

```
<message name="GetAccountIn">
  <part name="parameters" element="s0:GetAccount" />
</message>
<message name="GetAccountOut">
  <part name="parameters" element="s0:GetAccountResponse" />
</message>
```

In the preceding code, the **s0:GetAccount** and **s0:GetAccountResponse** attribute values refer to the top-level types that the **types** element defines.

**The portType element**

An XML Web service provider (a network node, which is a Web server) may expose multiple XML Web services. A single XML Web service can support invocation of its operations by using a variety of protocols. The format of the data that is exchanged between an XML Web service consumer and an XML Web service may depend on the protocol that is used to invoke an operation or a method. Therefore, there must be a way to match the operations to the endpoints from which they can be accessed. You can do this kind of matching by using the **portType** element.

The following XML code shows the **GetAccount** operation and a **portType** with which it is associated:

```
<portType name="BankService">
  <operation name="GetAccount">
    <input message="s0:GetAccountIn" />
    <output message="s0:GetAccountOut" />
  </operation>
</portType>
```

In the preceding code, notice that the **input** and **output** elements specify the names of the request and response messages that are to be transmitted.

**The binding element**

After defining the logical port, next you define how an XML Web service consumer can bind to the port on which the **GetAccount** operation is available. This involves associating an operation with a protocol and providing any protocol-specific binding information. To do this, you use the **binding** element. The following XML code shows the SOAP binding definition for the **GetAccount** operation:

```
<binding name="BankService" type="s0:BankService">
  <soap:binding transport="http" style="document" />
  <operation name="GetAccount">
    <soap:operation soapAction="http://woodgrovebank.com/GetAccount" style="document" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
```

The preceding example allows binding by using SOAP. It provides information for the **SOAPAction** header. It also specifies that the document-encoding style is **document** and not **rpc**, and that the parameter-encoding style is **literal** and not **encoded**. You can give the binding any convenient name.

To see some examples of bindings that use the HTTP-GET and HTTP-POST protocols and examine the **binding** elements, see the file at: <http://localhost/Woodgrove/Bank.asmx?wsdl>.

### The service element

All that remains in creating a WSDL file is defining the endpoints for each of the protocols that you can use to access an XML Web service. To define the endpoints, you use the **service** element.

The following XML code defines a bank service and specifies the ports that can be used to access the operations of the service:

```
<service name="BankService">
  <port name="BankService" binding="s0:BankService">
    <soap:address location =
      "http://localhost/woodgrove/Bank.asmx" />
  </port>
</service>
```

In the preceding code, the **binding** attribute of the **port** element specifies the name of the **binding** element that is to be used, in this example, **s0:BankService**. Also, notice that the endpoint location is specified as a child element of the **port** element.

---

**Note** The complete code for the preceding WSDL file is available in `<install folder>\Democode\<language>\Mod04\WoodgroveBank.wsdl`.

---

If you can access an XML Web service by using multiple protocols, then a WSDL document for the XML Web service will contain multiple **port** elements, each referring to a protocol-specific **binding** element by name. Also, each of the protocol-specific **binding** elements will in turn refer to protocol-specific **portType** elements. The protocol-specific **portType** elements will in turn refer to protocol-specific sets of input and output **message** elements, which in turn refer to types that the **types** element defines.

## XML Web Service Discovery

- Introducing Disco
- Locating Discovery Documents Using Disco.Exe
- Demonstration: Locating Discovery Documents Using Disco.exe
- Programmatic Discovery

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In the previous topic, you saw how to write WSDL documents. Because a WSDL document specifies the format of the messages that an XML Web service exchanges with its consumers, you must implement the consumer according to the WSDL document of an XML Web service that you want to consume. If you do not already have the WSDL document, you must be able to locate it.

The process by which you locate an XML Web service and its descriptions and learn how to interact with it is known as XML Web service discovery.

In this section, you will learn how to locate the WSDL documents at a known endpoint. For more information about how to locate an XML Web service whose endpoints are unknown, see Module 6, “*Publishing and Deploying XML Web Services*,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

## Introducing Disco

- What is Disco?
- Static discovery
- Dynamic discovery
- WS-Inspection

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

It is unlikely that an XML Web service provider would publish the service descriptions of all the XML Web services through an XML Web service broker. Therefore, the developers of the XML Web service consumer must be able to discover the service descriptions.

### What is Disco?

Disco is a mechanism for enumerating the XML Web services that are available at a particular endpoint and locating the service contracts for the enumerated XML Web services. Disco is a proprietary Microsoft solution to the static discovery problem. In the future, an industry standard solution will be supported.

---

**Note** A Web server that provides access to an XML Web service is not required to support discovery. Either another server can be responsible for providing the service contracts, or an XML Web service has been created for only private use with no public mechanism for discovery.

---

An XML Web service provider can make discovery information available to developers of XML Web service consumers. It can do this by either statically or dynamically generating a document that contains a link to the WSDL document for all of the XML Web services that the provider hosts.

**Static discovery**

Static discovery is possible when the location of a discovery document (usually with the extension .disco) is already known. Static discovery involves retrieving the discovery document and interpreting its contents. The following code is an example of a discovery document:

```
<?xml version="1.0"?>
<discovery xmlns:xsi="http://www.w3.org/2000/10/↵
        XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2000/10/XMLSchema/"
        xmlns="http://schemas.xmlsoap.org/disco/">
  <discoveryRef ref="http://www.contoso.com/MicroPayment↵
        /MicroPayment.vsdisco"/>
  <discoveryRef ref="http://www.contoso.com/Services/↵
        AccountMgmt.disco"/>
  <contractRef ref="http://www.contoso.com/Services/↵
        AcctMgmt.asmx?wsdl"
        docRef="http://www.contoso.com/Services/↵
        AcctMgmt.asmx"↵
        xmlns="http://schemas.xmlsoap.org/disco/scl"/>
</discovery>
```

The two main elements of a discovery document are:

- **discoveryRef**, which specifies the location of additional discovery documents.
- **contractRef**, which specifies the location of XML Web service contracts.

The **contractRef** element can also optionally specify the location of a document that provides the documentation for an XML Web service.

In a discovery document, the URLs that specify the various document locations can be absolute or relative. If the URLs are relative, then the locations are assumed to be relative to the location of the discovery document.

The root discovery document along with all the referenced discovery documents make up the catalog of XML Web services that are available at the interrogated endpoint. This catalog is effectively static.

**Dynamic discovery**

Dynamic discovery takes place when all that is known to the consumer is the endpoint of the XML Web service provider. In this situation, there is no static list of .disco files at the endpoint. Instead, the list of available XML Web services and the associated service contracts must be dynamically generated.

The dynamic discovery of XML Web services is disabled by default. You can enable it by removing the comment for .vsdisco httpHandler in either machine.config or web.config.

```
<httpHandlers>
  <!--<add verb="*" path="*.vsdisco"
        type=
          ↵"System.Web.Services.Discovery.DiscoveryRequestHandler,
          ↵System.Web.Services,
          ↵Version=1.0.3300.0,Culture=neutral,
          ↵PublicKeyToken=b03f5f7f11d50a3a" validate="false"
        />-->
```

You will also need to place a file named Default.vsdisco at the root of the Web site that hosts an XML Web service. If you install Visual Studio .NET on a computer, Default.vsdisco will be placed in the folder containing the root of the computer's default Web site. The following code is an example of a .vsdisco file:

```
<?xml version="1.0" ?>
<dynamicDiscovery xmlns="urn:schemas-
dynamicdiscovery:disco.2000-03-17">
  <exclude path="_vti_cnf" />
  <exclude path="_vti_pvt" />
  <exclude path="_vti_log" />
  <exclude path="_vti_script" />
  <exclude path="_vti_txt" />
</dynamicDiscovery>
```

The preceding .vsdisco document contains a list of subfolders to be excluded in the search for discovery documents.

For dynamic discovery, the .vsdisco extension is mapped to Aspnet\_isapi.dll. You use the **System.Web.Services.Discovery.DiscoveryRequestHandler** class to request a .vsdisco file. The handler searches the folder that contains the requested .vsdisco file and its subfolders for XML Web services that are created by using ASP.NET (.asmx), dynamic discovery files (.vsdisco), and static discovery files (.disco). The search in a folder terminates if an .asmx or .vsdisco file cannot be found, or if a .disco file is found. It is invalid to place a .vsdisco file in the same folder as a .disco file.

---

**Caution** It is not advisable to allow unknown clients the unrestricted ability to discover your XML Web services. Therefore, it is recommended that you use only dynamic discovery on development Web servers. For production deployment, it is recommended that you create a static discovery file (.disco) for those XML Web services that you want to enable clients to discover.

---

## WS-Inspection

WS-Inspection is a proposed standard for discovery. WS-Inspection is a collaborative effort by Microsoft and IBM. The WS-Inspection specification defines an XML format to allow a site to be inspected for available services. The specification also defines a collection of rules for how inspection-related information should be made available for consumption. The following code is an example of a simple WS-Inspection document:

```
<?xml version="1.0"?>
<inspection xmlns=
  ↪"http://schemas.xmlsoap.org/ws/2001/10/inspection/">
  <service>
    <description referencedNamespace=
      ↪"http://schemas.xmlsoap.org/wsdl/"
      ↪location="http://example.com/stockquote.wsdl" />
    </service>
  </inspection>
```

It is not difficult to see the similarity between .disco files and WS-Inspection documents.

The WS-Inspection specification may be found at:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsrvspec/html/ws-inspection.asp>

Implementations, tools, and samples are available as part of Visual Studio .NET and may be found at:

<http://msdn.microsoft.com/code/default.asp?url=/code/sample.asp?url=/msdn-files/026/002/541/msdncompositedoc.xml>



## Locating Discovery Documents Using Disco.exe

### ■ Syntax

```
disco [options] URL
```

### ■ Uses for Disco.exe

#### ● Example

```
disco /out:d:\disco /u:administrator /p:biffle ↵
http://www.woodgrovebank.com/catalog.disco
```

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

You can use the Microsoft XML Web services discovery tool, Disco.exe, to discover the endpoints of XML Web services and save the documents that are related to each of the XML Web services on a local disk.

### Syntax

The syntax of the command to search for discovery documents (files with the following extensions: .disco, .wsdl, .xsd, .discomap, .vsdisco) at a given URL is as follows:

```
disco [options] URL
```

The options for the URL argument that the **disco** command supports are described in the following table.

Options	Description
<b>/d[omain]:domain</b>	Specifies the domain name to use when connecting to a server that requires a domain name for authentication.
<b>/out:directoryName</b>	Specifies the output directory in which to save the discovered documents. The default is the current directory.
<b>/u[sername]:username</b>	Specifies the username to use when connecting to a server that requires authentication.
<b>/p[assword]:password</b>	Specifies the password to use when connecting to a server that requires authentication.

**Uses for Disco.exe**

The primary use of Disco.exe is to generate local copies of service contract documents (.wsdl documents) and static discovery documents (.disco documents). The tool also produces a file named Results.discomap. This file can be used as the input to the Wsdl.exe, which you will examine later in this module.

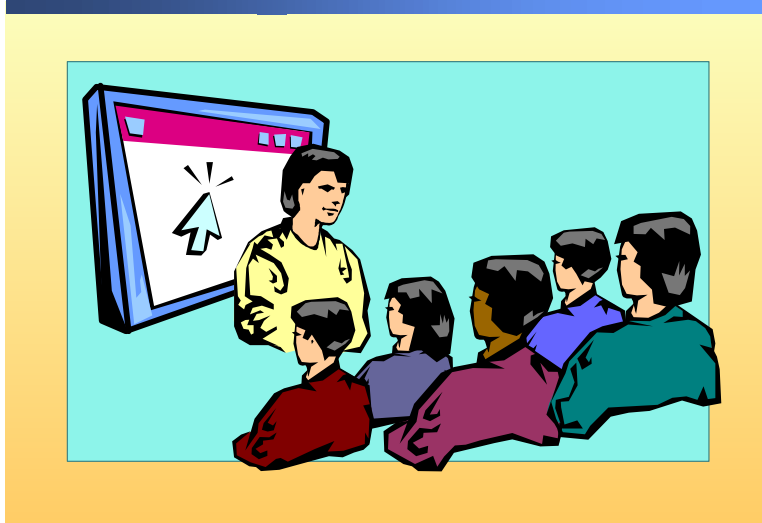
The Disco tool displays an error message if it cannot find discoverable resources at the supplied URL.

The following example shows how to use Disco.exe to search a URL for discovery documents and save them to a local folder:

```
disco /out:d:\disco /u:administrator /p:biffle→  
http://localhost/woodgrove/catalog.disco
```

In the preceding example, the username and password are supplied to allow the tool to connect to a server that requires authentication.

## Demonstration: Locating Discovery Documents Using Disco.exe



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this demonstration, you will learn how to locate discovery documents for an XML Web service by using Disco.exe.

## Programmatic Discovery

- The **System.Web.Services.Discovery** namespace
- An example

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

All of the discovery functionality of Disco.exe is available programmatically through the classes in the Microsoft .NET Framework.

### The **System.Web.Services.Discovery** namespace

The **System.Web.Services.Discovery** namespace in the .NET Framework contains a set of classes that model the elements of a discovery document. This namespace also contains classes that you can use to manipulate discovery documents.

### C# and Visual Basic .NET Examples

The following C# and Microsoft Visual Basic® .NET examples simulate the behavior of Disco.exe.

C#

---

```
using System.Web.Services.Discovery;
class DiscoverDemo {
    static void Main(string[] args)
    {

        GenerateDiscovery(@"http://localhost/woodgrove/catalog.disc
        ↪o",
            @"d:\disco",true);
    }
    public static void GenerateDiscovery(string sourceUri,↪
        string outputDirectory, bool printToConsole) {
        string outputPath;
        outputPath =
        ↪Path.Combine(outputDirectory,"results.discomap");
        DiscoveryClientProtocol client;
        client = new DiscoveryClientProtocol();
        FileInfo fi = new FileInfo(outputPath);
        if (fi.Exists)
            client.ReadAll(outputPath);
        DiscoveryDocument doc = client.DiscoverAny(sourceUri);
        client.ResolveAll();
        foreach (DictionaryEntry entry in client.Documents)
            Console.WriteLine((string) entry.Key);

        DiscoveryClientResultCollection results;
        results =
        ↪client.WriteAll(outputDirectory,"results.discomap");
        foreach (DiscoveryClientResult res in results)
        {
            Console.WriteLine(res.Filename + " <- " + res.Url);
        }
    }
}
```

**Visual Basic .NET**

---

```
Imports System.Web.Services.Discovery
Class DiscoverDemo
    Shared Sub Main(args() As String)
        GenerateDiscovery("http://localhost/catalog.disco", ↵
            "d:\disco", True)
    End Sub 'Main

    Public Shared Sub GenerateDiscovery(sourceUri As String,
        outputDirectory As String, printToConsole As Boolean)
        Dim outputPath As String
        outputPath = Path.Combine(outputDirectory,
            ↵"results.discomap")
        Dim client As DiscoveryClientProtocol
        client = New DiscoveryClientProtocol()
        Dim fi As New FileInfo(outputPath)
        If fi.Exists Then
            client.ReadAll(outputPath)
        End If
        Dim doc As DiscoveryDocument =
            ↵client.DiscoverAny(sourceUri)
        client.ResolveAll()
        Dim entry As DictionaryEntry
        For Each entry In client.Documents
            Console.WriteLine(CStr(entry.Key))
        Next
        Dim results As DiscoveryClientResultCollection
        results = client.WriteAll(outputDirectory,
            ↵"results.discomap")
        Dim res As DiscoveryClientResult
        For Each res In results
            Console.WriteLine((res.Filename + " <- " + res.Url))
        Next
    End Sub 'GenerateDiscovery
End Class 'DiscoverDemo
```

---

**Note** An in-depth discussion of the **System.Web.Services.Discovery** namespace is beyond the scope of this course. For complete information about using the **System.Web.Services.Discovery** namespace, refer to the .NET Framework SDK documentation.

---

## XML Web Service Proxies

- Proxies and WSDL
- Generating Proxies Using Wsdl.exe
- Configuring Proxies

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

An XML Web service consumer must be able to construct the messages that are to be sent to an XML Web service, and parse the messages that are received from an XML Web service. Manually writing the code to construct and parse the messages is time-consuming and error-prone. It is better to encapsulate this code in a class that you can reuse. We call such a class a *proxy class*. In this section, you will see how to generate proxy classes by using Wsdl.exe and WSDL documents.

## Proxies and WSDL

- Why are proxies needed?
- Using WSDL to generate proxies

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

*Proxies* are entities that act as intermediaries for other entities. For the purpose of our discussion, you can consider proxies as objects that expose the same logical interface to an XML Web service consumer as an XML Web service.

### Why are Proxies needed?

Module 3, “The Underlying Technologies of XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET* explains how to interact with an XML Web service by manually constructing the messages and sending them to an XML Web service. However, manual construction of messages is not an optimal solution.

Also, developers prefer to work with strongly-typed abstractions that model the logical interface of an entity with which they are interacting. They do not want to and should not have to manually process text documents that are exchanged during the invocation of an XML Web service method.

Proxies eliminate the preceding problems by providing a strongly-typed interface that matches the operations that an XML Web service exposes, and by hiding the construction and parsing details within the implementation of the proxy.

### Using WSDL to generate proxies

Because WSDL documents are XML documents, it is easy to generate language-specific proxies to XML Web services. It is also easy to match the types defined in the WSDL document to types that are native to the language of the consumer that uses the proxy.



Earlier in this module, in the topic on WSDL, you learned how to create the XSD representation of the following class definition.

C#	Visual Basic .NET
<pre>public class Acct {     public string Description;     public string Number;     public string Type;     public decimal Balance;     public string Status; }</pre>	<pre>Public Class Acct     Public Description As String     Public Number As String     Public Type As String     Public Balance As Decimal     Public Status As String End Class 'Acct</pre>

You can easily perform the reverse transformation of generating a class definition from a WSDL document, and you can automate the process. In the next topic, you will learn how to generate proxies by using Wsdl.exe and by using Visual Studio .NET.

---

**Note** For information about how to programmatically manipulate WSDL documents, you can use the classes in the **System.Web.Services.Description** namespace in the .NET Framework.

---

## Generating Proxies Using Wsd1.exe

### ■ Syntax for invoking Wsd1.exe

```
wsdl [options] {URL | Path}
```

### ■ Examples

```
wsdl http://www.woodgrovebank.com/services/bank.asmx?wsdl
```

```
wsdl /l:VB /protocol:HttpGet /out:Bank.cs ↵  
http://www.woodgrovebank.com/services/bank.asmx?wsdl
```

### ■ Proxy class details

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Microsoft provides Wsd1.exe, which is a tool that you can use to generate the code for a proxy to an XML Web service. Wsd1.exe uses documents such as WSDL files, XSD schemas, and .discomap to generate the proxy.

### Syntax

The syntax for the **wsdl** command that invokes Wsd1.exe is as follows:

```
wsdl [options] {URL | Path}
```

In the syntax, the URL is to a WSDL file (.wsdl), XSD schema file (.xsd), or a discovery document (.disco). Note that you cannot specify a URL to a .discomap discovery document.

The arguments and options for the path that the **wsdl** command supports are described in the following table.

Options	Description
	The path to a local .wsdl, .xsd, .disco, or .discomap.
/d[omain]: <i>domain</i>	Specifies the domain name to use when connecting to a server that requires a domain name for authentication.
/extendednaming	Allows the use of extended naming when generating dataset classes.
/l[anguage]: <i>language</i>	Specifies the language to use for the generated proxy class. You can specify any of the following as the language argument: <b>CS</b> for C#, which is the default option <b>VB</b> for Microsoft Visual Basic .NET <b>JS</b> for Microsoft JScript® You can also specify the fully qualified name of a class that implements the <b>System.CodeDom.Compiler.CodeDomProvider</b> class.
/n[amespace]: <i>namespace</i>	Specifies the namespace for the generated proxy. The default namespace is the global namespace.
/out: <i>filename</i>	Specifies the file in which to save the generated proxy code. The tool derives the default file name from the XML Web service name. The tool saves generated datasets in different files.
/u[sername]: <i>username</i>	Specifies the username to use when connecting to a server that requires authentication.
/p[assword]: <i>password</i>	Specifies the password to use when connecting to a server that requires authentication.
/protocol: <i>protocol</i>	Specifies the protocol to implement. This option overrides the default protocol. You can specify <b>SOAP</b> (default option), <b>HttpGet</b> , <b>HttpPost</b> , or a custom protocol specified in the configuration file (Web.config).
/server	Generates an abstract class for an XML Web service based on the contracts. The default is to generate client proxy classes.

**C# and Visual Basic  
.NET Examples**

The following example generates a proxy by using the default language (C#) and default protocol (SOAP).

```
wsdl http://localhost/woodgrove/bank.asmx?wsdl
```

A common use of Wsdl.exe is to generate a proxy for a specific protocol and non-default language. The following example generates a proxy that uses the HTTP-GET protocol and is implemented by using Visual Basic .NET:

```
wsdl /l:VB /protocol:HttpGet /out:Bank.vb ↵  
http://localhost/woodgrove/bank.asmx?wsdl
```

**Proxy Class Details**

A proxy class that Wsdl.exe generates exposes both synchronous and asynchronous methods for each of the operations in an XML Web service. For example, if an XML Web service supports an operation named **GetAccount**, then a proxy class for this XML Web service will contain the methods **GetAccount**, **BeginGetAccount**, and **EndGetAccount**. The **GetAccount** method is used to invoke the XML Web service synchronously and the **BeginGetAccount/EndGetAccount** pair is used to invoke the XML Web service asynchronously. You will see how to make both synchronous and asynchronous calls to an XML Web service later in this module.

Each of the methods in a proxy class correctly formats the messages and parses the responses. These tasks include translating data between XML and the .NET runtime supported types.

The methods in the proxy class also contain details on the network communication to be used. By default, the proxy classes use SOAP to invoke an XML Web service method. The proxy classes use SOAP because SOAP supports the richest set of data types in comparison with the other two supported protocols. However, if an XML Web service can only be called by using HTTP-GET or HTTP-POST protocols, then Wsdl.exe can also generate proxy classes that support these protocols.

If errors occur during a call to an XML Web service, then the proxy class throws standard .NET exceptions.

The proxy class is derived from a protocol-specific class, which is derived from the **HttpWebClientProtocol** class. For more information about proxy classes, refer to the .NET Framework SDK documentation.

## Configuring Proxies

- The **Url** property
- The **Credentials** property
- The **Timeout** property
- The **Proxy** property
- The **AllowAutoRedirect** property

```
Dim theBank As New Bank()
theBank.Url =
    "http://eastcoast.woodgrovebank.com/Bank.asmx"
Dim credentials = New
    NetworkCredential("Adam", "woodgrovebank.com", "ABarr-
    user")
theBank.Credentials = credentials
theBank.Timeout = 20000
Dim proxyObject As IWebProxy = New
    WebProxy("http://proxyserver:80", True)
theBank.Proxy = proxyObject
theBank.AllowAutoRedirect = True
```

C#

Visual Basic

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

After generating a proxy class by using Wsdl.exe, you can modify the default state of the proxy in a number of ways.

### The Url Property

Proxy classes that are generated by using Wsdl.exe set a default **Url** property for the use of clients or consumers. The default value of the **Url** property is determined by the **location** attribute that is specified in a service description that the proxy class was generated from.

You can change the **Url** property to refer to any XML Web service that implements the same service description that the proxy class was generated from. You might want to change the **Url** property to support load-balancing schemes or fail-over scenarios.

The following example code shows how you can set the **Url** property.

C#	Visual Basic .NET
Bank theBank = new Bank();	Dim theBank As Bank = new
theBank.Url =	Bank();
↳ "http://eastcoast.woodgroveb	theBank.Url =
↳ ank.com/Bank.asmx";	↳ "http://eastcoast.woodgroveb
	↳ ank.com/Bank.asmx"

### The Credentials property

If an XML Web service consumer must be authenticated by a credentials-based authentication mechanism, such as Basic, Digest, NTLM, or Kerberos, then you can supply credentials by using the **Credentials** property of a proxy class.

The following example code shows how to supply credential details by using the **Credentials** property.

C#	Visual Basic .NET
<pre>ICredentials credentials = new NetworkCredential("Adam", ↵     "woodgrovebank.com", "ABarr-     ↵user"); theBank.Credentials =     ↵credentials;</pre>	<pre>Dim credentials = New NetworkCredential("Adam", ↵     "woodgrovebank.com",     ↵"ABarr-user") theBank.Credentials =     ↵credentials</pre>

### The Timeout property

You use the **Timeout** property to modify the default timeout for synchronous invocations of the XML Web service method. When you modify the **Timeout** property, it applies to all subsequent requests made with the same instance of the proxy class, until it is modified again.

**Note** Even though an XML Web service consumer can set the **Timeout** property to infinity, a Web server can still force the request to time out on the server side.

The following example sets the timeout on the client side to 20 seconds.

C#	Visual Basic .NET
<pre>theBank.Timeout = 20000;</pre>	<pre>theBank.Timeout = 20000</pre>

### The Proxy property

If a client needs to use proxy settings that are different from the default system settings, then you can use the **Proxy** property. In the following example, you use the **WebProxy** class to set the proxy settings.

C#	Visual Basic .NET
<pre>// Set the proxy server to     ↵proxyserver, //port to 80, and specify that     proxy server //must be bypassed for local     ↵addresses IWebProxy proxyObject = new↵ WebProxy("http://proxyserver:8     ↵0", true); theBank.Proxy = proxyObject;</pre>	<pre>'Set the proxy server to     proxyserver, ↵ 'port to 80, and specify     that↵ proxy server 'must be bypassed for local↵     addresses IWebProxy proxyObject = new↵ WebProxy("http://proxyserver:8     ↵0", true) theBank.Proxy = proxyObject</pre>

### The AllowAutoRedirect property

If a client is sending authentication information, such as a user name and password, typically you would not want to allow the server to redirect the request to another server for security reasons. By default, the **AllowAutoRedirect** property is set to false, but you can override this value as shown in the following example.

C#	Visual Basic .NET
<pre>// Allow the server to     ↵automatically // redirect the request theBank.AllowAutoRedirect =     ↵true;</pre>	<pre>' Allow the server to     ↵automatically ' redirect the request theBank.AllowAutoRedirect =     ↵true</pre>

# Implementing an XML Web Service Consumer Using Visual Studio .NET

- Demonstration: Implementing a Console Client
- Demonstration: Implementing a Web Forms Client
- Synchronous vs. Asynchronous Clients

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Implementing a basic XML Web service consumer is a simple process. An XML Web service consumer can be any of the following types of applications:

- Console
- Windows Forms
- Web Forms
- Web Service

The type of application that you implement has very little effect on the mechanics of consuming an XML Web service.

For all XML Web service consumers, the basic steps involved in consuming an XML Web service are as follows:

1. Create a proxy class for the XML Web service.
2. Reference the proxy class in the application code.
3. Create an instance of the proxy class in the application code.
4. Invoke an XML Web service method by using the instance of the proxy class.

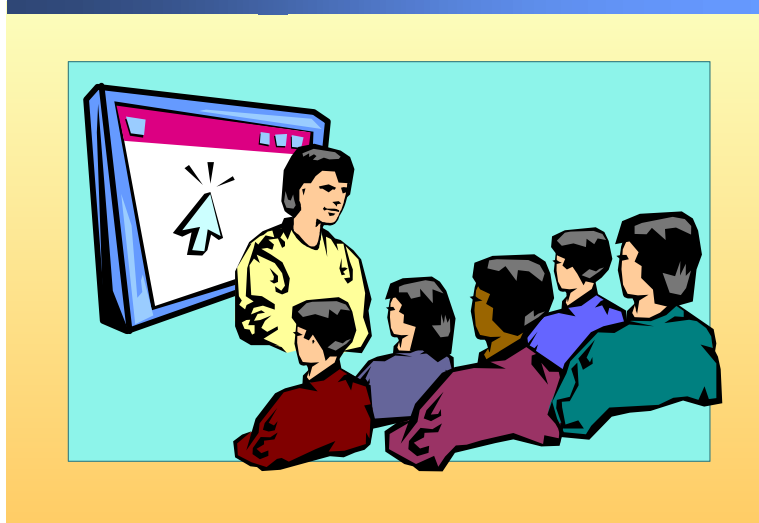
In the next two topics, you will see demonstrations on how to implement a console application client and a Web Forms client.

---

**Note** The XML Web service consumers that you will implement in this course are .NET applications.

---

## Demonstration: Implementing a Console Client



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this demonstration, you will learn how to implement a console application that is a consumer of an XML Web service.

### ► To create a console application

- In Visual Studio .NET, create a console application. You can use Microsoft Visual C#™ or Microsoft Visual Basic .NET. Name the application Mod04Consumer.

### ► To add a Web reference

After creating the base application, you must generate a proxy class and add a reference to this class to your project.

1. In Solution Explorer, right-click the **References** node and click **Add Web Reference**.
2. In the **Add Web Reference** dialog box, enter the address <http://localhost/woodgrove/bank.asmx>.

After performing the previous action, the Service Help page is displayed to you.

3. Add a reference to the XML Web service to your project by clicking **Add Reference**.

When you add a Web reference, Visual Studio .NET automatically generates an XML Web service proxy class.



4. Expand the **Web References** node in Solution Explorer.

You will see a child node with the name **localhost**. This name is used as a nested namespace name. The name is also used as the name of a folder under the project folder. This folder contains all the files (including source files for the XML Web service proxy) that are generated when you added the **Web Reference**.

5. Rename the **localhost** node to **Woodgrove**.
6. Open **Class View** to see the generated proxy class.

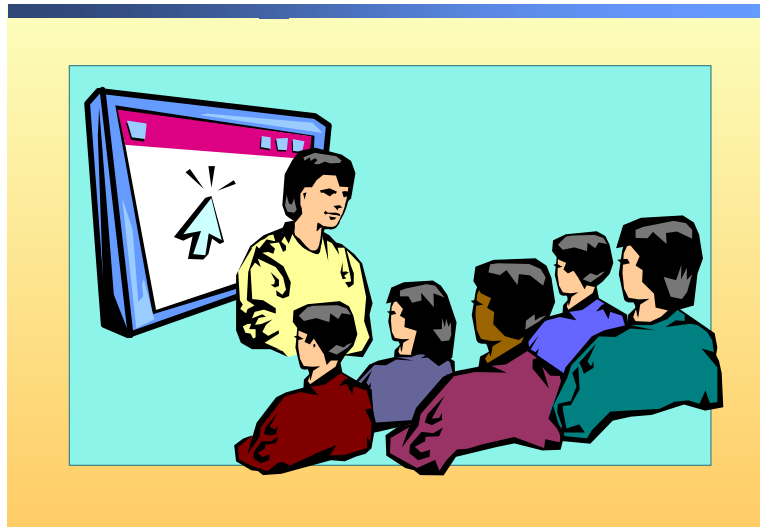
► **To invoke the XML Web service by using the proxy**

- Invoke the **GetAccount** method of the Woodgrove XML Web service.

The following code shows how to invoke the **GetAccount** method. Notice the name of the namespace that is used in the following code:

C#	Visual Basic .NET
<pre>using System; using Mod04Consumer.Woodgrove;  namespace Mod04Consumer {     class TheConsumer {         static void Main(string[] args) {             WoodgroveOnlineBank bank =                 new WoodgroveOnlineBank();             Acct acct;             acct = bank.GetAccount(1);             Console.WriteLine("The account ↳{1:C}",              acct.description,acct.balance);         }     } }</pre>	<pre>Imports Mod04Consumer.Woodgrove  Module TheConsumer     Sub Main()         Dim bank As New ↳             WoodgroveOnlineBank()         Dim acct As Acct         acct = bank.GetAccount(1)         Console.WriteLine("The account ↳'{0}' ↳             has a balance of {1:C}",↳             acct.description, acct.balance)     End Sub 'Main End Module</pre>

## Demonstration: Implementing a Web Forms Client



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this demonstration, you will learn how to implement a Web Form application that is an XML Web service consumer.

At the end of this demonstration, you will see that neither the code that is used to invoke an XML Web service, nor the steps to add an XML Web service proxy class to the project, depend on the type of client.

### ► To create a console application:

- In Visual Studio .NET, create a ASP.NET Web Application application. You can use Microsoft Visual C#™ or Microsoft Visual Basic .NET. Specify the location as <http://localhost/Mod04Web>.

### ► To add a Web reference

After creating the base application, you must generate a proxy class and add a reference to this class to your project.

1. In Solution Explorer, right-click the **References** node and click **Add Web Reference**.
2. In the **Add Web Reference** dialog box, enter the address <http://localhost/woodgrove/bank.asmx>.  
After performing the previous action, the Service Help page is displayed to you.
3. Add a reference to the XML Web service to your project by clicking **Add Reference**.

When you add a Web reference, Visual Studio .NET automatically generates an XML Web service proxy class.

4. Expand the **Web References** node in Solution Explorer.

You will see a child node with the name **localhost**. This name is used as a nested namespace name. The name is also used as the name of a folder under the project folder. This folder contains all the files (including source files for the XML Web service proxy) that are generated when you added the **Web Reference**.

5. Rename the **localhost** node to **Woodgrove**.
6. Open **Class View** to see the generated proxy class.

► **To invoke the XML Web service by using the proxy**

- In the Page\_Load event, invoke the **GetAccount** method of the Woodgrove XML Web service and display the account balance.

The following code shows how to invoke the **GetAccount** method. Notice the name of the namespace that is used in the following code:

C#	Visual Basic .NET
using System;	Imports Mod04Web.Woodgrove
using Mod04Web.Woodgrove;	
namespace Mod04Web	Public Class WebForm1
{	Inherits System.Web.UI.Page
public class WebForm1 :	...
System.Web.UI.Page	Private Sub Page_Load(ByVal sender
{	As System.Object, _
...	ByVal e As System.EventArgs) _
private void Page_Load(object	Handles MyBase.Load
↪sender,	Dim bank As New
System.EventArgs e)	↪WoodgroveOnlineBank()
{	Dim acct As Acct
WoodgroveOnlineBank bank = ↪	acct = bank.GetAccount(1)
new WoodgroveOnlineBank();	Response.Write( _
Acct acct;	String.Format("Balance :
acct = bank.GetAccount(1);	↪{1:C}", _
Response.Write(	acct.description,
string.Format("Balance :	↪acct.balance))
↪{1:C}",	End Sub
	End Class
acct.description,acct.balance));	
}	
}	

## Synchronous vs. Asynchronous Clients

- Limitations of synchronous calls
- Making asynchronous calls using the proxy

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

The easiest approach to invoking methods on an XML Web service is to make synchronous calls. In certain situations, this approach is not appropriate. This topic examines the situations when synchronous calls are not optimal and how to use proxies to make asynchronous calls.

### Limitations of synchronous calls

For most users, the Internet is a low-bandwidth network. Even for people or organizations with high-speed access to the Internet, latency on the Internet is far higher than for local area networks (LANs). As a result, making all of the calls to an XML Web service synchronously is often unacceptable. An application must remain responsive to the end user, and cannot afford to serialize all of its activities. A solution is to make the calls to an XML Web service asynchronous.

### Making asynchronous calls using the proxy

The XML Web service proxies that Visual Studio .NET and Wsdl.exe generate support asynchronous invocation of XML Web service methods. The asynchronous behavior is implemented by using delegates.

The following procedure will provide you with an example of how to make asynchronous calls by using proxies.

1. Create an instance of an **AsyncCallback** delegate.
2. Invoke a **Beginxxxx** method of the proxy and pass a reference to the proxy itself.

The proxy reference is passed so that the **Endxxxx** method for the appropriate proxy can be invoked when the callback delegate is invoked. Invoking the proxy at the same time that the callback delegate is invoked is necessary because you may have multiple pending calls and might have a separate delegate object handle the completion of each call.

3. When the callback delegate is invoked, access the proxy reference that is passed to the **Beginxxxx** method through the **IAsyncResult.AsyncState** property.
4. Use the proxy reference to call the **Endxxxx** method to complete the asynchronous call.

### C# and Visual Basic .NET examples

The following code shows how to make an asynchronous call to an XML Web service by using a proxy.

#### C#

---

```
class TheConsumer
{
    static void Callback(IAsyncResult ar)
    {
        OnlineBank bank = (OnlineBank) ar.AsyncState;
        Acct acct = bank.EndGetAccount(ar);
        Console.WriteLine("The account '{0}' has a balance of
↵{1:C}",
            acct.description,acct.balance);
    }
    static void Main(string[] args)
    {
        OnlineBank bank = new OnlineBank();
        AsyncCallback callback;
        callback = new AsyncCallback(TheConsumer.Callback);
        bank.BeginGetAccount("1234",callback,bank);
        Console.ReadLine();
    }
}
```

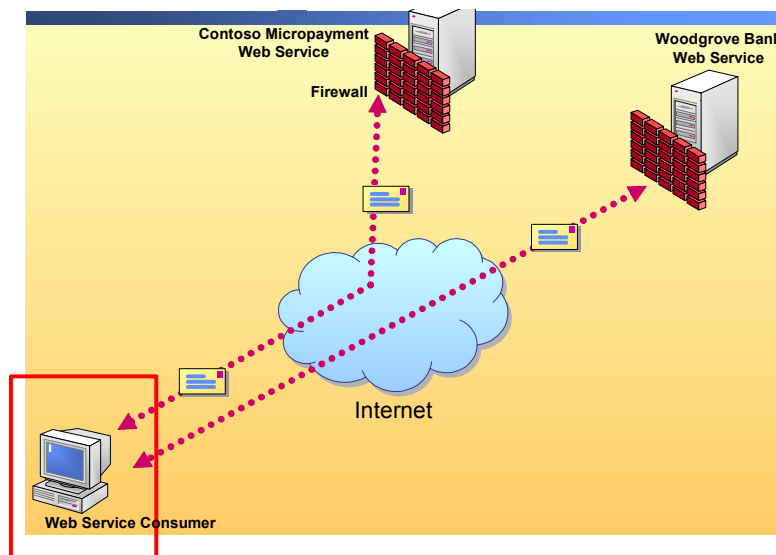
#### Visual Basic .NET

---

```
Class TheConsumer
    Shared Sub Callback(ar As IAsyncResult)
        Dim bank As OnlineBank = CType(ar.AsyncState, OnlineBank)
        Dim acct As Acct = bank.EndGetAccount(ar)
        Console.WriteLine("The account '{0}' has a balance of
↵{1:C}")
        acct.description,acct.balance);
    End Sub

    Shared Sub Main(args() As String)
        Dim bank As New OnlineBank()
        Dim callback As AsyncCallback
        callback = New AsyncCallback(TheConsumer.Callback) '
        bank.BeginGetAccount("1234", callback, bank)
        Console.ReadLine()
    End Sub 'Main
End Class 'TheConsumer
```

## Lab 4.1: Implementing an XML Web Service Consumer Using Visual Studio .NET



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Objectives

After completing this lab, you will be able to:

- Locate XML Web service contracts by using Disco.
- Generate XML Web service proxies by using XML Web service contracts.
- Implement an XML Web service consumer by using Visual Studio .NET.
- Invoke an XML Web service synchronously.

---

**Note** This lab focuses on the concepts in this module and as a result may not comply with Microsoft security recommendations.

---

### Lab Setup

There are starter and solution files that are associated with this lab. The starter files are in the folder <lab root>\Lab04\Starter. The solution files for this lab are in the folder <lab root>\Lab04\Solution.

**Scenario**

In this lab, you are provided with a Microsoft Visual Studio .NET Windows application as a starting point. The application is named Woodgrove and Contoso Account Manager. This application is a client to two XML Web services: the Woodgrove Online Bank XML Web service and the Contoso Micropayment XML Web service. These XML Web services are described as follows:

- **Woodgrove Online Bank**

A bank that provides traditional online services such as account balances. The bank's services are made available through an XML Web service.

- **Contoso Micropayment Service**

A micropayment service (similar to Microsoft Wallet) that manages accounts for users who do not want to provide their personal financial information to all of the e-commerce sites at which they shop. The account management features of this service are also exposed as an XML Web service.

In this lab, you will modify the Woodgrove and Contoso Account Manager to access only the Woodgrove Online Bank service. You will add support for the Contoso Micropayment Service in a subsequent lab.

You will locate the WSDL service contract for the Woodgrove Online Bank XML Web service by using Visual Studio .NET, and then create an XML Web service proxy class. You will then modify the client application to use the following Woodgrove Online Bank XML Web service methods:

- **GetAllAccounts**
- **GetAccount**
- **GetTransactionHistory**

**Estimated time to  
complete this lab: 75  
minutes**

## Exercise 1

# Locating a WSDL Service Contract and Generating an XML Web Service Proxy

In this exercise, you will examine the Woodgrove Online Bank XML Web service functionality. Then, you will locate the WSDL service contract for the Woodgrove Online Bank XML Web service and then generate an XML Web service proxy to use it.

### ► Examine the Woodgrove Online Bank XML Web service

1. In Microsoft Internet Explorer, open <http://localhost/Woodgrove/Bank.asmx>.  
Notice the methods that are implemented in the Woodgrove Online Bank XML Web service.
2. To test three of the methods:
  - a. On the GetAllAccounts test page, click **Invoke**.
  - b. On the GetAccount test page, type **1** for the **acctID**, and then click **Invoke**.
  - c. On the GetTransactionHistory test page, test the method with the following values.

Text boxes	Values
<b>accountID</b>	<b>1</b>
<b>startDate</b>	<b>1/1/2001</b>
<b>endDate</b>	<b>1/1/2010</b>

In the following two procedures of this exercise, you will enable the client application to use the **GetAllAccounts**, **GetAccount**, and **GetTransactionHistory** methods.

3. Close the browser.

### ► Examine the client application

1. In Visual Studio .NET, open the Woodgrove and Contoso Account Manager project, which is located in the folder *<lab root>\Lab04\Starter*.
2. Open the following files and examine the code.

C#	Visual Basic .NET
WebServiceClientForm.cs	WebServiceClientForm.vb

The next step is to generate a proxy class and add a reference to this class to your project.



► **Add a Web reference to the Woodgrove Online Bank XML Web service**

1. In Solution Explorer, right-click the **References** node and click **Add Web Reference**.
2. In the **Add Web Reference** dialog box, enter the URL `http://Localhost/Woodgrove/bank.asmx` in the address field and press ENTER.

You should see the Service Help Page for the Woodgrove XML Web service.

3. At this point in the discovery process, the **Add Reference** button becomes enabled. Click this button to complete adding a Web reference.

► **View and use the Woodgrove proxy classes**

1. To view the proxy classes, open the **Class View** (on the **View** menu, click **Class View**). Expand the XML Web service client namespace.

Notice the **localhost** namespace nested within this namespace. Adding a Web reference to an XML Web service deployed on your local computer creates this namespace and the proxy classes within this namespace.

2. Open the Solution Explorer (on the **View** menu, click **Solution Explorer**).
3. Rename the localhost namespace to **Bank**. To do this:
  - a. Right-click **localhost** and then click **Rename**.
  - b. Type **Bank**
4. Open the **Class View**.
5. Expand the **Bank** namespace and then expand the **WoodgroveOnlineBank** class.

Notice the synchronous and asynchronous methods that the **WoodgroveOnlineBank** proxy class implements.

6. To allow the WebServiceClient application to use the proxy classes without requiring the fully qualified class names, open the file **WebServiceClientForm (.cs or .vb)** and import the **WebServiceClient.Bank** namespace.

## Exercise 2

### Invoking an XML Web Service Method

In this exercise, you will add code to invoke the **GetAllAccounts** method of the **WoodgroveOnlineBank** proxy class to populate the **Account Information** list.

#### ► Add code to the **GetWoodgroveAccountList** method

1. Locate the **GetWoodgroveAccountList** method in the following file.

C#	Visual Basic .NET
WebServiceClientForm.cs	WebServiceClientForm.vb

The **GetWoodgroveAccountList** method is invoked in the **Form1\_Load** event handler when the main Microsoft Windows® Form loads.

2. Within the **try** block, do the following:
  - a. Declare and create an instance of **WoodgroveOnlineBank**.
  - b. Declare an array of **Acct** objects.

C#	Visual Basic .NET
Acct [ ] acct	Dim acct() as Acct

- c. Invoke the **GetAllAccounts** method on the **WoodgroveOnlineBank** object, and assign the result of the method call to the array of **Acct** objects.
- d. If **Acct** objects were returned, iterate over the array of **Acct** objects and test that the array of **Acct** objects that is returned is not null. If the array is not null, iterate through the **Acct** objects and add the **accountID** of each **Acct** object to the **listBoxAccounts** list.

C#	Visual Basic .NET
foreach(Acct account in acct)	Dim account As Acct
{	For Each account In acct
listBoxAccounts.Items.Add(	listBoxAccounts.Items.Add(
string.Format("{0}",	String.Format("{0}",
account.accountID));	account.accountID))
}	Next account

3. Build and run the Woodgrove and Contoso Account Manager application.

The **Accounts for Customer** list box in the **Woodgrove Online Bank** group box will display a list of account IDs.

## Exercise 3

### Using an XML Web Service Method That Returns Derived Types

In this exercise, you will add code to invoke the **GetAccount** method of the **WoodgroveOnlineBank** proxy class to fill the **Account Information** text box. The **GetAccount** method returns an object derived from the type **Acct**.

#### ► Add code to obtain a specific account

1. Locate the **GetWoodgroveAccountInfo** method in the following file.

C#	Visual Basic .NET
WebServiceClientForm.cs	WebServiceClientForm.vb

The **GetWoodgroveAccountInfo** method is invoked in the:

- **listBoxAccounts\_SelectedIndexChanged** event handler when an entry in the **Accounts for Customer** list is selected.
  - **buttonWoodgroveGetAccount\_Click** event handler when the Update Account Info button is clicked.
2. Within the **try** block of the **GetWoodgroveAccountInfo** method, do the following:
    - a. Create a **WoodgroveOnlineBank** object.
    - b. Invoke the **GetAccount** method on the **WoodgroveOnlineBank** object, passing the **acctID** local variable for the **acctID** parameter.
    - c. Save the returned **Acct** object.
    - d. Create a string by concatenating the name and value of each property of the **Acct** object that the **GetAccount** method returns.
  3. In the **Class View**, expand the **WebServiceClient** namespace and then expand the **Bank** namespace. Notice the **Acct**, **CheckingAcct**, and **SavingsAcct** classes within this namespace. Expand the **CheckingAcct** and **SavingsAcct** classes – notice that they are derived from the **Acct** class.

---

**Note** The **GetAccount** method for the **WoodgroveOnlineBank** class returns an **Acct** object. However, the return value can be an object of the derived **CheckingAcct** or **SavingsAcct** class. The type of the account can be determined by examining the value of the **type** data member of the **Acct** object.

---

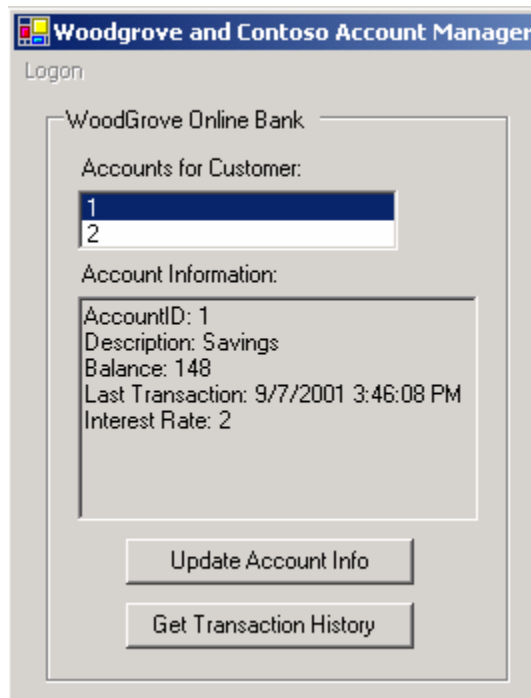
4. If the type data member has the value **SV**, then the object is a **SavingsAcct** object. Therefore, do the following:
  - a. Cast the **Acct** object to a **SavingsAcct** object.

C#	Visual Basic .NET
SavingsAcct acctSV = acct as SavingsAcct;	Dim acctSV As SavingsAcct = CType(acct, SavingsAcct)

- b. Use the **SavingsAcct** object to access the **interestRate** data member, and concatenate it with the display string.

5. Likewise, if the type data member has the value **CK**, then the object is a **CheckingAcct** object. Therefore, do the following:
  - a. Cast the **Acct** object to a **CheckingAcct** object.
  - b. Use the **CheckingAcct** object to access the **MinimumBalance** data member, and concatenate it with the display string.
6. Build and run the Woodgrove and Contoso Account Manager application.
7. In the **Accounts for Customer** list, select one of the listed account numbers.

You should see account information similar to the following in the **Account Information** text box that is contained in the **WoodGrove Online Bank** group box:



The screenshot shows a Windows application titled "Woodgrove and Contoso Account Manager". It features a "Lagon" label at the top. Below it is a group box labeled "WoodGrove Online Bank". Inside this group box, there is a section titled "Accounts for Customer:" containing a list box with two items, "1" and "2", where "1" is selected. Below the list box is a section titled "Account Information:" containing a text box with the following text: "AccountID: 1", "Description: Savings", "Balance: 148", "Last Transaction: 9/7/2001 3:46:08 PM", and "Interest Rate: 2". At the bottom of the group box, there are two buttons: "Update Account Info" and "Get Transaction History".

## Exercise 4

### Using an XML Web Service Method That Returns an ADO .NET Typed Dataset

In this exercise, you will add code to invoke the **GetTransactionHistory** method for the **WoodgroveOnlineBank** proxy class. This method returns an ADO.NET typed **DataSet**, the class **TransactionDataSet**.

#### ► Add code to obtain transactions

1. Locate the **buttonWoodgroveGetTransactions\_Click** method in the following file.

C#	Visual Basic .NET
WebServiceClientForm.cs	WebServiceClientForm.vb

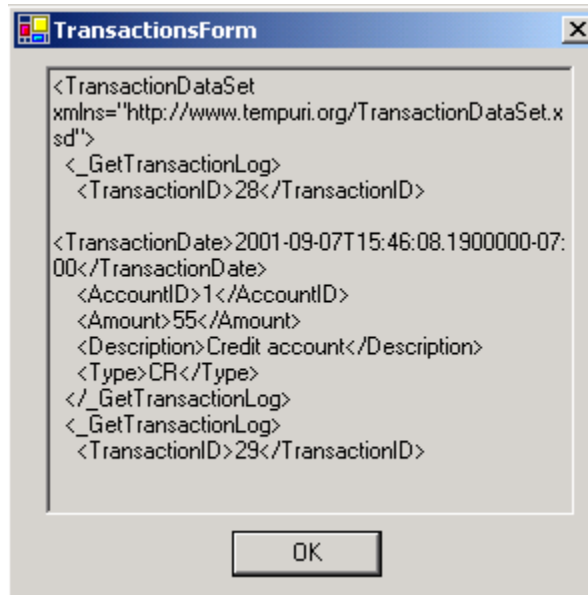
The **buttonWoodgroveGetTransactions\_Click** method is the event handler for the Woodgrove Online Bank **Get Transaction History** button click event.

2. Within the **try** block of the **buttonWoodgroveGetTransactions\_Click** method, do the following:
  - a. Create a **WoodgroveOnlineBank** object.
  - b. Invoke the **GetTransactionHistory** method on the **WoodgroveOnlineBank** object, passing the **acctID** local variable for the **acctID** parameter, **dtStart** as **startDate**, and **DateTime.Now** as **endDate**.
  - c. Save the returned **TransactionDataSet** object in a local variable.
  - d. Use the **TransactionDataSet.\_GetTransactionLog.Count** property to check if records were returned. If not use **MessageBox.Show** method to display an informational message.
  - e. If records were returned then modify the existing code that uses a **TransactionForm** as to call the **DataSetToXMLString** method to obtain a string representation of the **TransactionDataSet** object that was saved in step c.

► **Test the application**

1. Build and run the Woodgrove and Contoso Account Manager application.
2. In the **Woodgrove Online Bank** group box, in the **Accounts for Customer** list, select one of the listed account numbers. Click the **Get Transaction History** button.

You should see transactions that are similar to the following transactions that are listed in the **TransactionsForm**.



## Review

- WSDL Documents
- XML Web Service Discovery
- XML Web Service Proxies
- Implementing an XML Web Service Consumer Using Visual Studio .NET

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

1. What are WSDL documents used for?

**To describe the types, operations, messages, and protocols that an XML Web service supports.**

2. Which tool can you use to discover WSDL documents at a URL?

**Disco.exe**

3. Which tool can you use to generate client proxies for XML Web services?

**Wsd.exe or Visual Studio .NET**

---

4. Which tool can use a WSDL document to generate a class implementing the operations of the XML Web service that are described in the WSDL document?

**WSDL.exe**

5. When you add a Web reference to a client application, how are the types that are exposed by the XML Web service exposed in the client application?

**They are exposed in a nested namespace that is a child of the default namespace of the client application.**



---

## Module 5: Implementing a Simple XML Web Service

### Contents

Overview	1
Creating an XML Web Service Project	2
Implementing XML Web Service Methods	11
Managing State in an ASP.NET XML Web Service	33
Debugging XML Web Services	42
Lab 5.1: Implementing a Simple XML Web Service	61
Review	77



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, places or events 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, trademarks, 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.

© 2001–2002 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Active Directory, Authenticode, IntelliSense, FrontPage, Jscript, MSDN, PowerPoint, Visual C#, Visual Studio, Visual Basic, Windows NT, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

## Instructor Notes

**Presentation:**  
**150 Minutes****Lab:**  
**75 Minutes**

This module provides students with the skills that are required to implement an XML Web service by using Microsoft® Visual Studio® .NET and debug it. Students will also modify the solution to Lab 4.1, “Implementing an XML Web Service Consumer Using Visual Studio .NET,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*, to communicate with the XML Web service that they will create in the lab exercises for this module.

After completing this module, students will be able to:

- Create an XML Web service project.
- Implement XML Web service methods, expose them, and control their behavior.
- Manage state in a Microsoft ASP.NET-based XML Web service.
- Debug XML Web services.

**Required Materials**

To teach this module, you need the Microsoft PowerPoint® file 2524B\_05.ppt.

**Preparation Tasks**

To prepare for this module:

- Read all of the materials for this module.
- Try the demonstrations and study the code examples in this module.
- Complete the lab.

## Demonstration

This section provides demonstration procedures that are not appropriate for the student notes.

### Performing Tracing Using the SOAP Extension Class

1. Open the following files from the *<install folder>\Democode\<language>\Mod05* folder.
- | C#       | Visual Basic .NET |
|----------|-------------------|
| Trace.cs | Trace.vb          |
2. Walk through the code for the **TraceExtension** class, and point out the **ProcessMessage** function and the **WriteInput** and **WriteOutput** functions.
  3. Walk through the code for the **TraceExtensionAttribute** and point out the **Filename** property, the **AttributeUsage** attribute, and the **ExtensionType** property.
  4. Open the following files from the *<install folder>\Democode\<language>\Mod05\ Web References\Woodgrove* folder.

C#	Visual Basic .NET
reference.cs	reference.vb

5. Show how the **TraceExtension** attribute is applied to the **GetAllAccounts** method.
6. Run the test client and then show the log file contents (c:\Log.txt by default).

### Creating an XML Web Service Project

1. Open Visual Studio .NET.
2. On the **File** menu, point to **New**, and click **Project**.
3. Select the language of your choice, then select the **ASP.NET Web Service** project template. Set the project location to <http://localhost/FirstWebService>. Click **OK** to begin.
4. Open the *Service1.aspx* code behind file:
  - a. In Solution Explorer, right-click **Service1.aspx**.
  - b. On the shortcut menu, click **View Code**.
5. Locate the **HelloWorld** method.
6. Uncomment the method.
7. Compile and run the XML web service.
8. Click on the **HelloWorld** link.
9. Click **Invoke**.
10. Close all Internet Explorer windows.

## Module Strategy

Use the following strategy to present this module:

- Creating an XML Web Service Project

This section provides an overview of the mechanics of implementing a simple XML Web service by using the ASP.NET Web Service project template. It also explains the purpose of each of the files that are generated for the default XML Web service project.

- Implementing XML Web Service Methods

This section is intended to help students understand:

- How the various properties of the **WebMethod** attribute affect an XML Web service method.
- How to implement XML Web service methods with parameter lists of varying complexity.

Do not discuss the tradeoffs between the various options for representing parameter lists. Defer this discussion until Module 8, “Designing XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

Be sure to practice the Component Designer and XML Designer demonstrations in this section.

- Managing State in an ASP.NET XML Web Service

This section discusses application and session state management in XML Web services. It covers the mechanics of using application and session state, and not the advantages and disadvantages of using these state services. For students who are familiar with ASP programming, emphasize the similarity in the mechanics of using the application and session state services. Once again, defer any in-depth discussion of issues relating to performance or scaling to Module 8, “Designing XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

- Debugging XML Web Services

This topic is intended to familiarize students with various tools and techniques available for debugging XML Web services. The simplest debugging tools that this section covers are the **Trace** and **Debug** classes. The use of Simple Object Access Protocol (SOAP) extensions to provide more sophisticated tracing is also covered. Do not explain all the details of the **TraceExtension** implementation. Just cover the concepts of hooking the serialization process and the mechanics of associating a SOAP extension with a method through a custom attribute.

The last topic of this section provides a brief overview of application- and page-level tracing using Trace.axd, an event log, and performance counters. Focus on the mechanics and not on the issues that are related to performance, scaling, and deployment.



# Overview

- Creating an XML Web Service Project
- Implementing XML Web Service Methods
- Managing State in an ASP.NET XML Web Service
- Debugging XML Web Services

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

You can implement XML Web services in several ways by using any programming language. For example, you can implement an XML Web service by using Microsoft® Visual C#™, Microsoft Visual Basic®, or Managed Extensions for C++. You can also implement an XML Web service by using the Microsoft Active Template Library (ATL) Server.

To implement XML Web services, you must understand the components of a Microsoft ASP.NET-based XML Web service project and how to expose class methods as XML Web service operations. You must also understand how state can be managed in ASP.NET Web Services and some of the issues related to state management and XML Web services.

Debugging distributed applications is not easy, and XML Web services are not any different in that respect. Therefore, you must be familiar with some of the techniques that you can use to debug XML Web services.

## Objectives

After completing this module, you will be able to:

- Create an XML Web service project.
- Implement XML Web service methods, expose them, and control their behavior.
- Manage state in a Microsoft ASP.NET-based XML Web service.
- Debug XML Web services.

## Creating an XML Web Service Project

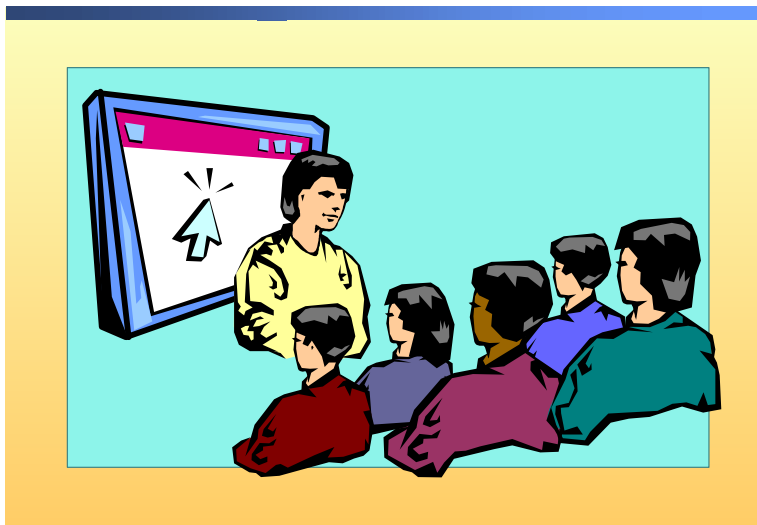
- **Demonstration: Creating an XML Web Service Project**
- **Examining the Parts of an XML Web Service Project**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this section, you will look at the first step in implementing an XML Web service, which is creating the base project. You will also examine the results of generating a project by using the ASP.NET Web Service project template.



## Demonstration: Creating an XML Web Service Project



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this demonstration, you will learn how to create a Visual Basic .NET and C#-based ASP.NET XML Web service project in Microsoft Visual Studio® .NET.

## Examining the Parts of an XML Web Service Project

- **References**
  - **System** namespace
  - **System.Data** namespace
  - **System.Web** namespace
  - **System.Web.Services** namespace
  - **System.XML** namespace
- **The .asmx file**
  - Service Help page
  - Service Method Help page
  - Service Description page

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

<b>Introduction</b>	ASP.NET XML Web services are ASP.NET applications; therefore, there are many elements that are common between them. This topic examines the various parts of the base project for an ASP.NET XML Web service, which include references and the .asmx file.
<b>References</b>	By default, references to a list of Microsoft .NET Framework namespaces are included in the base project. This list of namespaces includes the <b>System</b> , <b>System.Data</b> , <b>System.Web</b> , <b>System.Web.Services</b> , and <b>System.XML</b> namespaces.
<b>System namespace</b>	The <b>System</b> namespace contains classes that define commonly-used values and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.
<b>System.Data namespace</b>	<p>The <b>System.Data</b> namespace consists primarily of the classes that constitute the Microsoft ADO.NET architecture. The <b>DataSet</b> class plays a central role in the ADO.NET architecture. A DataSet is an in-memory cache of data obtained from many possible data sources, such as databases or Extensible Markup Language (XML) documents. A DataSet reads and writes data and schema as XML documents. In this XML format, any application can use DataSets on any platform that supports XML.</p> <p>You will learn about DataSets in detail in the next section.</p>
<b>System.Web namespace</b>	The <b>System.Web</b> namespace supplies classes and interfaces that facilitate communications between a browser and a server. This namespace includes the <b>HttpRequest</b> and <b>HttpResponse</b> classes that are discussed in Module 3, “The Underlying Technologies of XML Web Services,” in Course 2524B, <i>Developing XML Web Services Using Microsoft ASP.NET</i> . <b>System.Web</b> also includes classes for cookie manipulation, file transfer, exception information, and output cache control. You will revisit this namespace later in this module when you look at improving XML Web Service performance by using caching, and cookie-based and cookieless authentication.

**System.Web.Services namespace**

The **System.Web.Services** namespace consists of classes that help you to build and use XML Web Services. One of the most important classes in this namespace is the **WebService** class. If an XML Web service needs access to the ASP.NET intrinsic (built-in) objects, then the class which implements the XML Web Service operations must be derived from the **WebService** class.

**System.XML namespace**

The **System.XML** namespace exposes the XML classes that provide standards-based support for processing XML. The supported standards are:

- XML 1.0

The **XmlTextReader** class provides a parser for documents in XML 1.0.

- XML namespaces

The .NET Framework supports the use of namespaces in both XML streams and the Document Object Model (DOM).

- XML schemas

The .NET Framework supports schema mapping and XML serialization. However, it does not support validation by using XSD.

- XML Path Language (XPath) expressions

The **XPathNavigator** class provides read-only, random access to XML documents by using XPath expressions.

- Extensible Stylesheet Language Transformations (XSLT)

XSLT allows you to transform XML data by using XSLT style sheets.

- DOM

The **XmlDocument** class implements the World Wide Web Consortium (W3C) DOM Level 1 Core and DOM Level 2 Core specifications.

- Simple Object Access Protocol (SOAP) 1.1

You can find the classes that encapsulate SOAP support in the .NET Framework in the **System.Web.Services.Protocols** namespace.

**The .asmx file**

The .asmx file is the front-end for an XML Web service that is implemented by using ASP.NET. The way that you access this file through HTTP determines the type of response that you receive.

By default, the .asmx file contains a class that is similar to the following example class.

C#	Visual Basic .NET
public class Service1 : System.Web.Services.WebService {	Public Class Service1 Inherits ↳System.Web.Services.WebServi ↳ce

The class contains a placeholder WebMethod implementation that is similar to the following code.

```
C#  
[WebMethod]  
public string HelloWorld()  
{  
    return "Hello World";  
}
```

---

**Visual Basic .NET**

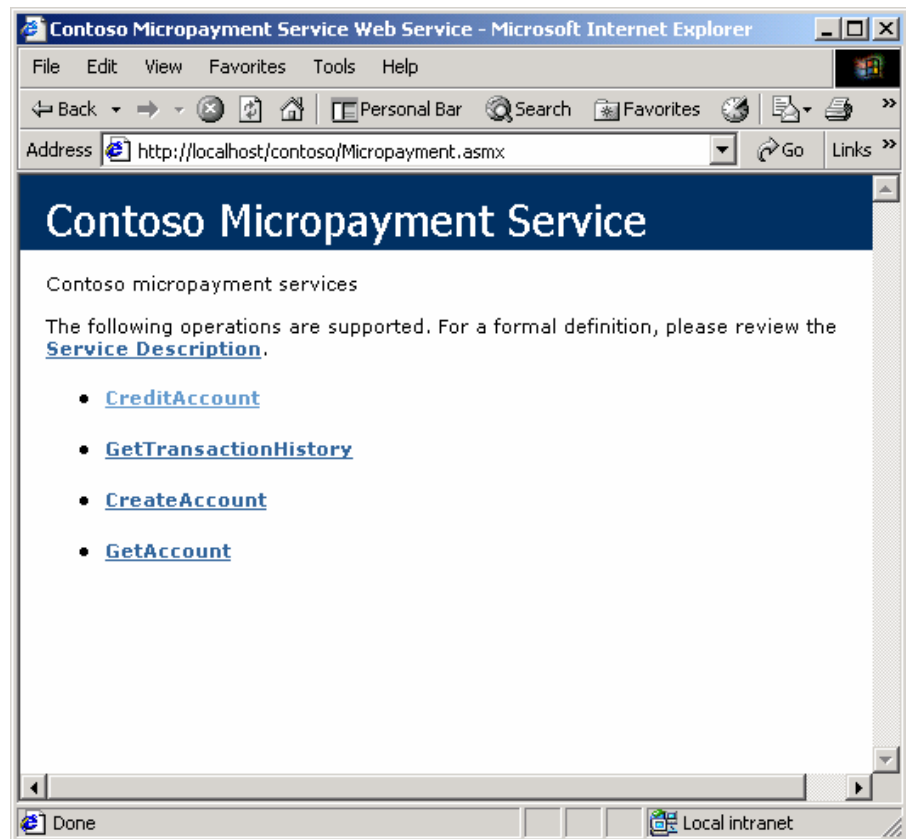
---

```
<WebMethod()> Public Function HelloWorld() As String  
    HelloWorld = "Hello World"  
End Function
```

**The Service Help page**

When you request an .asmx file from a Web browser without supplying a recognized query string, the file returns an automatically generated Service Help page for the XML Web service. If you performed an HTTP-GET request for the .asmx page without supplying a query string, the results would be the same. A Service Help page provides a list of the XML Web service methods that can be accessed programmatically. The page contains links for each method, and each of these links will take you to a Service Method Help page for the corresponding method.

A Service Help page also contains a link to the corresponding XML Web service description document as shown in the following illustration.



To access the Service Help page of an XML Web service:

- From your browser, navigate to the base URL for the corresponding XML Web service, using the following format:

`http://servername/projectname/webservicename.asmx`

The following table describes the parts of the preceding URL.

Part	Value
<i>servername</i>	The server on which the XML Web service resides.
<i>projectname</i>	The name of the XML Web service project and any additional path information that is needed to access the .asmx file for the XML Web service.
<i>webservicename.asmx</i>	The name of the .asmx file for the XML Web service.

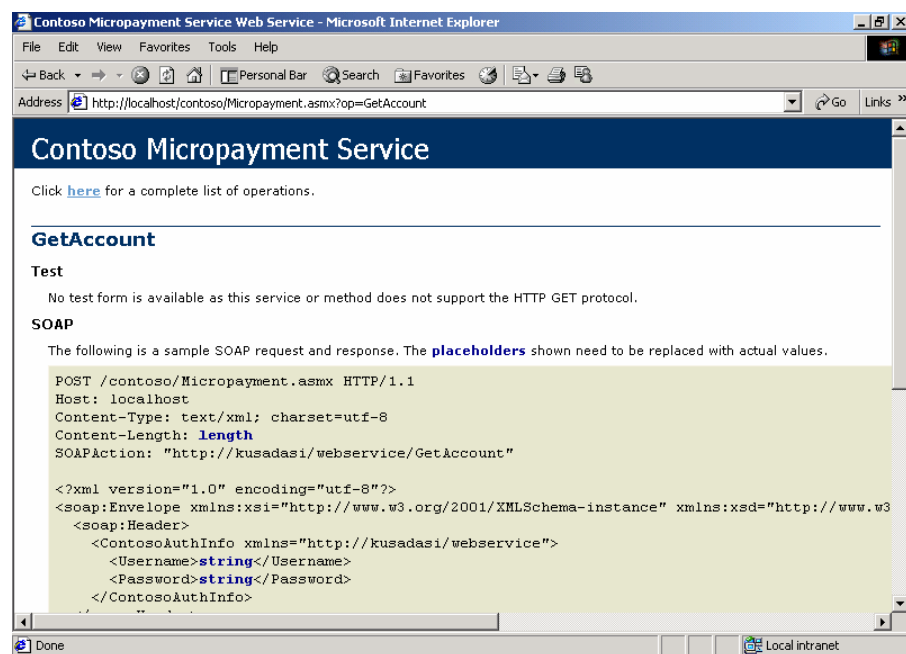
### The Service Method Help page

The Service Method Help page of an XML Web service provides additional information about a specific method of the XML Web service.

The page also allows you to invoke the method if it can be invoked by using the HTTP-POST protocol.

Sample request and response messages for the protocols that the XML Web service method supports are provided at the bottom of the Service Method Help page.

The following illustration shows the Service Method Help page.



### The Service Description page

The service description for an XML Web service is a Web Services Description Language (WSDL) document. The Service Help page provides a link to the service description. You can also access the service description of an XML Web service from a browser by typing in the base URL for the XML Web service, similar to the way that you access a Service Help page. However, you also need to supply the query string **WSDL** as shown in the following example:

`http://servername/projectname/webservicename.asmx?WSDL`

## Examining the Parts of an XML Web Service Project (*continued*)

- Global.asax
- Web.config
- The .vsdisco file
- AssemblyInfo (.cs or .vb)
- The /bin folder

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Apart from the list of references and the .asmx file for an XML Web service, the XML Web service project consists of several other entries.

### Global.asax

The Global.asax file is an optional file that contains the code for responding to application-level events that ASP.NET or the **HttpModule** class raises. The Global.asax file resides in the root directory of an ASP.NET application. At run time, Global.asax is parsed and compiled into a dynamically-generated .NET Framework class that is derived from the **HttpApplication** base class. The Global.asax file itself is configured so that any direct URL request for the file is automatically rejected. Therefore, external users cannot download or view the code written within the Global.asax.

When you save changes to a Global.asax file that is in use, the ASP.NET framework detects that the file has been changed. It completes all of the current requests for the application, sends the **Application\_OnEnd** event to listeners (if any), and restarts the application domain. In effect, the previous set of actions restarts the application, closes all browser sessions, and flushes all state information from the memory. When a new request arrives from a browser, the ASP.NET framework re-parses and recompiles the Global.asax file and fires the **Application\_OnStart** event.

### Web.config

It is essential for XML Web service developers to be able to implement configuration settings without embedding values into the code, and for Web site administrators to be able to easily adjust configuration settings of a deployed XML Web service. ASP.NET XML Web Services provide this capability through a file named Web.config.

The following list includes the features of Web.config:

- Web.config is an XML-based text file. You can use any standard text editor or XML parser to create and edit the file.
- Web.config applies configuration settings to the folder in which it resides and to all of its child folders.

Configuration files in child folders can supply configuration settings, in addition to the settings that are inherited from parent folders. The configuration settings for the child folder can override or modify the settings that are defined in parent directories. A root configuration file named Machine.config, located at

C:\WINNT\Microsoft.NET\Framework\version\CONFIG, provides ASP.NET configuration settings for the entire Web server.

- At run time, ASP.NET uses the configuration settings that the Web.config file provides to compute a collection of configuration settings for an ASP.NET application. The resulting configuration settings are then cached for all subsequent requests for a resource.
- ASP.NET detects changes to Web.config and automatically applies the new settings to the affected resources. The server does not need to be restarted for the changes to take effect.
- Web.config is extensible. You can define new configuration parameters and write configuration handlers to process them.
- ASP.NET protects Web.config from external access by configuring Microsoft Internet Information Services (IIS) to prevent access to configuration files directly from the browser. HTTP access error 403 (forbidden) is returned to any browser that attempts to directly request a configuration file.

#### The .vsdisco file

The .vsdisco file is a dynamic discovery document. When you deploy an XML Web service into a production environment you should only deploy the .vsdisco file if you want the XML Web service to be dynamically discoverable. This is generally not recommended.

#### AssemblyInfo (.cs or .vb)

AssemblyInfo (.cs or .vb) is a project information file that contains metadata, such as name, version, and culture information, about the assemblies in a project. This file is compiled into the XML Web service assembly. For more information about AssemblyInfo, see Course 2350A, *Securing and Deploying Microsoft .NET Assemblies*.

#### The /bin folder

Below your project root folder is a folder named **bin**. The **bin** folder contains the assembly that contains the compiled output of the project. The assembly is first compiled to Microsoft intermediate language (MSIL) code when an XML Web service project is compiled, and then the assembly is just-in-time (JIT)-compiled to native code on demand.



## Implementing XML Web Service Methods

- Exposing XML Web Service Methods
- Examining Data Types and Parameter Lists
- Demonstration: Creating a Typed DataSet Using the Component Designer
- Demonstration: Creating a Typed DataSet Using the XML Designer

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

After creating an XML Web service project, the next step in implementing an XML Web service is to define its operations. In this section, you learn how to expose methods that you implement as XML Web service operations, how to control the serialization behavior of a method, and finally examine how to implement methods with parameter lists and return types of varying complexity.

## Exposing XML Web Service Methods

- Applying the **WebMethod** attribute
- Configuring the **WebMethod** attribute properties
  - **BufferResponse**
  - **CacheDuration**
  - **Description**
  - **EnableSession**
  - **MessageName**
  - **TransactionOption**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

You expose an XML Web service method by applying the **WebMethod** attribute to the method. You can also control the behavior of the method by configuring the properties of the **WebMethod** attribute.

### Applying the **WebMethod** attribute

To expose a method of an XML Web service, you must do the following:

- Specify that the method is public.
- Apply the **WebMethod** attribute to the method.

---

**Note** Even if an XML Web service method is specified as public, you still need to attach the **WebMethod** attribute to expose it as part of an XML Web service.

---

### Configuring the **WebMethod** attribute properties

You can control the behavior of an XML Web service method by configuring the properties of the **WebMethod** attribute.

The **WebMethod** attribute has the following properties:

- **BufferResponse**
- **CacheDuration**
- **Description**
- **EnableSession**
- **MessageName**
- **TransactionOption**

**BufferResponse**

The **BufferResponse** property of the **WebMethod** attribute enables buffering of responses for an XML Web service method. When **BufferResponse** is set to **true**, which is the default setting, ASP.NET buffers the entire response before sending it to the client. The buffering is very efficient and helps improve performance by minimizing communication between the worker process and the IIS process. When **BufferResponse** is set to **false**, ASP.NET buffers the response in 16 KB pieces of data. Typically, you would set this property to **false** only if you did not want the entire contents of the response in memory at the same time. For example, if you are returning a collection whose items are streamed out of a database to a client, you might not want to wait to send the first byte to the client until the last byte has been retrieved from the database.

You can set the **BufferResponse** property as shown in the following example.

**C#**

---

```
[WebMethod(BufferResponse=false)]
public Transactions GetTransactionHistory() {
    //implementation code
}
```

**Visual Basic .NET**

---

```
Public<WebMethod(BufferResponse := False)> _
Function GetTransactionHistory() As Transactions
    'implementation code
End Function 'GetTransactionHistory
```

**CacheDuration**

The **CacheDuration** property enables caching of the results for an XML Web service method. This is known as output caching. ASP.NET caches the results for each unique parameter set. A *parameter set* is a set of values that are supplied as arguments to an operation. Each unique parameter set is associated with a cached response. The value of the **CacheDuration** property specifies the time duration (in seconds) for which ASP.NET must cache the response. The default value of zero disables the caching of results. For more information about output caching, see Module 8, “Designing an XML Web Service,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

You can set the **CacheDuration** property as shown in the following example.

**C#**

---

```
[WebMethod(CacheDuration=60)]
public double ConvertTemperature(double dFahrenheit){
    return ((dFahrenheit - 32) * 5) / 9;
}
```

**Visual Basic .NET**

---

```
Public<WebMethod(CacheDuration := 60)> _
Function ConvertTemperature(dFahrenheit As Double) As Double
    Return(dFahrenheit - 32) * 5 / 9
End Function 'ConvertTemperature
```

**Description**

The **Description** property supplies a description for an XML Web service method that will appear on the Service Help page. Unless set otherwise, the default value for this property is an empty string.

You can set the **Description** property as shown in the following example.

C#

---

```
[WebMethod(Description="This method converts a temperature in
↪degrees Fahrenheit to a temperature in degrees Celsius.")]
public double ConvertTemperature(double dFahrenheit) {
    return ((dFahrenheit - 32) * 5) / 9;
}
```

Visual Basic .NET

---

```
Public<WebMethod(Description := "This method converts a
↪temperature in degrees Fahrenheit to a temperature in
↪degrees Celsius.")> _
Function ConvertTemperature(dFahrenheit As Double) As Double
    Return(dFahrenheit - 32) * 5 / 9
End Function 'ConvertTemperature
```

## EnableSession

The **EnableSession** property enables session state for an XML Web service method. If an XML Web service method supports session state, the XML Web service can access the session state collection directly from the **HttpContext.Current.Session** property or with the **WebService.Session** property if the method inherits from the **WebService** base class. The default value of the **EnableSession** property is **false**. You will learn more about session state management later in this module. For more information about session state and state management in general from the perspective of performance and scalability, see Module 8, “Designing XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

You can set the **EnableSession** property as shown in the following example.

C#

---

```
[WebMethod(EnableSession=true)]
public double ConvertTemperature(double dFahrenheit) {
    Session("NumberOfConversions") =
↪Session("NumberOfConversions") + 1;
    return ((dFahrenheit - 32) * 5) / 9;
}
[WebMethod(EnableSession=true)]
public int GetNumberOfConversions() {
    return Session("NumberOfConversions");
}
```

Visual Basic .NET

---

```
Public<WebMethod(EnableSession := True)> _
Function ConvertTemperature(dFahrenheit As Double) As Double
    Session("NumberOfConversions") =
↪Session("NumberOfConversions") + 1
    Return(dFahrenheit - 32) * 5 / 9
End Function 'ConvertTemperature

Public<WebMethod(EnableSession := True)> _
Function GetNumberOfConversions() As Integer
    Return Session("NumberOfConversions")
End Function 'GetNumberOfConversions
```

**MessageName**

The **MessageName** property enables the XML Web service to uniquely identify overloaded methods by using an alias. The default value for this property is the method name. If you set the **MessageName** property to a different value, the resulting SOAP messages will reflect this name instead of the actual method name.

You can set the **MessageName** property as shown in the following example.

**C#**


---

```
[WebMethod(MessageName="AddDoubles")]
public double Add(double dValueOne, double dValueTwo) {
    return dValueOne + dValueTwo;
}
[WebMethod(MessageName="AddIntegers")]
public int Add(int iValueOne, int iValueTwo) {
    return iValueOne + iValueTwo;
}
```

**Visual Basic .NET**


---

```
Public<WebMethod(MessageName := "AddDoubles")> _
Function Add(dValueOne As Double, dValueTwo As Double) As
    Double
    Return dValueOne + dValueTwo
End Function 'Add

Public<WebMethod(MessageName := "AddIntegers")> _
Function Add(iValueOne As Integer, iValueTwo As Integer) As
    Integer
    Return iValueOne + iValueTwo
End Function 'Add
```

The **AddDoubles** SOAP request message for the method that adds two values that are of type **System.Double** will resemble the following:

```
POST /myWebService/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/AddDoubles"
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <AddDoubles xmlns="http://tempuri.org/">
      <dValueOne>double</dValueOne>
      <dValueTwo>double</dValueTwo>
    </AddDoubles>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
```

In the preceding code, notice that the name of the operation is not **Add**, but it is the name specified in the **MessageProperty**, which is **AddDoubles**.

The **AddDoubles** SOAP response message for the method would resemble the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope namespaces omitted for brevity>
  <soap:Body>
    <AddDoublesResponse xmlns="http://tempuri.org/">
      <AddDoublesResult>double</AddDoublesResult>
    </AddDoublesResponse>
  </soap:Body>
</soap:Envelope>
```

### TransactionOption

The **TransactionOption** property enables an XML Web service method to participate as the root object of a Microsoft Distributed Transaction Coordinator (MS DTC) transaction. Even though you can assign the **TransactionOption** property with any of the values of the **System.EnterpriseServices.TransactionOption** enumeration, an XML Web service method has only two possible behaviors:

- The method does not participate in a transaction (**Disabled**, **NotSupported**, **Supported**).
- The method initiates a new transaction (**Required**, **RequiresNew**).

The default value for the **TransactionOption** property is **TransactionOption.Disabled**. Before you can use the **TransactionOption** property, you must add a reference to **System.EnterpriseServices.dll** to your project. This assembly contains the **System.EnterpriseServices** namespace, which has methods and properties that expose the distributed transaction model that you find in Microsoft Component Object Model (COM+) services. The **System.EnterpriseServices.ContextUtil** class lets you vote on the outcome of a transaction by using the **SetAbort** or **SetComplete** methods.

In the following procedure, you will learn how to configure an XML Web service method to initiate a new MS DTC transaction.

To initiate a new DTC transaction:

1. Add a reference to System.EnterpriseServices.dll.
2. Add the **System.EnterpriseServices** namespace to the XML Web service.
3. Use the **TransactionOption** property as shown in the following code:

**C#**

---

```
public class Service1 : System.Web.Services.WebService
{
    [WebMethod(TransactionOption=TransactionOption.RequiresNew)]
    public string DoSomethingTransactional()
    {
        // The transaction was successful...
        ContextUtil.SetComplete();
        return ContextUtil.TransactionId;
    }
}
```

**Visual Basic .NET**

---

```
Public Class Service1
    Inherits System.Web.Services.WebService

    Public<WebMethod(TransactionOption := TransactionOption.RequiresNew)> _
        Function DoSomethingTransactional() As String
            ' The transaction was successful...
            ContextUtil.SetComplete()
            Return ContextUtil.TransactionId
        End Function 'DoSomethingTransactional
End Class 'Service1
```

## Examining Data Types and Parameter Lists

- Simple data types
- Input and output parameters
- Variable length parameter lists
- Complex data types
  - Classes and structures
  - Arrays
  - Collections
  - DataSets

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

When you implement XML Web service methods, the choice of data types that are used as parameter and return types can affect the protocols that can be used. A complete discussion of the implications of the choice of different data types will be deferred until Module 8 of Course 2524B, *Developing XML Web Services Using Microsoft Visual Studio .NET*. This topic examines how parameter lists of varying complexity can be used in XML Web service methods.

### Simple data types

You can use simple data types such as integers, strings, and floating point numbers as parameters, or as return values of XML Web service methods. The parameters are marshaled as XSD intrinsic data types.

### Input and output parameters

For C#, all of the **in** and **ref** parameters, and for Visual Basic .NET, all **ByVal** and **ByRef** parameters are defined in a WSDL document as part of the inbound message for an operation. Similarly, for C# any **out** and **ref** parameters, and for Visual Basic .NET, any **ByRef** parameters, and the function return value are defined as part of the outbound message for an operation. The only complexity that you must address in your code is when you pass object references. We will discuss this issue in detail later in this module.

### Variable length parameter lists

XML Web service methods can handle variable length parameter lists. The parameter lists can be *homogeneous* (all of the references in a list are of the same data type) or *heterogeneous* (not all of the references in a list are of the same data type).

Implementing an XML Web service method that has a variable length parameter list is no different from implementing a method on any class that takes a variable length parameter list.



**Examples**

The following examples show a method that takes a variable length list of strings as its argument, and a method that takes a variable length list of objects as its argument:

**C#**


---

```
[WebMethod]
public int ListOfString(params string [] list)
{ return list.Length; }
[WebMethod]
public int ListOfThings(params object [] list)
{ return list.Length; }
```

**Visual Basic .NET**


---

```
Public<WebMethod()> _
Function ListOfString(ParamArray list() As String) As Integer
    Return list.Length
End Function 'ListOfString

Public<WebMethod()> _
Function ListOfThings(ParamArray list() As Object) As Integer
    Return list.Length
End Function 'ListOfThings
```

By default, the XML Web service proxy methods that are generated for these XML Web service methods will take arrays of string and list respectively. For example, the proxy method for ListOfThings will resemble the following code.

**C#**


---

```
public int ListOfThings(object[] list) {
    object[] results = this.Invoke("ListOfThings", new
object[] { list });
    return ((int)(results[0]));
}
```

**Visual Basic .NET**


---

```
Public Function ListOfThings(ByVal list() As Object) As
Integer
    Dim results() As Object = Me.Invoke("ListOfThings", New
Object() { list })
    Return CType(results(0), Integer)
End Function
```

To call a method with a variable length argument list, you must manually add the **params** keyword for C# and the **ParamArray** keyword for Visual Basic .NET as shown in the following code.

#### C#

---

```
public int ListOfThings(params object[] list) {
    object[] results = this.Invoke("ListOfThings", new object[]
{list});
    return ((int)(results[0]));
}
```

#### Visual Basic .NET

---

```
Public Function ListOfThings(ParamArray list() As Object) As
Integer
    Dim results() As Object = Me.Invoke("ListOfThings", New
Object() {list})
    Return CType(results(0),Integer)
End Function
```

Remember that all of the preceding code is generated code. Therefore, if you regenerate the proxy, then you must edit the proxy code again. Otherwise, you can copy the modified code into another file so that it will not be overwritten.

The following is an example of calling an XML Web service method with varying argument lists.

#### C#

---

```
Fancy f = new Fancy();
Console.WriteLine("I sent {0}",f.ListOfThings ("one","two"));
Console.WriteLine("I sent {0}",f.ListOfThings (2,"one",3.5));
```

#### Visual Basic .NET

---

```
Dim f As New Fancy()
Console.WriteLine("I sent {0}", f.ListOfThings("one", "two"))
Console.WriteLine("I sent {0}", f.ListOfThings(2, "one", 3.5))
```

### Complex data types

For XML Web services to be widely adopted, it is necessary that XML Web service methods support complex data types as arguments and return values in addition to supporting simple data types. This section examines some examples of XML Web service methods that use complex data types.

**Classes and structures**

It is important to remember that XML Web services do not perform object remoting. In other words, when a proxy class is generated for an XML Web service, state information is never transmitted to the client. Nor is the object that implements the XML Web service methods persistent on the server side. This is because HTTP is stateless. For these reasons, if you specify a class as a parameter or a return type, you must think of the class as a structure.

When an instance of a class is passed between an XML Web service consumer and an XML Web service, the state of the object is serialized. The state that is serialized includes all read/write public properties and all read/write public fields.

Any class that is used as an argument to an XML Web service method must also have a default constructor. This is because, internally, the XML Web services infrastructure must be able to rebuild the object when it is passed into an XML Web service. Unlike classes, structures always have a default constructor, and therefore you do not have to define a default constructor to use a structure as a parameter for an XML Web service method.

**Arrays**

Arrays of simple types, structures, and classes require no special treatment. However, if you pass in or return an array of object references that include references to derived objects, you must explicitly specify all of the possible types for objects that can be in the array. The types are specified by using the **XmlInclude** attribute. If the types are not specified, then an exception will be thrown when the array is serialized.

**C# and Visual Basic .NET code examples**

Consider the following class definitions.

C#	Visual Basic .NET
<pre>public class Acct {     public string Description;     public string Number;     public string Type;     public decimal Balance;     public string Status; }</pre>	<pre>Public Class Acct     Public Description As String     Public Number As String     Public Type As String     Public Balance As Decimal     Public Status As String End Class 'Acct</pre>
<pre>public class SavingsAcct : ↳ Acct {     public decimal ↳ MinimumBalance; }</pre>	<pre>Public Class SavingsAcct     Inherits Acct     Public MinimumBalance As Decimal End Class 'SavingsAcct</pre>
<pre>public class CreditCardAcct :↳ Acct {     public int PayPeriod; }</pre>	<pre>Public Class CreditCardAcct     Inherits Acct     Public PayPeriod As Integer End Class 'CreditCardAcct</pre>

The following code is an example of an XML Web service method returning an array of **Acct** where the actual elements are derived from **Acct**.

#### C#

---

```
[WebMethod]
[XmlInclude(typeof(CreditCardAcct))]
[XmlInclude(typeof(SavingsAcct))]
[return:XmlArray("AccountList")]
[return:XmlArrayItem("Account")]
public Acct[] GetAllAccounts()
{
    SavingsAcct a = new SavingsAcct();
    CreditCardAcct cc = new CreditCardAcct();
    // populate the accounts
    Acct [] sa = new Acct[2];
    sa[0] = a;
    sa[1] = cc;
    return sa;
}
```

#### Visual Basic .NET

---

```
Public<WebMethod(), XmlInclude(GetType(CreditCardAcct)),
    ↳XmlInclude(GetType(SavingsAcct)),return:
    ↳XmlArray("AccountList")> _
Function GetAllAccounts() As<XmlArrayItem("Account")> Acct()
    Dim a As New SavingsAcct()
    Dim cc As New CreditCardAcct()
    ' populate the accounts
    Dim sa(2) As Acct
    sa(0) = a
    sa(1) = cc
    Return sa
End Function 'GetAllAccounts
```

**Collections**

You must treat collections in the same way as arrays of objects. The following code is an example of an XML Web service method returning an **ArrayList** of **Acct**, where the actual elements are derived from **Acct**.

**C# and Visual Basic .NET code examples****C#**


---

```
[WebMethod]
[XmlInclude(typeof(CreditCardAcct))]
[XmlInclude(typeof(SavingsAcct))]
[return:XmlArray("AccountList")]
[return:XmlArrayItem("Account")]
public ArrayList GetAllAccounts()
{
    SavingsAcct a = new SavingsAcct();
    CreditCardAcct cc = new CreditCardAcct();
    // populate accounts
    ArrayList listOfAccts = new ArrayList();
    listOfAccts.Add(a);
    listOfAccts.Add(cc);
    return listOfAccts;
}
```

**Visual Basic .NET**


---

```
Public<WebMethod(), XmlInclude(GetType(CreditCardAcct)),
    ↵XmlInclude(GetType(SavingsAcct)), return:
    ↵XmlArray("AccountList")> _
Function GetAllAccounts() As<XmlArrayItem("Account")>
    ↵ArrayList
    Dim a As New SavingsAcct()
    Dim cc As New CreditCardAcct()
    ' populate accounts
    Dim listOfAccts As New ArrayList()
    listOfAccts.Add(a)
    listOfAccts.Add(cc)
    Return listOfAccts
End Function 'GetAllAccounts
```

**DataSets**

DataSets present no special problems as arguments. DataSets may be either untyped or typed.

A *typed* DataSet is a generated **DataSet** class that is derived from the base **DataSet** class. A typed DataSet also uses an XSD schema to define tables, columns, and so on. When you generate a typed DataSet, the columns of the underlying query results are exposed as strongly typed properties of the DataSet.

An *untyped* DataSet has no built-in schema. As with a typed DataSet, an untyped DataSet contains tables, columns, and so on. However, they are only exposed as collections.

Visual Studio .NET provides better support for typed DataSets than untyped DataSets. Using typed DataSets makes programming with a DataSet easier and less error-prone. Therefore, it is recommended that you use typed DataSets whenever possible instead of untyped DataSets.

Visual Studio .NET provides the following tools to generate typed DataSets:

- The Component Designer
- The XML Designer

To select the right tool for your requirements, you must be aware of the limitations of both of these tools.

The disadvantages of using the Component Designer are:

- Initially, you have no direct control over the schema. You can control a DataSet definition only by configuring the data adapter from which Visual Studio .NET generates the schema. After generating the schema, you can edit it by using the XML Designer.
- You must manually regenerate the schema and its corresponding **DataSet** class file after changing a data adapter.

For example, if you change the Structured Query Language (SQL) statement that is used to fill a DataSet, then you must change the schema of the DataSet also.

- There are some DataSet functions that you cannot perform by using the Component Designer, such as defining **DataRelation** objects. In such cases, you must use the XML Designer.

The disadvantages of using the XML Designer are:

- The XML Designer offers a slightly lower-level of integration with other data tools in Visual Studio .NET.
- The .xsd file is not validated against any external data source.

For example, if you use the XML Designer to modify the schema of a DataSet, then you must also modify any data adapters that interact with the DataSet.

- A certain level of familiarity with both XML schemas and XML is required to be able to use the XML Designer.

**C# and Visual Basic  
.NET code examples**

The following example code shows an XML Web service method returning a typed DataSet.

C#

---

```
[WebMethod]
public TransactionHistory GetTransactionHistory(int
↳accountID,↳
    DateTime startDate,DateTime endDate)
{
    string connString = (string)
↳ConfigurationSettings.AppSettings["connectString"];

    SqlConnection conn = new SqlConnection(connString);
    SqlDataAdapter adapter = new SqlDataAdapter();
    adapter.TableMappings.Add("Table", "AnAccount");
    conn.Open();
    string cmdText = string.Format("SELECT * FROM
        TransactionLog WHERE AccountID='{0}' AND
        (TransactionDate >='{1}') AND (TransactionDate
↳<='{2}')" ,↳
        accountID,startDate,endDate);
    SqlCommand cmd= new SqlCommand(cmdText,conn);
    cmd.CommandType = CommandType.Text;

    adapter.SelectCommand = cmd;
    TransactionHistory ds = new TransactionHistory();
    adapter.Fill(ds);
    conn.Close();
    return ds; //return ds.GetXml();
}
```

**Visual Basic .NET**

---

```

Public<WebMethod()> _
Function GetTransactionHistory(accountID As Integer, startDate
    ↪As DateTime, endDate As DateTime) As TransactionHistory↪
    Dim connString As String =
    ↪CStr(ConfigurationSettings.AppSettings("connectString"))

    Dim conn As New SqlConnection(connString)
    Dim adapter As New SqlDataAdapter()
    adapter.TableMappings.Add("Table", "AnAccount")
    conn.Open()
    Dim cmdText As String = String.Format("SELECT * FROM" + _
        " TransactionLog WHERE AccountID='{0}' AND " + _
        "(TransactionDate >='{1}') AND (TransactionDate
    ↪<='{2}')" , _
        accountID,startDate,endDate)

    Dim cmd As New SqlCommand(cmdText, conn)
    cmd.CommandType = CommandType.Text

    adapter.SelectCommand = cmd
    Dim ds As New TransactionHistory()
    adapter.Fill(ds)
    conn.Close()
    Return ds 'return ds.GetXml();
End Function

```

---

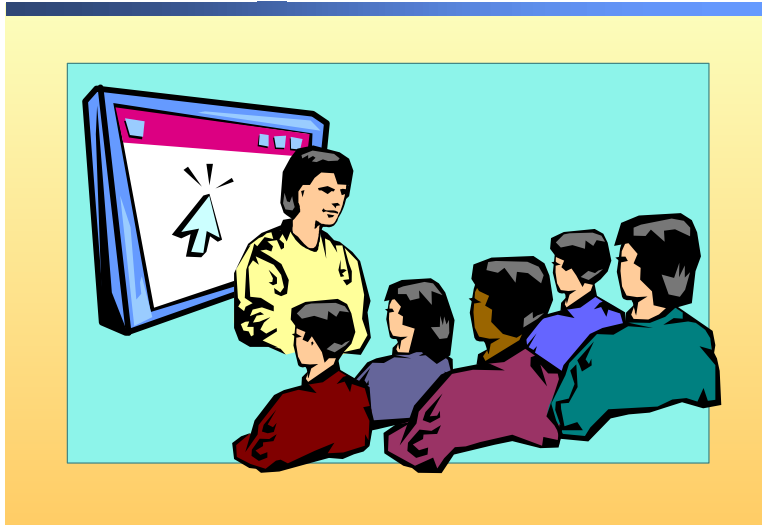
**Note** One important issue that you need to be aware of is that any changes to the schema of a typed DataSet after it is created may not be passed between an XML Web service and a client.

For example, if you have a typed DataSet containing typed tables named **Customers** and **Orders**, then if you add a non-typed table named **Table1** by using the **DataSet.Tables.Add** method-for example, **ds.Tables.Add("Table1")**-then **Table1** may not be propagated back to the client.

---



## Demonstration: Creating a Typed DataSet Using the Component Designer



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this demonstration, you will learn how to create a typed DataSet by using the Component Designer, to use it as an argument in your XML Web service methods, or as a return type of your XML Web service method.

The high-level steps for creating a typed DataSet are as follows:

1. Generate a **SqlDataAdapter**.
2. Create a typed DataSet by using the data adaptor that you created in the previous step.

---

**Note** The **SqlDataAdapter** is a class in ADO.NET, which represents a set of commands and a connection to a database. **SqlDataAdapter** is used to populate a DataSet.

---

To add and configure a **SqlDataAdapter**:

1. Create an ASP.NET Web service.
2. Open the **Design** view of the .asmx file (by default this file is named Service1.asmx). In Solution Explorer, double-click **Service1.asmx**.
3. On the **View** menu, click **Toolbox**.
4. In the **Toolbox**, click the **Data** tab.

5. In the **Toolbox**, drag a **SqlDataAdapter** object to the **Design** view of the .asmx file.
  - a. Step through the **Data Adapter Configuration Wizard**.
  - b. In the **Choose Your Data Connection** page, click **New Connection**.
  - c. In the **Datalink** dialog box, on the **Connection** tab, specify the following values for the various options to connect to a server running Microsoft SQL Server™.

Options	Values
Select the name of your SQL Server.	.MOC (dot followed by MOC)
Use a specific user name and password.	User name: <b>sa</b> Password: <b>Course_2524</b>
Select the database on the server option.	<b>Contoso</b>

- d. Click **Next** to continue the wizard configuration.
- e. You will be again prompted to type your *SQL Server name*, *Username*, and *Password*. Type the same information that you typed previously and click **OK**.
- f. On the **Choose a Query Type** page, select **Use existing stored procedures**. Click **Next**.
- g. On the **Bind Commands to Existing Stored Procedures** page, in the **Select** list, click **\_GetAccount**. Leave the **Insert**, **Update**, and **Delete** lists empty. Click **Next**.  
You will be prompted to enter your *SQL Server Username* and *password* again. Type the same information that you typed previously and click **OK**.
- h. On the **View Results** page of the wizard, click **Finish**.

Notice that the controls named **sqlDataAdapter1** and **sqlConnection1** appear in the **Design** view of the XML Web service implementation file.

You have now added code to your application that will create a Microsoft SQL Server™ connection by using a **SqlServerConnection** object, and then use a **SqlDataAdapter** object to invoke the **\_GetAccount** stored procedure.

Next, you need to generate a typed DataSet whose schema is based on the results of invoking the **\_GetAccount** stored procedure.

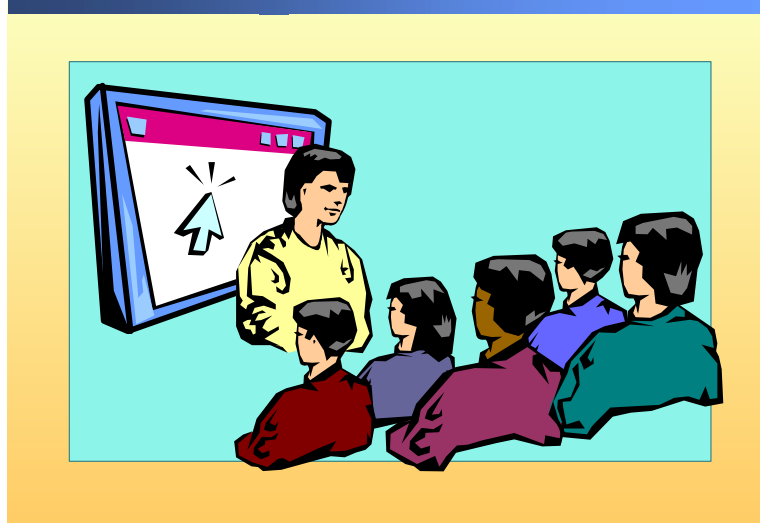
To generate a typed DataSet:

1. Right-click the **sqlDataAdapter1** control and click **Generate DataSet** on the shortcut menu.
2. In the **Generate Dataset** dialog box, select the following:
  - a. To create a new dataset, ensure that **New** is selected.
  - b. Select the default name **DataSet1** and type **AccountDataSet**.
  - c. Leave the other dialog box elements at the default settings and click **OK** to generate the **AccountDataSet** typed DataSet.
3. Open the **Class** view of the project, and notice that the **AccountDataSet** typed DataSet has been added to the project.

To view the generated DataSet:

1. In Solution Explorer, click **Show All Files**.  
You should see the file AccountDataSet (.cs or .vb) listed under AccountDataSet.xsd, in Solution Explorer.
2. Double-click AccountDataSet (.cs or .vb).  
You will see the generated DataSet.

## Demonstration: Creating a Typed DataSet Using the XML Designer



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this demonstration, you will learn how to create a typed DataSet by using the XML Designer.

To add an XML schema to the project:

1. On the **Project** menu, click **Add New Item**.
2. In the **Add New Item** dialog box, double-click the **XML Schema** icon.  
The XML Designer appears. The name of the schema file that is created is `XSDSchema1.xsd`. Click **OK**.

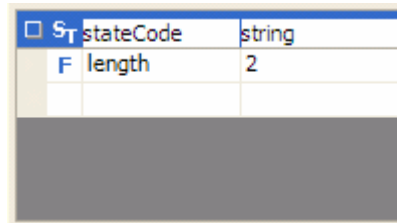
To add an XML simpleType element to the schema:

1. If the schema is not already open, double-click `XSDSchema1.xsd` to open the XML Designer.
2. In the **Toolbox**, on the **XML Schema** tab, drag a **simpleType** object onto the design surface.
3. Click the first text box in the header type `stateCode` to change the name of the simple type.
4. In the list in the second cell of the header, click **string** to set the base type for the `stateCode` type.
5. Click in the box directly below the letters **ST**. A down arrow appears. Click the down arrow to display a list. Select **facet**.
6. In the first cell of the first row (below `stateCode`), select **length** from the drop-down list.

7. In the second cell of the same row, type **2**

This information specifies that the value entered into the **stateCode** field must be two characters.

In the **Schema** view, the **stateCode** type that you create should look like the following illustration.



□ S	stateCode	string
F	length	2

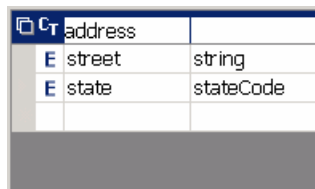
8. Click the **XML** tab to see the XML code that has been added:

```
<xsd:simpleType name="stateCode">
  <xsd:restriction base="xsd:string">
    <xsd:length value="2" />
  </xsd:restriction>
</xsd:simpleType>
```

To add an XML complexType element to the schema:

1. Click the **Schema** tab of the XML Designer.
2. On the **XML Schema** tab, on the **Toolbox**, drag a **complexType** object onto the design surface.
3. Select **complexType1** and type **address** to name the type.
4. Click the first cell of the first row, and in the list, click **element** to add an XML element to the **address** type.
5. In the second column of the first row, change the name to **street**.
6. In the third column of the first row, in the list, click **string**.
7. Click the first cell of the second row, and in the list, click **element** to add a second XML element to the **address** type.
8. In the second column of the second row, change the name to **state**.
9. In the third column of the second row, in the list, click **stateCode**.

In the **Schema** view, the **address** type that you created should resemble the following:



□ C	address	
E	street	string
E	state	stateCode

- Click the **XML** tab to see the XML code that has been added to your .xsd file.

You will see the following code:

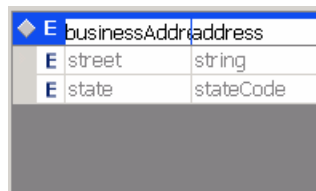
```
<xsd:complexType name="address">
  <xsd:sequence>
    <xsd:element name="street" type="xsd:string" />
    <xsd:element name="state" type="stateCode" />
  </xsd:sequence>
</xsd:complexType>
```

When you drag an **element** object from the **Toolbox** to the design surface, you are actually adding an element containing an unnamed complexType. Because the element contains an unnamed complexType, it is treated as a relational table. You can add additional elements under the complexType to define the relational fields (or columns).

To add an XML element to the project:

- Click the **Toolbox**, and on the **XML Schema** tab, drag an **element** object onto the design surface.
- In the element header, select **element1** and type **businessAddress** to name the element.
- In the second column of the element header, in the list, click **address** to set the data type.

In the **Schema** view, the **businessAddress** element should resemble the following illustration.



To generate the typed DataSet class:

- In Solution Explorer, right-click **XSDSchema1.xsd** and click **Properties**.
- In Solution Explorer, double-click **XSDSchema1.xsd**.
- Change the **id** property to **CustomerAddress**.
- Right-click the **Schema** view and select **Generate Dataset**.

To view the generated DataSet:

- In Solution Explorer, click **Show All Files**.

You should see the file **XSDSchema1.cs** listed under **XSDSchema1.xsd**, in Solution Explorer.

- Double-click **XSDSchema1 (.cs or .vb)**.

You will see the generated DataSet.

## Managing State in an ASP.NET XML Web Service

- Application State
- Session State

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

You have already seen that XML Web services are stateless. Therefore, as an XML Web service developer, if an XML Web service consumer needs to interact with an XML Web service by invoking a sequence of related operations of the XML Web service, you must consider where and how to store state.

It is generally not a good idea to have an XML Web service maintain state. However, if there is an overriding requirement that forces you to do so, then there are few options for maintaining state in an XML Web service.

Because ASP.NET-based XML Web services are ASP.NET applications, you can use ASP.NET **Application** and **Session** state objects to maintain state in your XML Web services. You could also use a custom state management solution. Irrespective of the solution that you choose, you must also consider how it would affect scalability.

## Application State

- ASP.NET support for application state
- Using application state
- Application state collections
- Application state synchronization

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

You can share global information throughout your XML Web service by using the **HttpApplicationState** class. This class exposes a key-value dictionary of objects that you can use to store both .NET Framework objects and scalar values that are related to multiple Web requests from multiple clients.

An instance of the **HttpApplicationState** class is created the first time a client requests a URL resource from within a specific ASP.NET application's virtual directory namespace. Access to this per-application instance is provided through an **HttpContext** property named **Application**. All HTTP modules and handlers (this includes ASP.NET-based XML Web Services) have access to an instance of the HTTP context during a given Web request.

### ASP.NET support for application state

ASP.NET provides the following application-state support:

- A state facility that is compatible with earlier versions of ASP, works with all .NET-supported languages, and is consistent with other .NET Framework application programming interfaces (APIs).
- A state dictionary that is available to all request handlers invoked within an application.
- A simple and intuitive synchronization mechanism that enables developers to easily coordinate concurrent access to global variables that are stored in the application state.



**Using application state**

Application-state variables are global variables for a given ASP.NET application. Just like client-side application developers, ASP.NET programmers should always consider the impact of storing any data globally.

Because of the following considerations, the use of application state is generally discouraged in an XML Web service that needs to scale. The following issues are specifically important in the context of XML Web services:

- The memory that is occupied by variables stored in application state is not released until the value is either removed or replaced.

Storing rarely-used, large DataSets in application state is not an efficient use of system resources.

- Multiple threads within an application can access values stored in an application state simultaneously. Therefore, you should always carefully ensure that access to read/write data that is stored in application state is always serialized.

---

**Note** For performance reasons, the built-in collections in the .NET Framework do not contain synchronization support. You must explicitly use the **Lock** and **Unlock** methods that the **HttpApplicationState** class provides to avoid problems when you place data in an application state.

---

- Because locks that protect global resources are themselves global, code running on multiple threads and accessing global resources could experience lock contention. Lock contention causes the operating system to block the worker threads until the lock becomes available.

In high-load server environments, this blocking can cause severe thrashing in the thread scheduler.

- The .NET application domain or a process hosting a .NET application can be shut down and destroyed at any moment during application execution as a result of failures, code updates, scheduled process restarts, and so on.

Because global data stored in application state is not durable, you can possibly lose the data if the AppDomain host is shut down. If you want to prevent loss of state information due to these types of failures, then you should store it in a database or some other durable store.

- Application state is not shared across a *Web farm* (in which an application is hosted by multiple servers) or a *Web garden* (in which an application is hosted by multiple processes on the same server).

Variables stored in application state in either of these scenarios are global only to the particular process in which the application is running. Each application process can have different values. Therefore, you cannot rely on application state to store unique values or update global counters in scenarios that use Web farms or Web gardens.

---

**Tip** Application scaling means the ability of an application to respond to increased user load with a constant (or near constant) cost in resources per-user. Increasing scalability does not mean improving performance. In fact, often steps taken to improve scalability may decrease the perceived performance for individual users.

---

**Application state collections**

The **HttpApplicationState** class exposes two state collections: **Contents** and **StaticObjects**.

The **Contents** collection exposes all variable items that have been added to the application-state collection directly through code, as shown in the following example.

**C#**

---

```
Application["Message"] = "Bar";  
Application["AppStartTime"] = DateTime.Now;
```

**Visual Basic .NET**

---

```
Application("Message") = "Bar"  
Application("AppStartTime") = DateTime.Now
```

For compatibility with earlier versions of ASP, you can also access these variables through an actual **Contents** property of the **Application** object, as in the following example.

**C#**

---

```
Application.Contents("Message") = "Bar"  
Application.Contents("AppStartTime") = DateTime.Now
```

**Visual Basic .NET**

---

```
Application.Contents("Message") = "Bar"  
Application.Contents("AppStartTime") = DateTime.Now
```

The **StaticObjects** collection exposes all objects that are defined in `Global.asax`, and that have been added to the application state collection through `<Object Runat=Server>` tags.

```
' Global.asax definition.  
<Object Runat=Server Scope=Application Id="Myinfo"↵  
    Progid="Mswc.Myinfo">  
</Object>
```

If you attempt to add objects directly through code, the **StaticObjects** collection throws a **NotSupportedException** exception.

**Application state synchronization**

Multiple threads within an application can simultaneously access values that are stored in an application state. The **HttpApplicationState** class provides two methods, **Lock** and **Unlock**, which serialize access to application state variables.

The following example code shows the use of locking to guard against race conditions.

**C#**

---

```
// Protecting shared application state via locking
Application.Lock();
Application["SomeGlobalCounter"] =
    ➤(int)Application["SomeGlobalCounter"] + 1;
Application.Unlock();
```

**Visual Basic .NET**

---

```
' Protecting shared application state via locking
Application.Lock()
Application("SomeGlobalCounter") =
    ➤CInt(Application("SomeGlobalCounter")) + 1 '
Application.Unlock()
```

If you do not explicitly call the **Unlock** method, the .NET Framework automatically removes the lock when either the request completes or times out, or when an unhandled error occurs during request execution and causes the request to fail. This automatic unlocking prevents the application from deadlocking.

## Session State

- ASP.NET support for session state
- Identifying a session
- Using session state
- Session state collections
- Session state configuration
- Cookieless sessions
- An example of retrieving data from session state

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

HTTP is a stateless protocol, and therefore it does not automatically indicate whether a sequence of requests are all from the same client, or even whether a single browser instance is continuing to actively view a page or a site. Most developers, however, are used to a stateful-programming model where objects maintain state for the lifetime of the object, not for the duration of a method call as shown in the following code examples.

### C# and Visual Basic .NET code examples

#### C#

---

```
BankAccount b;  
b = new BankAccount();  
int balance;  
balance = b.GetBalance(); // balance = n  
b.Deposit(50);  
balance = b.GetBalance(); // balance = n + 50 ???
```

#### Visual Basic .NET

---

```
Dim b As BankAccount  
b = New BankAccount()  
Dim balance As Integer  
balance = b.GetBalance() ' balance = n  
b.Deposit(50)  
balance = b.GetBalance() ' balance = n + 50 ???
```

Most developers would not expect that in the preceding code, the balance that is retrieved the second time would be unrelated to the balance that was initially retrieved. Neither would they expect that the **Deposit** method invocation would have no effect on the subsequent balance retrieval.

Building XML Web services that must maintain some cross-request state information can be extremely difficult without assistance from tools.

**ASP.NET support for session state**

ASP.NET provides the following support for session state:

- A session-state facility that is easy to use, familiar to ASP developers, and consistent with other .NET Framework APIs.
- A reliable session-state facility that can survive IIS restarts and worker-process restarts without losing session data.
- A scalable session-state facility that administrators can use in both Web farm and Web garden scenarios, and enables administrators to allocate more processors to a Web application to improve its scalability.
- A session-state facility that works with browsers that do not support HTTP cookies.

**Identifying a session**

Each active ASP.NET session is identified and tracked by using a 120-bit **SessionID** string containing URL-legal ASCII characters. **SessionID** values are generated by using an algorithm that guarantees uniqueness so that sessions do not collide, and randomness so that a malicious user cannot use a new **SessionID** to calculate the **SessionID** of an existing session.

**SessionIDs** are communicated across client-server requests either by means of an HTTP cookie or a modified URL, depending on how you set the application's configuration settings.

**Using session state**

ASP.NET session state is an in-memory cache of object references that live within the IIS process. However, the .NET state server stores chunks of binary data, either in memory or in a SQL Server database. ASP.NET worker processes are then able to take advantage of this simple storage service by serializing and saving (using .NET serialization services) all objects within a client's **Session** collection at the end of each Web request. When the client revisits the server, the relevant ASP.NET worker process retrieves these objects from the state server as binary streams, de-serializes them into live instances, and places them back into a new **Session** collection object exposed to the request handler.

An ASP.NET application can also choose to store session-state in a SQL Server database. ASP.NET worker processes then store serialized session data in a temporary table, which an ASP.NET worker process accesses by a combination of stored procedures in the database and the managed data access components for SQL Server.

By cleanly separating the storage of session data from its use, ASP.NET supports several powerful scenarios that were unavailable with earlier versions of ASP. These include:

- Recovery from application failure, because the memory that session state uses is not within the ASP.NET worker process.

This means state is not lost if the process crashes due to an access violation, or is forcibly restarted by the IIS Admin Service in the event of a deadlock or a memory leakage.

- Because all state is stored separately from running user code, it is not lost during the regular preventive restarts of each worker process after a specified interval.

ASP.NET performs preventive restarts every 20 minutes or 5000 requests to help prevent problems resulting from memory leakages, handle leakages, cache irregularities, and so on. This automatic purging process can dramatically improve the perceived availability and reliability of an application.

- Because all state is stored separately from worker processes, you can cleanly partition an application across multiple processes.

Such partitioning can dramatically improve both the availability and the scalability of an application on multi-processor computers. Moreover, because it associates each worker process with a single computer, ASP.NET is able to eliminate cross-processor lock contention, which is one of the major scalability bottlenecks in the earlier versions of ASP.

- Because all state is stored separately from worker processes, you can partition an application across multiple worker processes running on multiple computers.

The model for communicating state between a worker process and a state service running on different computers is almost the same as that for processes and servers running on the same computer.

#### Session state collections

The **SessionState** class exposes two state collections: **Contents** and **StaticObjects**. The syntax for using these two collections is very similar to the syntax for their counterparts in the **HttpApplicationState** class.

#### Session state configuration

The session state module is inserted in the HTTP request. By default it is inserted at the root of the configuration hierarchy in the Machine.config file.

```
<httpmodules>
...
  <add name="Session" type =
    ↪ "System.Web.SessionState.SessionStateModule,
    ↪ System.Web" />
...
</httpmodules>
```

Run-time parameters for the session state service are set as attributes of the **sessionState** element.

#### Cookieless sessions

Using session state no longer requires that the client support cookies. You can enable cookieless sessions by setting the following attribute of **sessionState**:

```
<configuration>
  <system.web>
    <sessionState cookieless="true"/>
  </system.web>
</configuration>
```

**C# and Visual Basic  
.NET code examples**

The following example shows how to access existing session-state data in a read-only manner. In this example, a customer's current account balances are stored in session state.

**C#**

---

```
<%@ WebService Language="C#" EnableSessionState="true" %>
.....
[WebMethod]
public DataSet GetCurrentBalances(string acctID)
{
    DataSet balances;
    balances = (DataSet) Session["CurrentBalances"];
    if (balances == null)
    {
        // retrieve current balances from database
        balances = GetCurrentBalancesLive(acctID);
        Session["CurrentBalances"] = balances;
    }
    return balances;
}
```

**Visual Basic .NET**

---

```
<%@ WebService Language="vb" EnableSessionState="true" %>
.....
Public<WebMethod()> _
Function GetCurrentBalances(acctID As String) As DataSet
    Dim balances As DataSet
    balances = CType(Session("CurrentBalances"), DataSet)
    If balances Is Nothing Then
        ' retrieve current balances from database
        balances = GetCurrentBalancesLive(acctID)
        Session("CurrentBalances") = balances
    End If
    Return balances
End Function 'GetCurrentBalances
```

## Debugging XML Web Services

- Debug, Trace, and Switch Classes
- Configuring Debug and Trace Settings
- SOAP Extensions and Tracing
- Demonstration: Performing Trace Using a SoapExtension
- Tools for Debugging Web Applications

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In any complex application, the capability to debug the application is crucial, and tracing remains one of the most valuable developer debugging tools. In this section, you will learn how to use tracing to debug XML Web services. First, you will learn how to use the **Trace**, **Debug**, and **Switch** classes in debugging XML Web services. Next, you will see an example of how to perform tracing at the SOAP level by using a SOAP extension. Finally, you will look at other tools that, although not specific to XML Web services, are useful in debugging XML Web services.



## Debug, Trace, and Switch classes

### ■ Debug

- The Debug class is usually used to display messages in a debug output window

### ■ Trace

- The Trace class is used to write trace messages to some trace output destination

### ■ Switches

- Control whether or not tracing occurs

### ■ Listeners

- Collect trace output

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

In programming, instrumentation usually refers to the ability of an application to incorporate the use of debugging, code tracing, performance counters, and event logs. Like any other applications, you can instrument XML Web services to facilitate debugging and performance monitoring.

### Debug and trace

The **Systems.Diagnostics** namespace includes the **Trace** and **Debug** classes. These two classes (which are essentially identical) include a number of static methods that you can use to gather information about code-execution paths, code coverage, and even performance profiling. The difference between these classes is that the **Debug** class only produces output with debug builds, while the **Trace** class produces output with debug and release builds of applications.

You usually use the **Debug** class during development to display messages in a debug output window.

The **Trace** class is usually used in a deployed application to write trace messages to a trace output destination.

When using the **Debug** class during testing, you can place debugging statements almost anywhere in an application under development. When using the **Trace** class, you must take more care about where you write trace output statements because trace statements are present and execute even in nondebug code.

### Switches

If you want to control whether tracing occurs, how extensive the tracing is, and where the trace output is written, you can place switches in your code. Placing switches in your code lets you monitor the health of your application based on its behavior in a production environment. This is especially important in a business application that uses multiple components running on multiple computers. You can control how the switches are used after deployment by updating values in the application configuration file.

## Trace switches

A trace switch is an object, which is an instance of the **Switch** class. One type of **Switch** class is the **BooleanSwitch** class, which acts as a toggle switch, by either enabling or disabling a variety of trace statements. Another type of switch class is the **TraceSwitch** class, which allows you to enable a trace switch for a particular tracing level so that the trace messages specified for that level and all levels below it will be output. If you disable the switch, the trace messages will not be output.

Typically, a deployed application is executed with its switches disabled, so that the users do not find irrelevant trace messages appearing on a screen or filling up a log file as the application runs. If a problem arises during program execution, you can stop the application, enable the switches, and restart the application. Then the tracing messages will be displayed.

## The TraceSwitch Class

A **TraceSwitch** object has four properties that return Boolean values indicating whether the switch is set to at least a particular level:

- **TraceError**
- **TraceWarning**
- **TraceInfo**
- **TraceVerbose**

---

**Note** The **TraceSwitch** properties indicate the maximum trace level for the switch. That is, tracing information is written for the level specified and for all the lower levels under the specified level.

---

The preceding properties correspond to the values 1 through 4 of the **TraceLevel** enumeration.

## Listeners

Objects, called listeners, collect trace output. All the tracing and debugging output methods send output to the listeners and this output is contained in the **Listeners** collection. Tracing information is always written at least to the default Trace output target, the **DefaultTraceListener**.

The six **Debug** and **Trace** output methods that write tracing information are listed in the following table. The output from these methods is sent to all defined listeners.

Method	Output
<b>Assert</b>	The specified text; or, if none is specified, the call stack
<b>Fail</b>	The specified text, written to the default Trace output
<b>Write</b>	The specified text
<b>WriteIf</b>	The specified text, if the condition specified as an argument in the <b>WriteIf</b> method is satisfied
<b>WriteLine</b>	The specified text and a carriage return
<b>WriteLineIf</b>	The specified text and a carriage return, if the condition specified as an argument in the <b>WriteLineIf</b> methods is satisfied

There are three types of predefined listeners:

- **TextWriterTraceListener**

A **TextWriterTraceListener** redirects output to an instance of the **TextWriter** class or to anything that is a **Stream** class. It can also write to a console or to a file, because these are **Stream** classes.

- **EventLogTraceListener**

An **EventLogTraceListener** redirects output to an event log.

- **DefaultTraceListener**

A **DefaultTraceListener** redirects output to the Output window in Visual Studio .NET, or calls **DebugOutputString**, a method in Microsoft Win32® APIs. This is the default behavior, because **DefaultTraceListener** is automatically included in every **Listeners** collection and is the only listener that is automatically included. If for some reason you have deleted the **DefaultTraceListener** without specifying any other listener, you will not receive any tracing messages.

**Debug** and **Trace** share the same **Listeners** collection, so if you add a listener object to a **Debug.Listeners** collection in your application, it is added to the **Trace.Listeners** collection also. As a result, any listener in the **Listeners** collection receives the same message from the trace output methods.

The following code shows how to add two listeners, one to send debug output to the console, and the other to send debug output to a text file.

#### C#

---

```
Trace.Listeners.Clear();  
Trace.Listeners.Add(new TextWriterTraceListener(Console.Out));  
Trace.Listeners.Add(new  
    ↪TextWriterTraceListener(File.Create("output.txt")));
```

#### Visual Basic .NET

---

```
Trace.Listeners.Clear()  
Trace.Listeners.Add(New TextWriterTraceListener(Console.Out))  
Trace.Listeners.Add(New  
    ↪TextWriterTraceListener(File.Create("output.txt")))
```

## Configuring Debug and Trace Settings

- Interactive debugging
- Tracing
- Configuring a TraceSwitch
- Configuring listeners

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

One of the most common ways to configure trace and debugging settings for an XML Web service is by using the Web.config file.

### Interactive debugging

The following code illustrates how you can specify that the ASP.NET runtime must compile ASP.NET pages (in the context of this course, .asmx files) to retail or debug binaries. If you specify **debug="false"**, you will not be able to use the Visual Studio Debugger to step into the implementation of an XML Web service method from an XML Web service consumer.

```
<configuration>
  <system.web>
    ...
    <compilation defaultLanguage="c#" debug="true" />
    ...
  </system.web>
</configuration>
```

---

**Note** It is recommended that you enable the **debug** attribute only when you are debugging an application, because it significantly affects the performance of the application.

---

You can also specify the **debug** attribute in the Machine.config file that is located in the directory for the .NET common language run-time directory. This setting will then affect all of the applications that are running on your computer.

**Tracing**

You can enable or disable tracing in your XML Web service by using the **trace** element as follows:

```
<configuration>
  <system.web>
    ....
    <trace
      enabled="true"
      requestLimit="10"
      pageOutput="false"
      traceMode="SortByTime"
      localOnly="true"
    />
    ....
  </system.web>
</configuration>
```

**Configuring a TraceSwitch**

When an application executes the code that creates an instance of a switch for the first time, it checks the configuration system for trace-level information about the named switch. The tracing system examines the configuration system only once for any particular switch, which happens the first time that your application creates the switch.

In a deployed application, you can control your tracing code by reconfiguring switch objects between the runs of your application. Typically this involves turning the switch objects on and off or by changing the tracing levels, and then restarting your application.

---

**Note** When you create an instance of a switch, you also initialize it by specifying two arguments: a **displayName** argument and a description argument. All switch management techniques identify switches by their display names.

---

To set the level of your switch, edit the configuration file that corresponds to the name of your application. In the following code, entries for two switches have been added.

```
<configuration>
...
  <system.diagnostics>
...
    <switches>
      <add name="acctInfo" value="0" />
      <add name="acctUpdates" value="4" />
    </switches>
...
  </system.diagnostics>
...
</configuration>
```

---

**Note** To improve performance, you can make Switch members static (Shared) in your class.

---

**C# and Visual Basic  
.NET code examples**

The following code shows how to use switches within an application:

**C#**


---

```
TraceSwitch tsInfo = new TraceSwitch("acctInfo","Info
↳traces");
TraceSwitch tsUpdate = new TraceSwitch("acctUpdate","Update
↳traces");

Trace.WriteLineIf(tsInfo.TraceVerbose,"Writing all acctInfo
↳traces..");
Trace.WriteLineIf(tsUpdate.TraceVerbose,"Writing all
↳acctUpdate traces..");
...
tsInfo.Level = TraceLevel.Warning;
Trace.WriteLineIf(tsInfo.TraceWarning,"some warning...");
...
tsInfo.Level = TraceLevel.Error;
Trace.WriteLineIf(tsInfo.TraceWarning,"another warning...");
```

**Visual Basic .NET**


---

```
Dim tsInfo As New TraceSwitch("acctInfo", "Info traces")
Dim tsUpdate As New TraceSwitch("acctUpdate", "Update traces")
'

Trace.WriteLineIf(tsInfo.TraceVerbose, "Writing all acctInfo
↳traces..") '
Trace.WriteLineIf(tsUpdate.TraceVerbose, "Writing all
↳acctUpdate traces..")
...
tsInfo.Level = TraceLevel.Warning
Trace.WriteLineIf(tsInfo.TraceWarning, "some warning...")
...
tsInfo.Level = TraceLevel.Error
Trace.WriteLineIf(tsInfo.TraceWarning, "another warning...")
```

The level of the **tsInfo** switch is initialized to 0 (Off) and the level of the **tsUpdate** switch is initialized to 4 (Verbose). In the latter part of the preceding code, the level of the **tsInfo** switch is programmatically modified.

**Configuring listeners**

To set the level of a listener, edit the configuration file that corresponds to the name of the application. For XML Web services, this is Web.config. Within this file, you can add a listener, set its type and parameters, remove a listener, or delete all the listeners that the application previously added. The following XML is an example of how to configure listeners:

```
<configuration>
  <system.diagnostics>
    <listeners>
      <add name="myTextListener"↳
        type="TextWriterTraceListener"↳
        parameter="c:\myListeners.log" />
    </listeners>
  </system.diagnostics>
</configuration>
```

## SOAP Extensions and Tracing

- SOAP extensions
- Tracing using SOAP extensions
  1. Implement a class derived from **SoapExtension**
  2. Implement a custom attribute
  3. Apply the custom attribute to a Web Service method

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

ASP.NET XML Web services provide an extensibility mechanism for calling XML Web services by using SOAP. The extensibility mechanism revolves around an extension that is allowed to inspect or modify a message at specific stages in message processing on either the client or the server.

### SOAP extensions

ASP.NET SOAP extensions derive from the **SoapExtension** class. The **ProcessMessage** method is the most important part of SOAP extensions because it is called at each stage that is defined in the **SoapMessageStage** enumeration. For more detailed information about SOAP extensions, see Module 7, “Securing XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

ASP.NET XML Web services enable applying SOAP extensions to an XML Web service method by applying an attribute. When a custom extension attribute is added to an XML Web service method or a proxy class client, ASP.NET Web services invoke the associated extension at the appropriate time. An extension attribute is a custom attribute class deriving from the **SoapExtensionAttribute** class.

### Tracing using SOAP extensions

The following code example implements a SOAP extension named **TraceExtension**. The example also implements a custom attribute named **TraceExtensionAttribute** so that you can apply **TraceExtension** to an XML Web service method or an XML Web service proxy class method.

1. First, you implement a class that is derived from **SoapExtension**.

**C#**

---

```
public class TraceExtension : SoapExtension {
    ....
    public override void Initialize(object initializer) {
        filename = (string) initializer;
    }

    public override void ProcessMessage(SoapMessage message)
    {
        switch (message.Stage) {
            case SoapMessageStage.BeforeSerialize:
                break;
            case SoapMessageStage.AfterSerialize:
                WriteOutput( message );
                break;
            case SoapMessageStage.BeforeDeserialize:
                WriteInput( message );
                break;
            case SoapMessageStage.AfterDeserialize:
                break;
            default:
                throw new Exception("invalid stage");
        }
    }
    ....
}
```

**Visual Basic .NET**

---

```
Public Class TraceExtension
    Inherits SoapExtension
    ....
    Public Overrides Sub Initialize(initializer As Object)
        filename = CStr(initializer)
    End Sub 'Initialize

    Public Overrides Sub ProcessMessage(message As
SoapMessage)
        Select Case message.Stage
            Case SoapMessageStage.BeforeSerialize
            Case SoapMessageStage.AfterSerialize
                WriteOutput(message)
            Case SoapMessageStage.BeforeDeserialize
                WriteInput(message)
            Case SoapMessageStage.AfterDeserialize
            Case Else
                Throw New Exception("invalid stage")
            End Select
        End Sub 'ProcessMessage
    ....
End Class 'TraceExtension
```



2. Next, implement a custom attribute that is derived from **SoapExtensionAttribute**. Notice that the **ExtensionType** property returns the type of your SOAP extension class.

C#

---

```
[AttributeUsage(AttributeTargets.Method)]
public class TraceExtensionAttribute :
    SoapExtensionAttribute {
    private string filename = "c:\\log.txt";
    private int priority;

    public override Type ExtensionType {
        get { return typeof(TraceExtension); }
    }

    public override int Priority {
        get { return priority; }
        set { priority = value; }
    }

    public string Filename {
        get {return filename; }
        set { filename = value; }
    }
}
```

**Visual Basic .NET**

---

```
<AttributeUsage(AttributeTargets.Method)> _
Public Class TraceExtensionAttribute
    Inherits SoapExtensionAttribute

    Private filename As String = "c:\log.txt"
    Private priority As Integer

    Public Overrides ReadOnly Property ExtensionType() As
Type
        Get
            Return GetType(TraceExtension)
        End Get
    End Property

    Public Overrides Property Priority() As Integer
        Get
            Return priority
        End Get
        Set
            priority = value
        End Set
    End Property

    Public Property Filename() As String
        Get
            Return filename
        End Get
        Set
            filename = value
        End Set
    End Property
End Class 'TraceExtensionAttribute
```

3. Finally, apply the custom attribute to an XML Web service method.

**C#**

---

```
[WebMethod]
[TraceExtensionAttribute(Filename=@"c:\log.txt")]
public DataSet GetTransactionHistory(
    DateTime startDate,DateTime endDate)
{
    ...
}
```

**Visual Basic .NET**

---

```
Public<WebMethod(), TraceExtensionAttribute(Filename :=
    ↵"c:\log.txt")> _
Function GetTransactionHistory(startDate As DateTime,
    endDate As DateTime) As DataSet
```

The following is an example of the resulting output. Note that you can see the complete SOAP request and response:

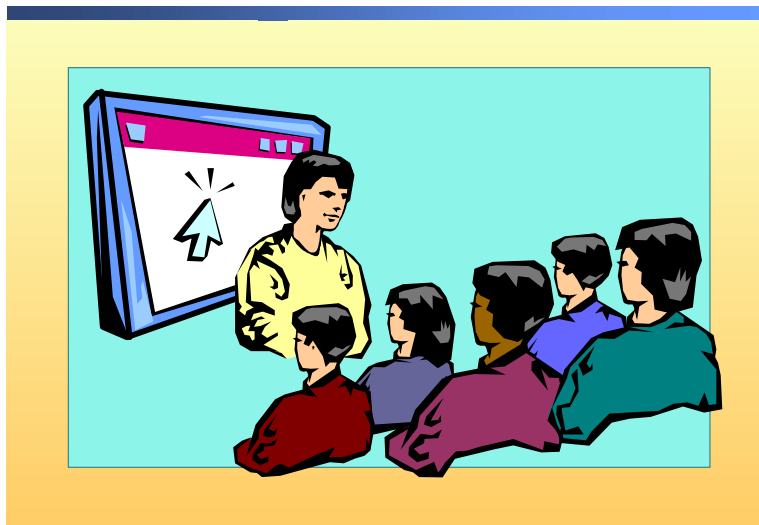
```

===== Request at 7/24/2001 11:53:49 AM
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope namespaces omitted for brevity >
  <soap:Body>
    <GetTransactionHistory xmlns="http://tempuri.org/">
      <startDate>2001-01-01T00:00:00.0000000-08:00</startDate>
      <endDate>2002-01-01T00:00:00.0000000-08:00</endDate>
    </GetTransactionHistory>
  </soap:Body>
</soap:Envelope>

----- Response at 7/24/2001 11:53:49 AM
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope namespaces omitted for brevity >
  <soap:Body>
    <GetTransactionHistoryResponse xmlns="http://tempuri.org/">
      <GetTransactionHistoryResult>
        <xsd:schema id="NewDataSet" ...>
          <xsd:element name="NewDataSet" msdata:IsDataSet="true">
            <xsd:complexType>
              omitted for brevity
            </xsd:complexType>
          </xsd:element>
        </xsd:schema>
        <diffgr:diffgram ...>
          <NewDataSet xmlns="">
            <AnAccount diffgr:id="AnAccount1" msdata:rowOrder="0">
              <TransactionID>1</TransactionID>
              <TransactionDate>2001-07-19T00:00:00.0000000-07:00</TransactionDate>
              <CustomerID>1</CustomerID>
              <Amount>12</Amount>
              <Description>No details available</Description>
            </AnAccount>
          </NewDataSet>
        </diffgr:diffgram>
      </GetTransactionHistoryResult>
    </GetTransactionHistoryResponse>
  </soap:Body>
</soap:Envelope>

```

## Demonstration: Performing Tracing Using a SoapExtension



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this demonstration, you will view how to perform tracing in an XML Web service by using SOAP extensions.

## Tools for Debugging Web Applications

- Page-level and application-level tracing
- Writing to an event log
- Performance counters

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Visual Studio .NET and the .NET Framework provide many useful tools that assist in debugging and monitoring Web applications. This topic provides information about some of these tools. For a comprehensive overview of other tools, see the Visual Studio .NET and .NET Framework SDK documentation.

### Page-level tracing

When implementing XML Web services, it is important to be able to trace all of the requests that are sent to the XML Web service including the contents of each request. It is also important to be able to remotely access this trace information.

### Application-level tracing

The configuration file settings, which you saw earlier when learning about controlling trace, included an element named trace. The trace element supports five attributes.

Attribute	Description
<b>Enabled</b>	Specifies whether tracing is enabled for an application. The value of true indicates that the tracing is enabled.
<b>requestLimit</b>	Specifies the number of trace requests to store on the server. The default is 10.
<b>pageOutput</b>	Specifies whether trace output is rendered at the end of each page. The value true indicates that trace output is appended to each page. The value false indicates that trace output is accessible through the trace utility only. The default is false.
<b>traceMode</b>	Indicates how to display trace information. SortByTime indicates that trace information is displayed in the order in which it is processed. The default is SortByTime. SortByCategory indicates that trace information is displayed alphabetically by user-defined categories.

These settings affect the results that are displayed when you use the Trace.axd utility. Trace.axd is a utility that can be used to retrieve application-level tracing information. If you type the following URL in a browser address box, tracing has been enabled, and the **localOnly** attribute is set to false:

`http://servername/applicationname/trace.axd`

The details that you see resemble the following:

No.	Time of Request	File	Status Code	Verb	Remaining:
1	7/24/2001 12:27:23 PM	/PaymentService.asmx	200	POST	<a href="#">View Details</a>
2	7/24/2001 1:23:01 PM	/PaymentService.asmx	200	POST	<a href="#">View Details</a>
3	7/24/2001 1:23:07 PM	/PaymentService.asmx	200	POST	<a href="#">View Details</a>
4	7/24/2001 1:23:07 PM	/PaymentService.asmx	200	POST	<a href="#">View Details</a>
5	7/24/2001 2:28:21 PM	/PaymentService.asmx	200	POST	<a href="#">View Details</a>
6	7/24/2001 2:28:21 PM	/PaymentService.asmx	200	POST	<a href="#">View Details</a>
7	7/24/2001 2:28:22 PM	/PaymentService.asmx	200	POST	<a href="#">View Details</a>
8	7/24/2001 2:28:32 PM	/PaymentService.asmx	200	POST	<a href="#">View Details</a>
9	7/24/2001 2:28:32 PM	/PaymentService.asmx	200	POST	<a href="#">View Details</a>
10	7/24/2001 2:28:33 PM	/PaymentService.asmx	200	POST	<a href="#">View Details</a>

If you click one of the **View Details** links, the details that you see resemble the following:

Request Details	
Session Id:	
Time of Request:	7/24/2001 2:28:32 PM
Request Encoding:	Unicode (UTF-8)
Request Type:	POST
Status Code:	200
Response Encoding:	Unicode (UTF-8)

Control Tree	
Control Id	Type
	Render Size Bytes (including children)
	Viewstate Size Bytes (excluding children)

Headers Collection	
Name	Value
Connection	Keep-Alive
Content-Length	363
Content-Type	text/xml; charset=utf-8
Expect	100-continue
Host	localhost
User-Agent	Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client Protocol 1.0.2914.16)
SOAPAction	"http://tempuri.org/GetAccountInfo"

Server Variables	
Name	Value
ALL_HTTP	HTTP_CONNECTION:Keep-Alive HTTP_CONTENT_LENGTH:363 HTTP_CONTENT_TYPE:text/xml; charset=utf-8 HTTP_EXPECT:100-continue HTTP_HOST:localhost HTTP_USER_AGENT:Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client Protocol 1.0.2914.16) HTTP_SOAPACTION:"http://tempuri.org/GetAccountInfo"
ALL_RAW	Connection: Keep-Alive Content-Length: 363 Content-Type: text/xml; charset=utf-8 Expect: 100-continue Host: localhost User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services

## Writing to an Event Log

An important tool for monitoring an application is the Microsoft Windows® event log. The .NET Framework makes it very easy to write information to event logs. The following code illustrates how to write a warning to a custom event log named ContosoLog.

---

**C#**

---

```
string LogName = "ContosoLog";
string AppName = "ContosoMicropayments";

EventLog Log = new EventLog(LogName);
string Message = string.Format("Invalid account number
↳{0}", acctNumber);
Log.Source = AppName;
Log.WriteEntry(Message, EventLogEntryType.Warning);
```

---

**Visual Basic .NET**

---

```
Dim LogName As String = "ContosoLog"
Dim AppName As String = "ContosoMicropayments"

Dim Log As New EventLog(LogName)
Dim Message As String = String.Format("Invalid account number
↳{0}", acctNumber)
Log.Source = AppName
Log.WriteEntry(Message, EventLogEntryType.Warning)
```

It is not a good idea to make frequent calls to the event log because of the performance implications, but it is recommended that you use the event log if you want to log exceptions.

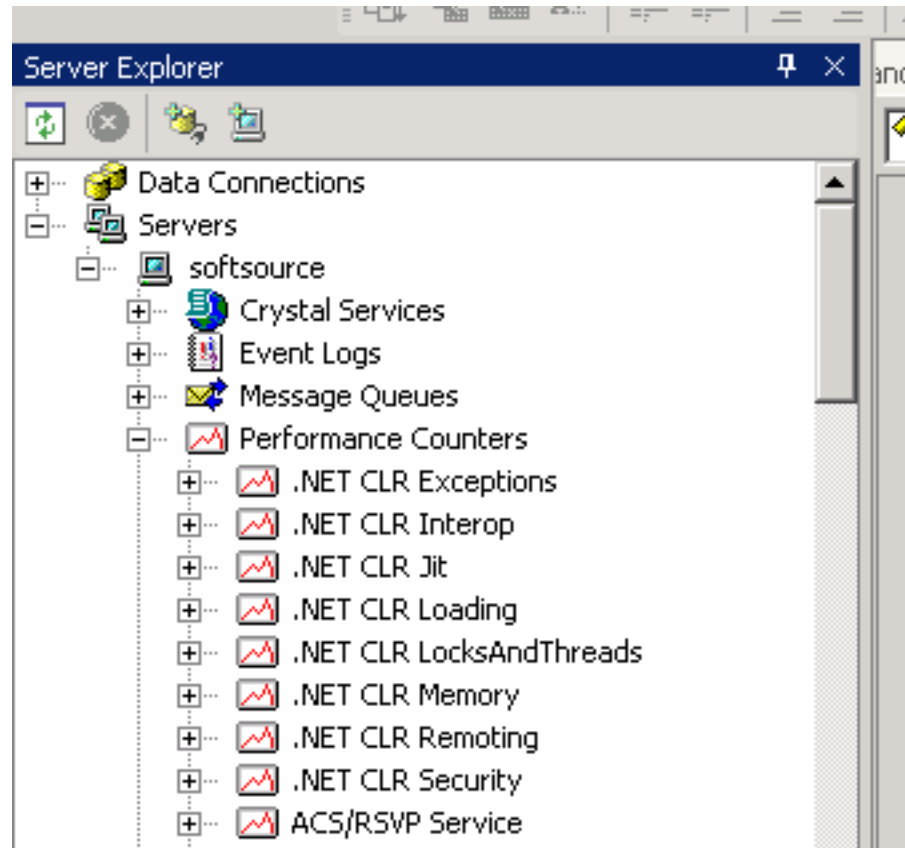
**Performance counters**

Performance counters provide an excellent way to monitor the performance of your XML Web service at run time. The .NET Framework makes it easy to access performance counters in addition to creating custom counters specifically for XML Web services.

You will now look at how to create a new category and custom performance counter at design time.

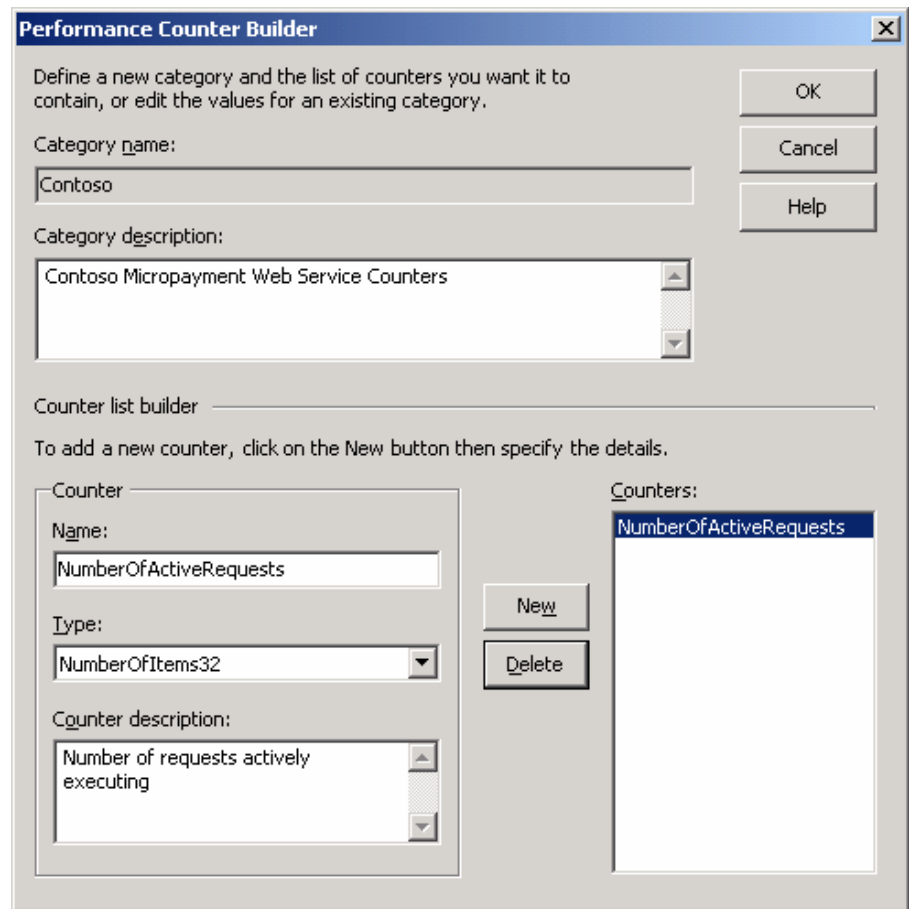
To create custom performance counters:

1. Open Server Explorer and expand the node for the server that you want to view. The contents on your screen should resemble the following.





2. Right-click the **Performance Counters** node and click **Create New Category**.



The Performance Counter Builder dialog box is shown. It has a title bar with a close button. The main area contains instructions: "Define a new category and the list of counters you want it to contain, or edit the values for an existing category." Below this are fields for "Category name:" (containing "Contoso") and "Category description:" (containing "Contoso Micropayment Web Service Counters"). To the right are "OK", "Cancel", and "Help" buttons. Below these is a section titled "Counter list builder" with instructions: "To add a new counter, click on the New button then specify the details." This section contains a "Counter" sub-dialog with fields for "Name:" (containing "NumberOfActiveRequests"), "Type:" (a dropdown menu showing "NumberOfItems32"), and "Counter description:" (containing "Number of requests actively executing"). To the right of this sub-dialog are "New" and "Delete" buttons. Further right is a "Counters:" list box containing "NumberOfActiveRequests".

3. Type the required information.

---

**Tip** Before you exit the dialog box, you can select any of the counters in the Counters list and edit their values, or delete the counters.

---

The counters and categories that you create in the dialog box are read-write enabled by default, but your interaction with them through an instance of the PerformanceCounter component will be restricted to read-only unless you specify otherwise.

The following code sample shows how you can use the performance counter that you created earlier:

1. Create an instance of the **PerformanceCounter** class to give users access to the performance counter. You do this in Global.asax. You can store the object reference in the **Application** object for later use.

**C#**

---

```
protected void Application_Start(Object sender, EventArgs
e)
{
    PerformanceCounter perf = new PerformanceCounter(
        "Contoso", "NumberOfActiveRequests",
        "Contoso Web Service", false);
    Application.Add("perf", perf);
}
```

**Visual Basic .NET**

---

```
Protected Sub Application_Start(sender As Object, e As
EventArgs)
    Dim perf As New PerformanceCounter( _
        "Contoso", "NumberOfActiveRequests", _
        "Contoso Web Service", False)
    Application.Add("perf", perf)
End Sub 'Application_Start
```

2. In a Web method, you can retrieve the object reference and manipulate the performance counter as shown in the following code.

**C#**

---

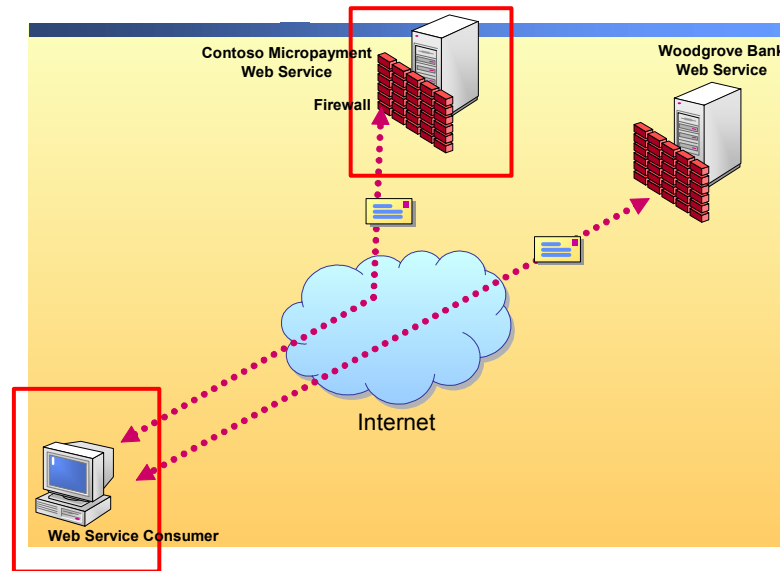
```
[WebMethod]
public DataSet GetTransactionHistory(DateTime
↵startDate, DateTime endDate)
{
    PerformanceCounter perf;
    perf = (PerformanceCounter)Application["perf"];
    perf.IncrementBy(1);
    ...
}
```

**Visual Basic .NET**

---

```
Public<WebMethod()> _
Function GetTransactionHistory(startDate As DateTime,
endDate ↵As DateTime) As DataSet '
    Dim perf As PerformanceCounter
    perf = CType(Application("perf"), PerformanceCounter)
    perf.IncrementBy(1)
    ...
End Function 'GetTransactionHistory
```

## Lab 5.1: Implementing a Simple XML Web Service



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Objectives

After completing this lab, you will be able to:

- Implement a simple XML Web service by using ASP.NET.
- Use typed DataSet from within an XML Web service.
- Add tracing statements and performance counters to an XML Web service.
- Make use of **Application** state to store information.

**Note** This lab focuses on the concepts in this module and as a result may not comply with Microsoft security recommendations. For instance, this lab does not comply with the recommendation that you not specify hardcoded values for authentication information. Also, the lab has you save passwords for database access in clear text when generating typed datasets.

### Lab Setup

There are starter and solution files that are associated with this lab. The starter files are in the folder <labroot>\Lab05\Starter. The solution files for this lab are in the folder <labroot>\Lab05\Solution.

### Scenario

In this lab, you will create the initial version of the Contoso Micropayment Service. This is an XML Web service that manages accounts for users who do not want to provide their personal financial information to all of the e-commerce sites that they shop at.

In this lab, you will also extend the Woodgrove and Contoso Account Manager client to use the Contoso Micropayment Service. You will modify the client application to use the following Contoso XML Web service methods:

- **GetAccount**
- **GetTransactionHistory**

Estimated time to  
complete this lab: 75  
minutes

## Exercise 1

### Creating the Contoso Micropayment Web Service

In this exercise, you will implement the skeleton code for the Contoso Micropayment XML Web service by using Visual Studio .NET. You will also configure an application setting for a SQL Server connection string in the Web.config file for the XML Web service.

#### ► Create the Contoso XML Web service

1. Open Visual Studio .NET.
2. On the **File** menu, point to **New**, and click **Project**.
3. Select the language of your choice, then select the **ASP.NET Web Service** project template. Set the project location to `http://localhost/Contoso`. Click **OK** to begin.
4. Rename the default implementation file.
  - a. In Solution Explorer, right-click **Service1.aspx**.
  - b. On the shortcut menu, click **Rename**.
  - c. In the textbox, type **Micropayment.aspx**.
5. Open the Micropayment.aspx code behind file:
  - a. In Solution Explorer, right-click **Micropayment.aspx**.
  - b. On the shortcut menu, click **View Code**.
6. Rename the **Service1** class to **MicroPaymentService**.
7. For C# projects only, rename the constructor in the class to **MicroPaymentService**.
8. Add a **using** (for C#) or **Imports** (for Visual Basic) statement for each of the following namespaces:
  - `System.Web.Services.Protocols`
  - `System.Data`
  - `System.Data.SqlClient`
  - `System.Configuration`
9. Add a **WebService** attribute to the **MicroPaymentService** class. Set the following attribute properties. Visual Basic .NET programmers will only need to modify the attribute that was already added.

Properties	Values
Name	Contoso Micropayment Service
Description	Contoso Micropayment Services

► **Add the skeleton code for the XML Web service methods**

- Add the stubs for the **GetAccount** and **GetTransactionHistory** methods in the Contoso Micropayment XML Web service. The stubs are as follows.

C#

---

```
[WebMethod]
public AccountDataSet GetAccount()
{
}

[WebMethod]
public TransactionDataSet GetTransactionHistory(DateTime
↪startDate, DateTime endDate)
{
}
```

Visual Basic .NET

---

```
<WebMethod()> _
Public Function GetAccount() As AccountDataSet
End Function 'GetAccount

<WebMethod()> _
Public Function GetTransactionHistory(startDate As
↪DateTime, endDate As DateTime) As TransactionDataSet
End Function 'GetTransactionHistory
```

At this point, you can ignore the undefined types **AccountDataSet** and **TransactionDataSet**. You will define these types in Exercise 2.

► **Use temporary authentication information**

1. Add the following class definition, before the **MicroPaymentService** class.

C#

---

```
public class ContosoAuthInfo : SoapHeader
{
    public string Username;
    public string Password;
}
```

Visual Basic .NET

---

```
Public Class ContosoAuthInfo
    Inherits SoapHeader
    Public Username As String
    Public Password As String
End Class 'ContosoAuthInfo
```

2. Add a public data member called **authInfo** of type **ContosoAuthInfo** to the **MicroPaymentService** class.

3. In the constructor of the **MicroPaymentService** class, test the value of the **authInfo** field. If the value is null, then:
  - a. Create an instance of **ContosoAuthInfo** and set the **authInfo** data member to this instance.
  - b. Set the **Username** and **Password** fields to **John** and **password** respectively. This user name and password corresponds to a preconfigured database user.

---

**Note** In real-world applications, it is recommended that you not specify hardcoded values for authentication information.

---

► **Add configuration information for the SQL connect string**

1. Open Web.config.
2. Create an **appSettings** element directly following the **configuration** tag as follows:

```
<appSettings>
  <add key="connectStringContoso" value="data
  ↪source=.\MOC;
  ↪initial catalog=Contoso;user id=sa;pwd=Course_2524"
/>
</appSettings>
```

## Exercise 2

### Accessing the Database

In this exercise, you will add functionality to the **GetAccount** and **GetTransactionHistory** methods to call the existing stored procedures in the SQL Server Contoso database that return account data and transactions, respectively.

#### ► Add a new database connection to Server Explorer

1. Open Server Explorer.
2. Right-click **Data Connections** and click **Add Connection...**
3. Complete the **Data Link Properties** by using the information in the following table.

On this wizard page	Do this
<b>Connection tab of the Data Link Properties dialog box</b>	<p>For the numbered fields, type the following values:</p> <ol style="list-style-type: none"><li>1. <i>The name of your computer</i>\MOC</li><li>2. User name: <b>sa</b> Password: <b>Course_2524</b> Select the <b>Allow saving password</b> check box.</li><li>3. <b>Contoso</b></li></ol> <p>To verify that the connection information is correct, click <b>Test Connection</b>.</p>
<b>Microsoft Datalink</b>	Click <b>OK</b> .

4. A dialog will be displayed warning that your connection information is not encrypted. Click **OK**. In general, this is not a safe practice, but in the classroom it is convenient.

#### ► Add an AccountDataSet typed dataset

1. In Solution Explorer, right-click the **Contoso** project and click **Add** and then **Add New Item** on the shortcut menu.
2. From the list of available templates, click **DataSet**.
3. In the **Name** field, rename the file to **AccountDataSet.xsd**.

#### ► Generate a typed AccountDataSet

1. Expand the Stored Procedures node under the newly added connection in the Server Explorer.
2. Click the **\_GetAccount** stored procedure and drag it to the designer surface for **AccountDataSet.xsd**.

### ► Complete the **GetAccount** method

1. Open the code behind file for **Micropayment.asmx**.
2. Locate the **GetAccount** method.
3. Add and instantiate local variables for a **SqlCommand**, **SqlConnection** and **SqlDataAdapter**.
4. Initialize the **SqlCommand** object as shown in the following example.

C#

```
cmd.CommandText = "_GetAccount";
cmd.CommandType = CommandType.StoredProcedure;
cmd.Connection = conn;
cmd.Parameters.Add(new SqlParameter("@RETURN_VALUE", SqlDbType.Int, 4,
ParameterDirection.ReturnValue, true, ((System.Byte)(10)), ((System.Byte)(0)), "",
DataRowVersion.Current, null));
cmd.Parameters.Add(new SqlParameter("@userID", SqlDbType.NVarChar, 16,
ParameterDirection.Input, true, ((System.Byte)(10)), ((System.Byte)(0)), "",
DataRowVersion.Current, authInfo.Username));
cmd.Parameters.Add(new SqlParameter("@password", SqlDbType.NVarChar, 16,
ParameterDirection.Input, true, ((System.Byte)(10)), ((System.Byte)(0)), "",
DataRowVersion.Current, authInfo.Password));
```

Visual Basic .NET

With cmd

```
.CommandText = "_GetAccount"
.CommandType = CommandType.StoredProcedure
.Connection = conn
.Parameters.Add(New SqlParameter("@RETURN_VALUE", SqlDbType.Int, 4,
↳ParameterDirection.ReturnValue, True, CType(10,Byte), CType(0,Byte), "",
↳DataRowVersion.Current, Nothing))
.Parameters.Add(New SqlParameter("@userID", SqlDbType.NVarChar, 16,
↳ParameterDirection.Input, True, CType(0,Byte), CType(0,Byte), "",
↳DataRowVersion.Current, authInfo.Username))
.Parameters.Add(New SqlParameter("@password", SqlDbType.NVarChar, 16,
↳ParameterDirection.Input, True, CType(0,Byte), CType(0,Byte), "",
↳DataRowVersion.Current, authInfo.Password))
End With
```

5. Initialize the **SqlConnection** object as shown in the following example.

C#

```
conn.ConnectionString =
(string)ConfigurationSettings.AppSettings["connectStringContoso"];
conn.Open();
```

Visual Basic .NET

```
conn.ConnectionString = ConfigurationSettings.AppSettings("connectStringContoso")
conn.Open()
```

6. Initialize the **SqlDataAdapter** as shown in the following example.

C#

```
adapter.SelectCommand = cmd;
```

Visual Basic .NET

```
adapter.SelectCommand = cmd
```



7. Complete the **GetAccount** method by creating an instance of the **AccountDataSet** typed dataset, using the **SqlDataAdapter** object to fill it and then return the dataset from the method. The following code is an example of how to do this.

**C#**

---

```
AccountDataSet ds = new AccountDataSet();
int nRecords = adapter.Fill(ds, "_GetAccount");
conn.Close();
if (nRecords == 0)
    throw new Exception("Account not found");
return ds;
```

**Visual Basic .NET**

---

```
Dim ds As New AccountDataSet()
Dim nRecords As Integer = adapter.Fill(ds, "_GetAccount")
conn.Close()
If nRecords = 0 Then
    Throw New Exception("Account not found")
End If
Return ds
```

► **Add a TransactionDataSet typed dataset**

1. In Solution Explorer, right-click the **Contoso** project and click **Add** and then **Add New Item** on the shortcut menu.
2. From the list of available templates, click **DataSet**.
3. In the **Name** field, rename the file to **TransactionDataSet.xsd**.

► **Generate a typed TransactionDataSet**

1. Expand the Stored Procedures node under the newly added connection in the Server Explorer.
2. Click the **\_GetTransactionLog** stored procedure and drag it to the designer surface for **TransactionDataSet.xsd**.

► **Complete the GetTransactionHistory method**

1. Open the code behind file for Micropayment.asmx.
2. Locate the **GetTransactionHistory** method.
3. Add and instantiate local variables for a **SqlCommand**, **SqlConnection**, and **SqlDataAdapter**.

4. Initialize the **SqlCommand** object as shown in the following example.

C#

---

```
cmd.CommandText = "_GetTransactionLog";
cmd.CommandType = CommandType.StoredProcedure;
cmd.Connection = conn;
cmd.Parameters.Add(new SqlParameter("@RETURN_VALUE", SqlDbType.Int, 4,
ParameterDirection.ReturnValue, true, ((System.Byte)(10)), ((System.Byte)(0)), "",
DataRowVersion.Current, null));
cmd.Parameters.Add(new SqlParameter("@userID", SqlDbType.NVarChar, 16,
ParameterDirection.Input, true, ((System.Byte)(10)), ((System.Byte)(0)), "",
DataRowVersion.Current, authInfo.Username));
cmd.Parameters.Add(new SqlParameter("@password", SqlDbType.NVarChar, 16,
ParameterDirection.Input, true, ((System.Byte)(10)), ((System.Byte)(0)), "",
DataRowVersion.Current, authInfo.Password)); cmd.Parameters.Add(new
SqlParameter("@startDate", SqlDbType.DateTime, 8, ParameterDirection.Input, true,
((System.Byte)(0)), ((System.Byte)(0)), "", DataRowVersion.Current, startDate));
cmd.Parameters.Add(new SqlParameter("@endDate", SqlDbType.DateTime, 8,
ParameterDirection.Input, true, ((System.Byte)(0)), ((System.Byte)(0)), "",
DataRowVersion.Current, endDate));
```

Visual Basic .NET

---

With cmd

```
.CommandText = "_GetTransactionLog"
.CommandType = CommandType.StoredProcedure
.Connection = conn
.Parameters.Add(New SqlParameter("@RETURN_VALUE", SqlDbType.Int, 4,
➔ParameterDirection.ReturnValue, True, CType(10,Byte), CType(0,Byte), "",
➔DataRowVersion.Current, Nothing))
.Parameters.Add(New SqlParameter("@userID", SqlDbType.NVarChar, 16,
➔ParameterDirection.Input, True, CType(0,Byte), CType(0,Byte), "",
➔DataRowVersion.Current, authInfo.Username))
.Parameters.Add(New SqlParameter("@password", SqlDbType.NVarChar, 16,
➔ParameterDirection.Input, True, CType(0,Byte), CType(0,Byte), "",
➔DataRowVersion.Current, authInfo.Password))
.Parameters.Add(New SqlParameter("@startDate", SqlDbType.DateTime, 8,
➔ParameterDirection.Input, True, CType(0,Byte), CType(0,Byte), "",
➔System.Data.DataRowVersion.Current, startDate))
.Parameters.Add(New SqlParameter("@endDate", SqlDbType.DateTime, 8,
➔ParameterDirection.Input, True, CType(0,Byte), CType(0,Byte), "",
➔System.Data.DataRowVersion.Current, endDate))
```

End With

5. Initialize the **SqlConnection** object as shown in the following example.

C#

---

```
conn.ConnectionString =
(string)ConfigurationSettings.AppSettings["connectStringContoso"];
conn.Open();
```

Visual Basic .NET

---

```
conn.ConnectionString = ConfigurationSettings.AppSettings("connectStringContoso")
➔conn.Open()
```

6. Initialize the **SqlAdapter** as shown in the following example.

**C#****Visual Basic .NET**

---

```
adapter.SelectCommand = cmd; adapter.SelectCommand = cmd
```

7. Complete the **GetTransactionHistory** method by creating an instance of the **TransactionDataSet** typed dataset, using the **SqlDataAdapter** object to fill it and then return the dataset from the method. The following code is an example of how to do this.

**C#**

---

```
TransactionDataSet ds = new TransactionDataSet();  
int nRecords = adapter.Fill(ds, "_GetTransactionLog");  
conn.Close();  
return ds;
```

**Visual Basic .NET**

---

```
Dim ds As New TransactionDataSet()  
Dim nRecords As Integer = adapter.Fill(ds,  
    ↪ "_GetTransactionLog")  
conn.Close()  
Return ds
```

### ► Test the Contoso XML Web service

1. Build the Contoso XML Web service.
2. To invoke the help test page, press F5. Verify that you see the **GetAccount** and **GetTransactionHistory** methods.
3. Click the GetAccount link. Click **Invoke**. An XML document should be returned.

## Exercise 3

# Using the Contoso XML Web Service in the Woodgrove and Contoso Account Manager

In this exercise, you will add a Contoso Web reference to the client application for the Woodgrove and Contoso Account Manager, and add code to invoke the proxy class methods for the Contoso Micropayment Service.

### ► Add a Web reference

1. Open the Woodgrove and Contoso Account Manager project that you worked on in Lab 4.1, Implementing an XML Web Service Consumer Using Visual Studio .NET, in Module 4, “Consuming XML Web Services” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

---

**Note** If you did not start or complete Lab 4.1, use the Woodgrove and Contoso Account Manager project found in the `<labroot>\Lab05\Starter\Woodgrove and Contoso Account Manager` folder. Otherwise, continue to use the Woodgrove and Contoso Account Manager project that you worked on in Lab 4.1 (in the `<labroot>\Lab04\Starter\Woodgrove and Contoso Account Manager` folder).

---

2. In Solution Explorer, right-click the **References** node and click **Add Web Reference**.
3. In the **Add Web Reference** dialog box, enter the URL `http://localhost/Contoso/Micropayment.asmx` in the address field and press ENTER.  
You should see the Service Help Page for the Contoso XML Web service.
4. At this point in the discovery process, the **Add Reference** button becomes enabled. Click this button to complete adding a Web reference.
5. Expand the **Web References** node in the project tree in Solution Explorer.
6. Rename the localhost namespace to **Micropayment**. To do this, right-click **localhost** and then click **Rename**. Type **Micropayment**.

► **Retrieve Contoso account information**

1. In WebServiceClientForm(.cs or .vb), add a **using** (for C#) or **Imports** (for Visual Basic) statement to reference the namespace **WebServiceClient.Micropayment**.
2. In WebServiceClientForm form, set the **Enabled** property of the **buttonContosoGetAccount** to **true**.
3. To retrieve the transactions:
  - a. Locate the **GetContosoAccountInfo** method.
  - b. Create an instance of the **ContosoMicropaymentService** proxy class.
  - c. Call the **GetAccount** method of the proxy class. Save the returned **AccountDataSet** typed DataSet.
  - d. Call the **DataSetToXMLString** helper method to obtain a string representation of the **AccountDataSet**.
  - e. Assign the string that is returned from **DataSetToXMLString** to the **Text** property of the **textBoxContosoAccount** text box.
4. In the **Form1 Load** event handler, call the **GetContosoAccountInfo** method.

► **Retrieve Contoso transactions**

1. In the WebServiceClientForm form, set the **Enabled** property of the **buttonContosoGetTransactions** to **true**.
2. Locate the **buttonContosoGetTransactions\_Click** method.
3. Create an instance of the **ContosoMicropaymentService** proxy class.
4. Call the **GetTransactionHistory** method with the **dtStart** variable as the *startDate* parameter and **DateTime.Now** as the *endDate* parameter.

The **GetTransactionHistory** method also returns a typed DataSet, also named **TransactionDataSet**.

To distinguish Contoso **TransactionDataSet** from the Woodgrove **TransactionDataSet**, prefix the Contoso **TransactionDataSet** with the **WebServiceClient.Micropayment** namespace.

5. Check the number of transactions. If there are more than 0 transactions, then:
  - a. Call the **DataSetToXMLString** helper method to obtain a string representation of the **TransactionDataSet**.
  - b. Assign the string that is returned from **TransactionDataSet** to **frm.Transactions**.
6. If there are no records then display the message "No transactions for this user within the past week. ".

► **To test the modified client application**

1. Build and run the application.  
Notice that the Contoso account information is displayed.
2. Click an account number in the **Accounts for Customer** list.
3. Click **Get Transaction History** for the Woodgrove XML Web service.  
Existing transactions are displayed.
4. Click **Get Transaction History** for the Contoso XML Web service.  
If there are no recent transactions, a message that indicates that there have been no recent transactions is displayed. Otherwise, a form that lists the recent transactions is displayed.

## Exercise 4

### Debugging the Contoso Micropayment Service

In this exercise, you write code to output debug statements and add performance counters to an XML Web service.

#### ► Write debug information using the Trace class

1. Open the Contoso Micropayment XML Web Service project with Visual Studio .NET.
2. Open the code behind file for Micropayment.asmx and locate the **GetTransactionHistory** method.
3. At the top of the method implementation, output a debug statement by:
  - a. Creating a string that indicates the beginning of the **GetTransactionHistory** method and displays the value of the **authInfo.Username** field.
  - b. Calling the **Context.Trace.Write** method. Pass the string as the message parameter.
4. Immediately before the **return** statement in the method, call the **Context.Trace.Warn** method. Set the message parameter to a string that indicates the record count of the returned DataSet.
5. Build the application.

#### ► Enable tracing in Web.config

1. Open Web.config and locate the **<trace>** element.
2. Set the **enabled** attribute to **true**.  
Do not change the default values for the **requestLimit**, **pageOutput**, **traceMode**, and **localOnly** attributes.
3. Save Web.config.

#### ► View trace output

1. Run the Woodgrove and Contoso Account Manager.
2. Retrieve the Contoso transaction history several times.
3. To view trace output, in Internet Explorer open <http://localhost/Contoso/Trace.axd>.  
Notice that the page displays a line for each request to the Micropayment.asmx page.
4. Click the associated **View Details** link for each of the requests.
5. At each Request Details page, locate the **SOAPAction** header listed in the **Headers Collection** table. The value associated with the **SOAPAction** specifies the XML Web service that is method invoked for this request.
6. Locate a request for which the **SOAPAction** shows the value **http://Tempuri.org/GetTransactionHistory**.

7. Locate the **Trace Information** table on this Request Detail page.

In the **Trace Information** table, you should see the 2 trace messages that you output within the **GetTransaction** method. These messages should resemble the following:

#### **GetTransactionHistory for John**

#### **GetTransactionHistory returns 0 records**

The second message should be displayed in red, because it was output by using the **Context.Trace.Warn** method.

---

**Note** Tracing impacts performance and therefore it is recommended that you use it only when debugging. Before deploying an XML Web service, remember to disable tracing by setting the **enabled** attribute of the **<trace>** element to **false** in Web.config.

---

### ► **Add performance counters**

1. Open the Contoso Micropayment XML Web Service project with Visual Studio .NET.
2. Open Server Explorer and expand the node for your server.
3. Right-click the **Performance Counters** node and click **Create New Category**.
4. In the **Performance Counter Builder** dialog box, set the following information:

Field	Value
Category name	Contoso
Category description	Contoso Micropayment Web Service

5. Click **New** to add a new performance counter to the Contoso category. Set the following information.

Field	Value
Name	NumberOfActiveRequests
Type	NumberOfItems32
Description	Number of active GetAccount requests

6. Click **OK**.
7. Open the code behind file for Global.asax.
8. C# programmers must import the **System.Diagnostics** namespace. This is unnecessary for Visual Basic .NET programmers.



9. Within the **Application\_Start** method, create an instance of the **PerformanceCounter** class. Store the object reference in the **Application** object for later use, using the key **perfCtr**. Enclose the code in a **try/catch** block as shown below.

C#

---

```
protected void Application_Start(Object sender, EventArgs
    e)
{
    try
    {
        PerformanceCounter perfCtr = new
        PerformanceCounter("Contoso",
            "NumberOfActiveRequests", "Contoso Web
        Service", false);
        Application.Add("perfCtr", perfCtr);
    }
    catch
    {
        // non critical error do nothing
    }
}
```

Visual Basic .NET

---

```
Protected Sub Application_Start(sender As [Object], e As
    EventArgs)
    Try
        Dim perf As PerformanceCounter = New
        PerformanceCounter("Contoso", _
            "NumberOfActiveRequests", "Contoso Web
        Service", False)
        Application.Add("perfCtr", perf)
    Catch
        'no critical error
    End Try
End Sub 'Application_Start
```

10. Open the code behind file for **Micropayment.asmx**.
11. Locate the **GetAccount** method. At the beginning of this method, declare a **PerformanceCounter** variable. Retrieve the **PerformanceCounter** reference from the **Application** object. Cast this reference to the **PerformanceCounter** class.
12. If **PerformanceCounter** variable created in step 11 contains a valid object reference, then call the **Increment** method on the performance counter. This will cause the performance counter to be incremented whenever this method is invoked.
13. If **PerformanceCounter** variable created in step 11 contains a valid object reference, then call the **Decrement** method on the performance counter immediately after closing the database connection. This will give a count of the number of clients that are concurrently executing the **GetAccount** method.
14. Build the application.

► **View performance counters**

1. To open the **Performance** utility, on the **Start** menu, point to **Control Panel**, click **Performance and Maintenance**, click **Administrative Tools** and then double-click **Performance**.
2. In the **tree** view, click **System Monitor**.
3. On the toolbar, click the **Delete** button until there are no more counters being monitored.
4. On the toolbar, click the **Add** button to add a performance counter to the chart.
5. In the **Add Counter** dialog box, do the following:
  - a. Click **Use local computer counters**.
  - b. In the **Performance object** list, click **Contoso**.
  - c. Click **All counters**.
  - d. Click **Add**.
  - e. Click **Close**.
6. Right-click in the graphing window and click **Properties**. Click on the **Data** tab. In the **Scale** list, click 10.0. Click **OK**.
7. To chart the **NumberOfActiveRequestsCounters** counter at a non-zero value, you must invoke the **GetAccount** method. To do this:
  - a. Within the Contoso Micropayment Service project, set a breakpoint in the **GetAccount** method.
  - b. Start the application from within Visual Studio .NET to debug.
  - c. In the browser, open the GetAccount test page and invoke the method. Click **Invoke**.
  - d. When the application stops at the breakpoint in the **GetAccount** method, step beyond the line that increments the performance counter.
8. Switch to the Performance utility.

Notice that the **NumberOfActiveRequestsCounters** now charts the value 1.
9. Within Visual Studio .NET, press F5 to continue execution.

Notice that the **NumberOfActiveRequestsCounters** again charts the value 0 because the **Decrement** method was called at the end of the **GetAccount** method.

## Review

- Creating an XML Web Service Project
- Implementing XML Web Service Methods
- Managing State in an ASP.NET XML Web Service
- Debugging XML Web Services

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

1. If you want to expose a C# method as an XML Web service operation, what must you do?

**Apply the `WebMethod` attribute to the method, and make sure that the method is public.**

2. If an XML Web service method returns a stream that is 1 megabyte in size, which property of the **`WebMethod`** attribute should you modify to minimize the amount of time a client would wait for the arrival of the first set of data?

**`BufferResponse`**

3. Which properties and fields of a class are serialized when an XML Web service method returns an instance of a class?

**All public, read/write properties and all public, read/write fields.**

4. If your XML Web service will be deployed on a Web farm, what kind of data can be stored appropriately in Application state?

**Read-only data that can be restored exactly in the event of an XML Web service being restarted.**

5. You deploy an XML Web service in production, and want to store trace information in a disk file on the server hosting the XML Web service. Which two classes could you use to do this?
- **Trace to emit trace output**
  - **TextWriterTraceListener to write the trace output to disk**
6. Which .NET Framework class is provided to allow you to hook into various stages of SOAP message processing?

**SoapExtension**

---

## Module 6: Publishing and Deploying XML Web Services

### Contents

Overview	1
Overview of UDDI	2
Publishing an XML Web Service	16
Finding an XML Web Service	21
Publishing an XML Web Service on an Intranet	24
Configuring an XML Web Service	26
Lab 6.1: Publishing and Finding Web Services in a UDDI Registry	29
Review	39



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, places or events 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, trademarks, 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.

© 2001–2002 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Active Directory, Authenticode, IntelliSense, FrontPage, Jscript, MSDN, PowerPoint, Visual C#, Visual Studio, Visual Basic, Windows NT, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

# Instructor Notes

**Presentation:**  
**90 Minutes****Lab:**  
**60 Minutes**

This module teaches students how to publish XML (Extensible Markup Language) Web services and locate XML Web services by using the Microsoft Universal Description, Discovery, and Integration (UDDI) software development kit (SDK). A local development UDDI registry is used in the demonstrations and lab for this module, but the mechanics of publishing and finding XML Web services is no different on the public UDDI registry nodes.

After completing this module, students will be able to:

- Explain the role of UDDI in XML Web services.
- Publish an XML Web service in a UDDI registry by using the Microsoft UDDI SDK.
- Search a UDDI registry to locate XML Web services by using the UDDI SDK.
- Explain the various options for publishing an XML Web service on an intranet.
- Explain some of the options for modifying the default configuration of an XML Web service.

**Required Materials**

To teach this module, you need the Microsoft® PowerPoint® file 2524B\_06.ppt.

**Preparation Tasks**

To prepare for this module:

- Read all of the materials for this module.
- Practice all of the demonstrations.
- Review the walkthrough demonstration code in *<install folder>\Democode\<language>\Mod06*.
- Complete the lab.

## Module Strategy

Use the following strategy to present this module:

- Overview of UDDI

This topic provides an overview of the role of UDDI registries in the process of XML Web service discovery. The UDDI data structures and application programming interfaces (APIs) are explained. Ensure that you cover each of the elements in the UDDI data structure and describe how they are used. Many students might have difficulty understanding the **tModel** element. Explain that the **tModel** element is deliberately generically defined because it can be used in multiple ways in UDDI. Explain that using **tModel** to represent Web Services Description Language (WSDL) documents is just one of its uses. Provide only a brief overview of publisher assertions. Explain the publisher and inquiry APIs. Emphasize that each of these APIs is an XML Web service operation that can be invoked by using Simple Object Access Protocol (SOAP). Explain how the UDDI SDK encapsulates these operations.

- Publishing an XML Web Service

Explain how to publish **tModels** and business entities. Emphasize that **tModels** are not explicitly associated with business entities. Therefore, before publishing a business service, you must publish the necessary business entities and **tModels**. Explain how to publish a binding template and also explain the importance of binding templates in locating the endpoints for XML Web services.

- Finding an XML Web Service

Explain how to locate binding information for an XML Web service by using the UDDI SDK. Emphasize that at the moment there are no standards for how developers should publish XML Web services and how developers should use the information that is published about an XML Web service to locate the XML Web service. Explain to the students that they will not be able to locate endpoints for all XML Web services programmatically.

- Publishing an XML Web Service on an Intranet

Explain the issues that are involved in deploying an XML Web service on an intranet from the perspective of publishing and finding XML Web services. This topic explains some of the options that are available for publishing an XML Web service on an intranet in the absence of a commercial private UDDI registry.

- Configuring an XML Web Service

This topic focuses on the issues specific to the assemblies that make up an XML Web service. The topic covers modifying the default discovery behavior, configuring assembly settings, and security considerations. Use this topic to introduce students to topics such as localization of assemblies, and building and deploying satellite assemblies, which are beyond the scope of this course, and yet applicable to XML Web services.



# Overview

- Overview of UDDI
- Publishing an XML Web Service
- Finding an XML Web Service
- Publishing an XML Web Service on an Intranet
- Configuring an XML Web Service

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

You can locate XML (Extensible Markup Language) Web services at unknown endpoints by using a public registry of XML Web services. Universal Description, Discovery, and Integration (UDDI) registries facilitate service discovery on the Web.

## Objectives

After completing this module, you will be able to:

- Explain the role of UDDI in XML Web services.
- Publish an XML Web service in a UDDI registry by using the Microsoft® UDDI software development kit (SDK).
- Search a UDDI registry by using the UDDI SDK to locate an XML Web service that implements a specific service interface.
- Explain the various options for publishing an XML Web service on an intranet.
- Explain some of the options for modifying the default configuration of an XML Web service.

## Overview of UDDI

- What Is UDDI?
- UDDI Data Structures
- Demonstration: Using UDDI Explorer
- UDDI Programmer's API

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

The role of an XML Web service broker is essential in XML Web services architecture to make it possible for potential consumers to easily find an XML Web service. UDDI registries fulfill the role of an XML Web service broker. In this section, you will learn about UDDI and the associated specifications.

## What Is UDDI?

### ■ A collection of specifications

- Specifications for distributed Web-based information registries of XML Web services
  - UDDI Programmer's API Specification
  - UDDI Data Structure Specification

### ■ UDDI registry implementations

- Implementations of the specifications

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Before learning how to programmatically publish and discover XML Web services, it is important to understand the functions of UDDI from a programmer's perspective.

#### A collection of specifications

Universal Description, Discovery, and Integration (UDDI) is a collection of specifications for distributed Web-based information registries of XML Web services. These specifications are broken down into a number of categories. The UDDI Programmer's API (application programming interface) Specification and the UDDI Data Structure Specification are of specific interest to an XML Web service developer. You can find the current versions of these specifications at <http://www.uddi.org>.

---

**Note** All of the information about the UDDI Programmer's API and UDDI Data Structure Specifications is based on version 2.0 of the specifications.

---

#### The UDDI Programmer's API Specification

The UDDI Programmer's API Specification defines functions that provide a simple request/response model for accessing UDDI registries. There are two types of API defined in the API reference:

- A *publisher's* API that allows you to publish data in a registry.
- An *inquiry* API that allows you to read information from a registry.

The Programmers' API Specification defines approximately 40 Simple Object Access Protocol (SOAP) messages that are used to query for information and publish functions in any UDDI-compliant service registry.

#### The UDDI Data Structure Specification

The UDDI Data Structure Specification outlines the details of each of the XML structures that are associated with the messages that the Programmer's API Specification defines.

**UDDI registry implementations**

UDDI is also a set of implementations of the specifications that allow businesses to register information about the XML Web services that they offer so that other businesses can find them. These implementations are publicly accessible. Also, UDDI registries are themselves available as XML Web services.

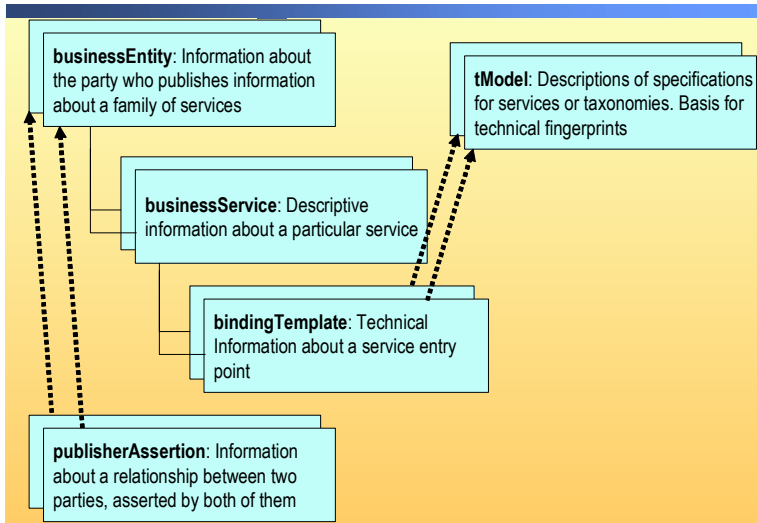
At the time of this writing, there are two public implementations of these specifications. They can be located at <http://uddi.microsoft.com> and <http://www-3.ibm.com/services/uddi>.

UDDI can provide answers to queries such as:

- What XML Web services does a specific business provide?
- What are all the known endpoints for a specific XML Web service?
- What is the current binding information (supported protocols, and so on.) for a specific XML Web service endpoint?

Other possible queries, such as price comparison of XML Web services or geographic proximity, are not part of the UDDI specification. Currently, such additional queries and associated metadata are considered value-added services that vendors are free to implement and offer.

## UDDI Data Structures



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

The UDDI Data Structure Specification defines the XML schema that is used to describe types in UDDI. The specification defines five data types:

- **businessEntity**
- **businessService**
- **bindingTemplate**
- **tModel**
- **publisherAssertion**

The preceding data structures are defined by using the XML Schema Definition Language (XSD). This topic examines each of the data structures.

### The businessEntity element

The **businessEntity** element describes a business or an entity that has registered an XML Web service within UDDI. The **businessEntity** is the top-level element that contains descriptive information about a business or an entity. This element supports standard information such as name, description, and contacts, in addition to metadata information such as identifiers and categories. The latter information is used for standard taxonomies of business identifiers (tax identifiers, D-U-N-S numbers, and so on. D-U-N-S numbers are unique nine-digit numbers of single business entities) and categories (industry codes, geography codes, and so on.) Service descriptions and technical information are related to a **businessEntity** by using containment.

The following XML code shows a document that you can use to register a **businessEntity**:

```
<businessEntity ↵
  businessKey="434554F4-6E17-1342-EA4136E642531DA0"↵
  operator="">
  <name>Contoso Micropayments</name>
  <description xml:lang="en">
    The Contoso Micropayment Service
  </description>
  <contacts>
    <contact>
      <description xml:lang="en">
        Website Administrator
      </description>
      <personName>Jeff Smith</personName>
      <phone>800-555-1212</phone>
      <email>jeff.smith@contoso.com</email>
      <address>
        <addressLine>1 Microsoft Way</addressLine>
        <addressLine>Redmond, WA</addressLine>
      </address>
    </contact>
  </contacts>
</businessEntity>
```

#### The **businessService** element

The **businessService** element describes an XML Web service that a business entity exposes. This element supports naming an XML Web service and associating it with a business entity and binding information. It also supports the assignment of categories (industry, product, geographic codes, and so on) to the XML Web service.

The following XML code shows a document that you can use to register a **businessService**:

```
<businessService ↵
  businessKey="434554F4-6E17-1342-EA41-36E642531DA0"↵
  serviceKey="AEAC8990-2891-3894-DEC1-AEF97501DD1B">
  <name>Business Service example</name>
  <description xml:lang="en">
    description goes here
  </description>
  <bindingTemplates>
    <!-- zero or more binding templates -->
    <bindingTemplate>
      elements go here...
    </bindingTemplate>
  </bindingTemplates>
</businessService>
```

#### The **bindingTemplate** element

The **bindingTemplate** element describes the technical information that is necessary for binding to a particular XML Web service. This element supports naming an XML Web service and associating it with a business entity and binding information. The binding information is described as either an access point or a hosting redirector.

**The accessPoint element**

The **accessPoint** element describes a XML Web service entry point. The **accessPoint** element has an attribute named **URLType**, which is used to specify one of the seven entry point types:

- **mailto**-the access point is an e-mail address.
- **http**-the access point is an Hypertext Transfer Protocol (HTTP) compatible URL.
- **https**-the access point is an HTTP Secure (HTTPS) compatible URL.
- **ftp**-the access point is a File Transfer Protocol (FTP) directory address URL.
- **Fax**-the access point is a telephone number that is answered by a fax machine.
- **Phone**-the access point is a telephone number that is answered by a human or voice response system.
- **Other**-the access point is any format other than the preceding ones. When this URL type is specified, the specification information (in the **tModel** element) must suggest a transport type.

**The hostRedirectory element**

Alternatively, you can use the **hostRedirectory** element in the absence of the **accessPoint** element to point to another **bindingTemplate** for specific binding information. The **hostRedirectory** element is also used to provide a mechanism to allow multiple binding templates to be associated with a single XML Web service.

The following XML code shows a document that you can use to register a **bindingTemplate**:

```
<bindingTemplate ↵
  bindingKey="FE542889-EE4B-2348-2345-AEFC3901223A"↵
  serviceKey="AEAC8990-2891-3894-DEC1-AEF97501DD1B">
    <description xml:lang="en">
      Micropayments binding template
    </description>
    <accessPoint URLType="http">
      http://www.contoso.com/micropayments/payments.asmx
    </accessPoint>
    <tModelInstanceDetails>
      <!-- ...zero or more -->
      <tModelInstanceInfo>
        elements go here
      </tModelInstanceDetails>
    </bindingTemplate>
```

**The tModelInstanceDetails element**

A **bindingTemplate** also contains a **tModelInstanceDetails** element. The **tModelInstanceDetails** element contains an unordered list of **tModelKey** references. This list of references forms a unique fingerprint. When a **bindingTemplate** is registered within a **businessEntity** structure, it will contain one or more references to a specific set of specifications. The **tModelKey** values that are provided with the registration imply the identity of these specifications.

When a **bindingTemplate** is registered, the information contained in the specifications that are referred to, can later be used during an inquiry for a service to locate a specific **bindingTemplate** that contains a particular **tModel** reference or set of **tModel** references. By listing a **tModelKey** reference in a **bindingTemplate**, an XML Web service containing this **bindingTemplate** claims to be compatible with the specifications that the **tModelKey** implies.

Within a **tModelInstanceDetails** element is a list of zero or more **tModelInstanceInfo** elements. Each **tModelInstanceInfo** has an attribute named **tModelKey**, which identifies a specific **tModel**. A **tModelInstanceInfo** element also has a description, a reference to an overview document, and instance parameters. The optional overview document contains a URL for locating the entry point specification document. The instance parameters contain either XML or a URL to an XML document that contains parameter setting information.

The following XML code shows an example of a **tModelInstanceInfo** element within a **tModelInstanceDetails**:

```
<tModelInstanceInfo ↵
  tModelKey="uuid:E31A569A-AEFF-4468-BA4D-2BF22FE4ACEE">
    <description xml:lang="en">
      Micropayments tModel
    </description>
    <instanceDetails>
      <description xml:lang="en">
        Micropayment instance details description
      </description>
      <overviewDoc>
        <description xml:lang="en">
          Micropayment Service Overview
        </description>
        <overviewURL>
          http://www.contoso.com/↵
          micropayments/overview.aspx
        </overviewURL>
      </overviewDoc>
      <instanceParms>
        http://www.contoso.com/↵
        micropayments/params.aspx
      </instanceParms>
    </instanceDetails>
  </tModelInstanceInfo>
```

### The tModel element

One of the most important goals of UDDI is to provide a facility to make XML Web service descriptions complete enough so that developers can easily learn how to interact with a service that they did not know much about. To accomplish this goal, there must be a way to attach metadata to a description of an XML Web service. You can use this metadata in a variety of ways. For example, the metadata can define how the XML Web service behaves, what conventions it follows, or what specifications or standards it is compliant with. The **tModel** element provides the ability to describe compliance with a specification, concept, or even a shared design.



The structure of a **tModel** element takes the form of metadata with associated keys. Although a **tModel** registration can define almost anything, currently there are two primary ways to use **tModel** elements:

- To determine if two XML Web services are compatible.
- To provide keyed namespace references.

The information that makes up a **tModel** is quite simple. There's a key, a name, an optional description, and a URL that indicates where you can find more information about the XML Web service. Also, because many different business entities can reference **tModels**, multiple business entities may implement the same XML Web service interface.

If a **tModel** represents a Web Services Description Language (WSDL) document, then the **categoryBag** element of the **tModel** structure should contain a **keyedReference** element. The **keyName** attribute of the **keyedReference** element should have the value **uddi-org:types**, and the **keyValue** attribute should have the value **wsdlSpec**. Also the **overviewURL** element in the **overviewDoc** element should contain the location of the WSDL document.

The following XML code shows a document that you can use to register a **tModel**:

```
<tModel tModelKey="uuid:455655B7-4C43-4f3e-BB0B-695FE2120C53">
  <name>Micropayment TModel</name>
  <description xml:lang="en">
    A TModel for the Micropayment XML Web Service
  </description>
  <overviewDoc>
    <description xml:lang="en">
      The Micropayment XML Web Service tmodel
    </description>
    <overviewURL>
      http://192.168.0.13/ContosoBank/overview.htm
    </overviewURL>
  </overviewDoc>
</tModel>
```

#### The publisherAssertion element

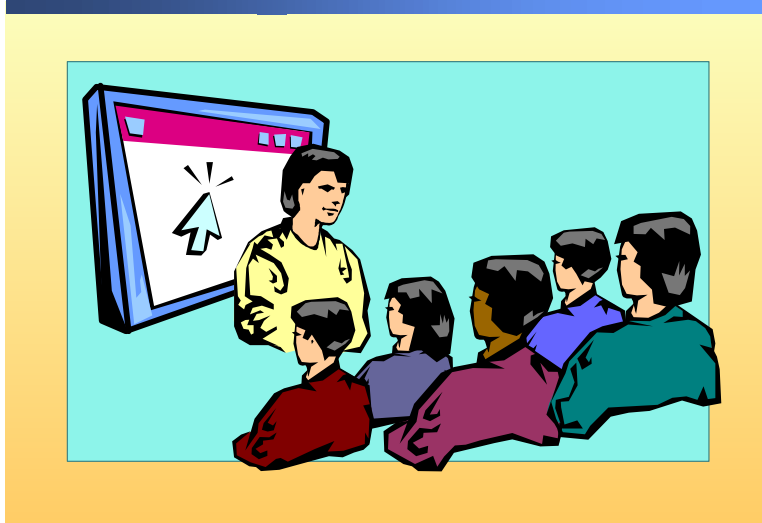
A single **businessEntity** may not effectively represent many large organizations, because there might be many ways to categorize them or the services that they offer. As a result, such organizations may publish several **businessEntity** structures. However, these business entities are still related and at least some of their relationships should be visible in their UDDI registrations. Therefore, two related businesses use the **set\_publisherAssertions** message to publish assertions of business relationships.

To eliminate the possibility that one publisher claims a relationship between two businesses that is, in fact, not reciprocally recognized, each publisher must agree that the relationship is valid by publishing their own **publisherAssertion**. Both publishers have to publish exactly the same information. When this happens, the relationship becomes visible.

In this example, the **businessEntity** with the **businessKey** value of “1234-...” is the parent holding company of the **businessEntity** with the **businessKey** value of “4567-...”.

```
<publisherAssertion>
  <fromKey>1234-....</fromKey>
  <toKey>4567-....</toKey>
  <keyedReference tModelKey="uuid:1357..." ↵
    keyName="Holding Company" keyValue="parent-child">
</publisherAssertion>
```

## Demonstration: Using UDDI Explorer



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this demonstration, you will see how you can explore and manipulate UDDI registries by using UDDI Explorer.

---

**Note** UDDI Explorer is one of the sample applications that ship with the UDDI SDK.

---

To locate information about a business by using its name:

1. Open UDDI Explorer (UDDIExplorer.exe) from the following folder  
<install folder>\Democode\<language>\Mod06.
2. In the URL list, click <http://glasgow/uddi/api/inquire.asmx>.
3. In the **Name** box, type **Contoso**.
4. Click **Search**.
5. Expand the nodes in the resulting tree.

To locate information about a tModel by using its name:

1. Select the **tModels** option.
2. In the **Name** box, type **Bank**.
3. Click **Search**.
4. Expand some of the nodes in the resulting tree.

## UDDI Programmer's API

- The Inquiry APIs
- The Publisher APIs
- The Microsoft UDDI SDK

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

The UDDI Programmer's API Specification documents the SOAP APIs that provide programmatic access to UDDI registries. These APIs are divided into two categories: Inquiry APIs and Publisher APIs.

### The Inquiry APIs

The Inquiry APIs support finding UDDI elements and retrieving detailed information about the elements. Each element in the UDDI repository has a key, which is a universally unique identifier (UUID). Inquiry methods return these keys (example: find a business by name), and the keys are also used as parameters to inquiry methods (example: finding all XML Web services exposed by a business entity with a specified businessEntityID).

You can use the Inquiry APIs to browse for UDDI data, retrieve information about specific elements, and find binding information for an XML Web service.

For browsing the UDDI repository, there are **find\_xxx** methods for each of the four main UDDI types:

- **find\_binding**
- **find\_business**
- **find\_service**
- **find\_tModel**

The following is an example of a SOAP message, which is used to search for the business entity named “Microsoft” by using the **find\_business** method:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <find_business generic="1.0" xmlns="urn:uddi-org:api">
      <name>Microsoft</name>
    </find_business>
  </Body>
</Envelope>
```

You use the **find\_relatedBusinesses** method to find business entities that have a relationship with a specific business entity.

There are **get\_xxx** methods for each of the four main UDDI types. You use these methods to retrieve more detailed information for a specific element:

- **get\_bindingDetail**
- **get\_businessDetail**
- **get\_businessDetailExt**
- **get\_serviceDetail**
- **get\_tModelDetail**

With the ability to browse for business entities and XML Web services and to retrieve specific binding information for a particular XML Web service, you can programmatically bind to any XML Web service whose endpoint is published.

There are several ways that you might use the **get\_xxx** methods to implement a rudimentary failure recovery mechanism in a situation where an XML Web service at a specific endpoint becomes unavailable. An XML Web service consumer queries a UDDI registry for binding information for an XML Web service. The consumer caches this information and uses it whenever it needs to interact with an XML Web service. If the XML Web service becomes unavailable, the consumer can search the UDDI registry for other business entities that implement the same XML Web service interface. You can then retrieve the binding information for a compatible XML Web service.

## The Publisher APIs

The Publisher APIs allow authorized access to the UDDI repository in addition to adding and deleting elements. Both authentication and authorization are required for modifying information in UDDI. To use the Publisher APIs that require authorization, you must first acquire an authorization token through the **get\_authToken** operation. You then use the token as a parameter to subsequent Publisher API calls. Finally, after you are finished using the Publisher APIs, you must discard the authorization token by invoking the **discard\_authToken** operation.

For adding or changing elements in UDDI there are **save\_**xxx methods for each of the core UDDI types:

- **save\_binding**
- **save\_business**
- **save\_service**
- **save\_tModel**

For deleting elements in UDDI, there are **delete\_**xxx methods for each of the core UDDI types:

- **delete\_binding**
- **delete\_business**
- **delete\_service**
- **delete\_tModel**

You use publisher assertions for specifying relationships between business entities. The Publisher APIs include four methods for creating and managing publisher assertions:

- **add\_publisherAssertions**
- **get\_assertionStatusReport**
- **get\_publisherAssertions**
- **delete\_publisherAssertions**

All of the APIs that this topic covers are actually operations that an XML Web service exposes that may be invoked from UDDI registries by using SOAP. However, for the application developer, it is much more convenient if the UDDI APIs can be called as if they were local functions.

#### **The Microsoft UDDI SDK**

To make using UDDI APIs easier, the UDDI SDK from Microsoft provides managed wrappers for the UDDI data structures and APIs. All of the wrappers reside within the **Microsoft.UDDI** namespace. The UDDI data structures are represented as .NET classes with many properties. Each of the UDDI API functions is implemented as a managed class with a **Send()** method.

The UDDI SDK contains the following namespaces:

Namespace	Description
<b>Microsoft.UDDI</b>	Contains classes that map to the UDDI SOAP APIs
<b>Microsoft.UDDI.Api</b>	Contains base classes and utility classes for the other namespaces
<b>Microsoft.UDDI.Authentication</b>	Contains a class to represent an authentication token in addition to classes for getting and discarding a token
<b>Microsoft.UDDI.Binding</b>	Contains classes that represent binding template elements
<b>Microsoft.UDDI.Business</b>	Contains classes that represent business entity elements
<b>Microsoft.UDDI.Service</b>	Contains classes that represent business service elements
<b>Microsoft.UDDI.ServiceType</b>	Contains classes that represent <b>tModel</b> , <b>tModelInstanceDetail</b> , and <b>tModelInstanceCollection</b> elements

## Publishing an XML Web Service

- Getting an authentication token
- Publishing a tModel
- Publishing a businessEntity
- Publishing a businessService
- Publishing a bindingTemplate

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

You must take a number of steps to successfully register an XML Web service in a UDDI registry.

To register an XML Web service in UDDI by using the UDDI APIs:

1. Obtain an authentication token by using the **get\_authToken** method.
2. Add a **tModel** by using **save\_tModel**.
3. Add a **businessEntity** by using **save\_business**.
4. Add a **businessService** by using **save\_service**.
5. Add a **bindingTemplate** by using **save\_binding**.
6. Discard the authentication token by using **discard\_authToken**.

### Getting an authentication token

All of the operations that are related to publishing an XML Web service require an authentication token. You can obtain an authentication token by invoking the **get\_authToken** operation at the UDDI registry. In the Microsoft UDDI SDK, the authentication information is provided through static members of the **Publish** class as follows:

**C#**

```
Publish.Url = strPublishUrl;
Publish.User = strUID;
Publish.Password = strPassword;
```

**Microsoft Visual Basic® .NET**

```
Publish.Url = strPublishUrl
Publish.User = strUID
Publish.Password = strPassword
```

When any publication action is taken, if an authentication token has not already been obtained, the wrapper classes automatically make the call to **get\_authToken** to obtain the authentication token.



**Publishing a tModel**

The first step in publishing an XML Web service is to publish the **tModels** that the XML Web service supports. The following code provides an example of publishing a **tModel** method.

**C#**

---

```
TModel tModel = new TModel();
tModel.Name = "Micropayment TModel";
tModel.Descriptions.Add("TModel for the Micropayment
service");
tModel.OverviewDoc.OverviewURL =
    "http://www.contoso.com/ContosoBank/overview.htm";
tModel.OverviewDoc.Descriptions.Add("The micropayment XML ↵
    Web Service tmodel");
SaveTModel saveTModel = new SaveTModel();
saveTModel.TModels.Add(tModel);
TModelDetail tmd = saveTModel.Send();
```

**Visual Basic .NET**

---

```
Dim tModel As New TModel()
tModel.Name = "Micropayment TModel"
tModel.Descriptions.Add("TModel for the Micropayment service")
tModel.OverviewDoc.OverviewURL = ↵
    "http://www.contoso.com/ContosoBank/overview.htm"
tModel.OverviewDoc.Descriptions.Add("The micropayment XML ↵
    Web Service tmodel")
Dim saveTModel As New SaveTModel()
saveTModel.TModels.Add(tModel)
Dim tmd As TModelDetail = saveTModel.Send()
```

---

**Note** You do not specify the **tModelKey** when you register a **tModel**. The **tModelKey** is returned as part of the **TModelDetail** object that is returned from the **Send** method.

---

**Publishing a businessEntity**

Before you can publish an XML Web service, you need to publish the **businessEntity** with which the service will be associated. The following code provides an example of how to publish a **businessEntity**.

**C#**

---

```
Contact contact = new Contact();
contact.PersonName = "Adam Barr";
contact.Emails.Add("Adam@contoso.com");
contact.Phones.Add("(425)555-0101");
contact.Descriptions.Add("en","Web Site Administrator");

BusinessEntity businessEntity = new BusinessEntity();
businessEntity.Name = "ContosoBank";
businessEntity.Descriptions.Add("en","The Contoso
↵Micropayment Bank");
businessEntity.Contacts.Add(contact);

SaveBusiness saveBusiness = new SaveBusiness();
saveBusiness.BusinessEntities.Add(businessEntity);
BusinessDetail bd = saveBusiness.Send();
```

**Visual Basic .NET**

---

```
Dim contact As New Contact()
contact.PersonName = "Adam Barr"
contact.Emails.Add("Adam@contoso.com")
contact.Phones.Add("(425)555-0101")
contact.Descriptions.Add("en", "Web Site Administrator")

Dim businessEntity As New BusinessEntity()
businessEntity.Name = "ContosoBank"
businessEntity.Descriptions.Add("en", "The Contoso
↳Micropayment Bank")
businessEntity.Contacts.Add(contact)

Dim saveBusiness As New SaveBusiness()
saveBusiness.BusinessEntities.Add(businessEntity)
Dim bd As BusinessDetail = saveBusiness.Send()
```

---

**Note** You do not specify the **businessKey** when you register a business entity. The **businessKey** is returned as part of the **BusinessDetail** object that is returned from the **Send** method.

---

**Publishing a  
businessService**

Publishing a **businessService** allows developers to associate a **businessEntity** with an XML Web service. It is possible to separately publish binding information, but in the following example, the binding information is added with the rest of the service information.

**C#**

---

```
BindingTemplate bindingTemplate = new BindingTemplate();
...
// add all binding information
...
BusinessService businessService = new BusinessService();
businessService.BusinessKey = businessKey;
businessService.Name = "Micropayments";
businessService.BindingTemplates.Add(bindingTemplate);

SaveService saveService = new SaveService();
saveService.AuthInfo = "udditest";
saveService.BusinessServices.Add(businessService);
saveService.BusinessServices[0].BindingTemplates.Add(
↳bindingTemplate );
ServiceDetail sd = saveService.Send();
```

---

**Visual Basic .NET**

---

```
Dim bindingTemplate As New BindingTemplate()
...
' add all binding information
...
Dim businessService As New BusinessService()
businessService.BusinessKey = businessKey
businessService.Name = "Micropayments"
businessService.BindingTemplates.Add(bindingTemplate)

Dim saveService As New SaveService()
saveService.AuthInfo = "udditest"
saveService.BusinessServices.Add(businessService)
saveService.BusinessServices(0).BindingTemplates.Add(bindingTemplate)
'

Dim sd As ServiceDetail = saveService.Send()
```

---

**Note** You do not specify the serviceKey when you register a business entity. The serviceKey is returned as part of the **ServiceDetail** object that is returned from the **Send** method.

---

**Publishing a bindingTemplate**

To associate a **tModel** and a **businessService** and provide the necessary binding information, you must publish a **bindingTemplate**. The following code provides an example of how to publish a **bindingTemplate**.

**C#**

---

```
BindingTemplate bindingTemplate = new BindingTemplate();
bindingTemplate.ServiceKey = serviceKey;
bindingTemplate.Descriptions.Add("A binding template");
AccessPoint accessPoint = new AccessPoint(URLTypeEnum.Http,
    "http://www.contoso.com/ContosoBank/PaymentService.asmx");
bindingTemplate.AccessPoint = accessPoint;

TModelInstanceInfo tModelInstanceInfo = new TModelInstanceInfo();
tModelInstanceInfo.TModelKey = tModelKey;
tModelInstanceInfo.Descriptions.Add("TModel instance info");
bindingTemplate.TModelInstanceDetail.TModelInstanceInfos.Add(tModelInstanceInfo);

SaveBinding saveBinding = new SaveBinding();
saveBinding.BindingTemplates.Add(bindingTemplate);
BindingDetail bd = saveBinding.Send();
```

**Visual Basic .NET**

---

```
Dim bindingTemplate As New BindingTemplate()
bindingTemplate.ServiceKey = serviceKey
bindingTemplate.Descriptions.Add("A binding template")
Dim accessPoint As New AccessPoint(URLTypeEnum.Http,
    "http://www.contoso.com/ContosoBank/PaymentService.asmx")
bindingTemplate.AccessPoint = accessPoint

Dim tModelInstanceInfo As New TModelInstanceInfo()
tModelInstanceInfo.TModelKey = tModelKey
tModelInstanceInfo.Descriptions.Add("TModel instance info")
bindingTemplate.TModelInstanceDetail.TModelInstanceInfos.Add(tModelInstanceInfo)

Dim saveBinding As New SaveBinding()
saveBinding.BindingTemplates.Add(bindingTemplate)
Dim bd As BindingDetail = saveBinding.Send()
```

## Finding an XML Web Service

- Locate a business
- Retrieve binding information
- Bind to the XML Web service

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

One of the most important activities for an XML Web service consumer is to locate the endpoint of an XML Web service that implements a specific interface.

To locate an XML Web service entry point in UDDI by using the UDDI APIs:

1. Locate a business by using the **find\_business** operation (example: search on name).
2. Obtain the service information for each service that the business entity supports.
3. Obtain the binding template for a service.
4. Use the binding information to access the XML Web service.

### Locate a business

To find a business that meets certain criteria, you need to invoke the **find\_business** operation. The following code shows how to find a business.

C#

```
FindBusiness fb = new FindBusiness();  
fb.Name = "Northwind EFT Portal";  
BusinessList bl = fb.Send();
```

Visual Basic .NET

```
Dim fb As New FindBusiness()  
fb.Name = "Northwind EFT Portal"  
Dim bl As BusinessList = fb.Send()
```

**Note** The search criterion is not restricted to a simple name. You could also use a list of business identifiers, category references, discovery URLs, or **tModel** references in your search criteria.

**Retrieve binding information**

After obtaining a list of businesses that satisfy your search criteria, you can navigate the list of services that the business entity supports to retrieve binding information. When you find the binding template of your choice, you can easily extract the URL for the XML Web service. The following code demonstrates how you can extract binding information.

**C#**

---

```
foreach (BusinessInfo bi in bl.BusinessInfos)
{
    foreach(ServiceInfo si in bi.ServiceInfos)
    {
        FindBinding fb = new FindBinding();
        fb.ServiceKey = si.ServiceKey;
        BindingDetail bindDetails = fb.Send();
        foreach (BindingTemplate bt in
            ↪bindDetails.BindingTemplates)
        {
            if
            ↪(bt.TModelInstanceDetail.TModelInstanceInfos[0].TModelKey ==
                tModelKey)
            {
                strURL = bt.AccessPoint.Text;
                goto found;
            }
        }
    }
}
found:
...
```

**Visual Basic .NET**

---

```
Dim bi As BusinessInfo
For Each bi In bl.BusinessInfos '
    Dim si As ServiceInfo
    For Each si In bi.ServiceInfos
        Dim fb As New FindBinding()
        fb.ServiceKey = si.ServiceKey
        Dim bindDetails As BindingDetail = fb.Send()
        Dim bt As BindingTemplate
        For Each bt In bindDetails.BindingTemplates
            If
            ↪bt.TModelInstanceDetail.TModelInstanceInfos(0).TModelKey ==
                tModelKey Then
                strURL = bt.AccessPoint.Text
                GoTo found
            End If
        Next bt
    Next si
Next bi
found:
...
```

**Bind to the XML Web service**

After you find the binding information, it is easy to set the URL property of the XML Web service proxy and invoke XML Web service methods. The following code shows how to do this.

**C#**

---

```
NorthwindEFTService eft = new NorthwindEFTService();  
eft.Url = strURL;  
XmlNode balances = eft.GetCurrentBalances("1XF99-S");
```

**Visual Basic .NET**

---

```
Dim eft As New NorthwindEFTService()  
eft.Url = strURL  
Dim balances As XmlNode = eft.GetCurrentBalances("1XF99-S")
```

## Publishing an XML Web Service on an Intranet

- Private UDDI registries
- Custom publish/discover solutions
  - Implement a UDDI registry
  - Implement a custom publish/discover mechanism
  - Hard-coded endpoints

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

It will not be uncommon for an XML Web service to be deployed behind a firewall on an intranet. In this topic, we will focus on the options that are available for publishing an XML Web service on an intranet.

### Private UDDI registries

The simplest solution is to have a private registry. Because UDDI was envisioned as a public, replicated repository that remained essentially synchronized, there was no provision made for private registry nodes. As a result, there are currently no commercial, private UDDI registries. It is possible to use the private UDDI node that is distributed with the Microsoft UDDI SDK. However, this UDDI SDK is not a commercial product and is not intended to be used for any purpose other than development.

### Custom publish/discover solutions

There are three options for implementing custom solutions:

- Implement a private UDDI registry

You could make a private implementation of a UDDI registry by using the UDDI specifications.

The advantage of implementing a private UDDI registry that is compliant with the UDDI specifications is that when commercial private registries become available your code will not have to be rewritten. Also, you will have to learn only one API—the UDDI Inquiry and Publisher APIs.

The disadvantage is that correctly implementing the full UDDI specification is not a simple task.



- Implement a custom publish/discover mechanism

You could implement a private publication and discovery mechanism that provides just the minimum required functionality. For example, you could choose to ignore the categories for classification and searching.

The advantage of developing a fully customized solution is that you do not have to wait for commercial, private UDDI nodes to become available. Also, you are not constrained by a complex specification that is intended to fulfill the requirements of almost every industry and organization.

The disadvantage is that you will implement discovery code that will not be compatible with the industry standards.

- Hard-coded endpoints

Another alternative is to hard code the XML Web service endpoints in your XML Web service consumers.

This solution has the advantage of being very simple and quick to implement.

The disadvantage is that the resulting solutions will not be able to handle changes in the location of an XML Web service.

## Configuring an XML Web Service

- **Configuring discovery**
- **Permissions and security policy**
- **Configuring assemblies**
- **Localizing an XML Web service**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

It is unlikely that the environment in which an XML Web service is developed will exactly match the environment in which it is deployed.

### **Configuring discovery**

There are a number of scenarios that will either force or encourage you to modify the default discovery mechanisms that are associated with the deployment of an XML Web service from Microsoft Visual Studio® .NET.

### **Disabling dynamic discovery**

You might want to disable the dynamic discovery document in the root directory of your Web site if you want to prevent all of the XML Web services on your site from automatically exposing themselves. To do this, you would delete, move, or rename the dynamic discovery document in the root directory of your Web server; and remove any links to it that exist in the Web server's default page.

### **Creating your own discovery document**

If you want to customize the information that is exposed about an XML Web service in its discovery document, you must write your own discovery document.

Assume that the root Web site for a Web server is located in the folder C:\Inetpub\Wwwroot. If you deploy an XML Web service in a folder that is not an immediate child of the folder C:\Inetpub\Wwwroot, then the dynamic discovery infrastructure will not be able to locate your XML Web service. In this scenario, you must write your own discovery document if you want your XML Web service to be dynamically discoverable.

If you deploy an XML Web service without using Visual Studio .NET, you must write your own discovery document.

For details on the structure of discovery documents, see Module 4, “Consuming XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*.

**Permissions and security policy**

A security policy is a configurable set of rules that the .NET common language runtime follows when determining the permissions to grant to any code. The runtime examines identifiable characteristics of the code, such as where the code originates, and what the identity of the current caller is, to determine the level of access that the code can have to resources. During execution, the runtime ensures that the code accesses only those resources that it has permission to access. These permissions determine if an XML Web service can write to an event log, if an XML Web service can access the file system, and so on.

**Assembly permissions**

An XML Web service developer is typically not responsible for configuring the security policy on a production Web server. Therefore, it is unlikely that the security policy in the development environment will match the security policy in the production environment. Because of this reason, it is a good idea for an XML Web service developer to explicitly specify which permissions are required for an XML Web service to correctly function.

By requesting permissions you let the runtime know what your code must be allowed to do. You can request permissions for an assembly by placing attributes at the assembly scope of your code. For example, you can request the file I/O permission by adding the **FileIOPermissionAttribute**. When you create an assembly, the language compiler stores the requested permissions in the assembly manifest. At load time, the runtime examines the permission requests, and applies security policy rules to determine which permissions to grant to the assembly. Permission requests can only influence the runtime to deny permissions to your code and can never influence the runtime to grant additional permissions to your code. The local administration policy always has the final control over the maximum permissions that your code is granted.

For more information about some of the security concepts relevant to XML Web services, see Module 7, “Securing XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*. However, a full coverage of the details on assembly security and deployment is beyond the scope of this course. For more information on these topics, see Course 2350A, *Securing and Deploying Microsoft .NET Assemblies*.

**Configuring assemblies**

To successfully deploy an XML Web service, you must understand how the common language runtime locates and binds to the assemblies that make up your service. By default, the runtime attempts to bind with the exact version of an assembly with which a service is built. Configuration file settings can override this default behavior.

The common language runtime performs a number of steps when attempting to locate an assembly and resolve an assembly reference. The term *probing* is often used when describing how the runtime locates assemblies. Probing is the set of heuristics that are used to locate an assembly based on its name, version, and culture.

You can configure assembly-binding behavior at different levels based on three XML-based files:

- The application configuration file.
- The publisher policy configuration file.
- The machine configuration file.

All of the preceding files follow the same syntax and provide information such as binding redirects, the location of the assemblies, and binding modes for specific assemblies. Each configuration file can contain elements that redirect the binding process.

For complete coverage of configuring assemblies, see Course 2350A, *Securing and Deploying Microsoft .NET Assemblies*.

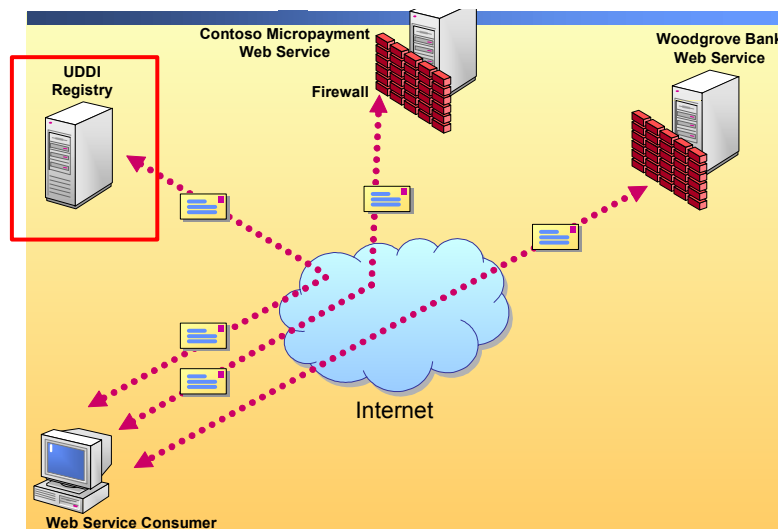
### **Localizing an XML Web service**

It is possible that you may need to localize an XML Web service. Localizing an XML Web service is typically done by separating out the culture-specific resources into separate resource-only assemblies, which are known as *satellite assemblies*. Each satellite assembly contains the resources for a single culture, but does not contain any code. There are several advantages to this model:

- You can incrementally add resources for new cultures after you have deployed an application.  
Because development of culture-specific resources can require a significant amount of time, adding resources for a new culture after application deployment allows you to deploy your XML Web service for the primary culture that you want to support, and then deliver other culture-specific resources at a later date.
- You can update and replace the satellite assemblies of an XML Web service without recompiling the XML Web service.
- At run time, an XML Web service only needs to load the satellite assemblies that contain the resources that are required for a specific culture. This can significantly reduce the use of system resources.

For more information about localizations and building and deploying satellite assemblies, see Course 2350A, *Securing and Deploying Microsoft .NET Assemblies*.

## Lab 6.1: Publishing and Finding Web Services in a UDDI Registry



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Objectives

After completing this lab, you will be able to:

- Register an XML Web service in a UDDI registry by using the Microsoft UDDI SDK.
- Find WSDL documents in a UDDI registry by using the UDDI SDK.

---

**Note** This lab focuses on the concepts in this module and as a result may not comply with Microsoft security recommendations.

---

### Lab Setup

There are starter and solution files that are associated with this lab. The starter files are in the folder <labroot>\Lab06\Starter. The solution files for this lab are in the folder <labroot>\Lab06\Solution.

### Scenario

In Lab 5.1, Implementing a Simple XML Web Service, in Module 5, “Implementing a Simple XML Web Service,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*, you implemented the Contoso XML Web service. As an XML Web service provider, you might want to publish this XML Web service. As an XML Web service consumer, you might want to find the WSDL document and the endpoint for this XML Web service.

In this lab, you will implement a simple application to register both the Contoso and Woodgrove XML Web services and implement a simple application to find these XML Web services.

Estimated time to  
complete this lab: 60  
minutes

## Exercise 1

### Implementing the RegisterWebServices Application

In this exercise, you will implement the functionality that is required to register a business entity, business service, and binding template in a UDDI registry. The starter application project is provided to you in the folder `<labroot>\Lab06\Starter\RegisterWebServices`.

► **Add references to required assemblies and import required namespaces**

1. Add a reference to the **System.Web.Services.dll** assembly.
2. Open the following file:

C#	Visual Basic .NET
Register.cs	Register.vb

3. Import the **System.Net** namespace.

► **Implement the PublishBusiness method**

1. Locate the **PublishBusiness** method.
2. Set the following the properties for the **Publish** class (which is in the **Microsoft.Uddi** assembly):
  - a. Set the **Publish.Url** property to **comboUrl.SelectedItem**. C# programmers will need to cast **comboUrl.SelectedItem** to a **string**.
  - b. Set the **Publish.AuthenticationMode** property to **AuthenticationMode.UddiAuthentication**.
  - c. Create a new instance of the class **NetworkCredential**. Pass the following values to the constructor:

Parameter	Value
Username	txtUser.Text
Password	txtPwd.Text
Domain	GLASGOW

- d. Assign the instance of **NetworkCredential** to the **Publish.HttpClient.Credentials** property.
3. Create a **BusinessEntity** object.
  4. Populate the **BusinessEntity** object.
    - a. Set the **Name** property to the value of the **name** parameter.
    - b. Add a description to the **Descriptions** collection. Use the **description** parameter as the description and specify **"en"** as the language code.
    - c. Add the **contact** parameter to the **Contacts** collection.
  5. Create a **SaveBusiness** object.
  6. Add the business entity that was created in step 3 to the **SaveBusiness.BusinessEntities** collection.

7. Call the **Send** method of the **SaveBusiness** object.
8. Return the **BusinessDetail** object that the **Send** method returns.

► **Implement the btnWoodgroveBusiness\_Click method**

1. Locate the **btnWoodgroveBusiness\_Click** method.
2. Create a **Contact** object.
3. Populate the **Contact** object.
  - a. Set the **PersonName** property to **"John Chen"**.
  - b. Add **"someone@example.com"** to the **Emails** collection.
  - c. Add a description to the **Descriptions** collection. Use the **"Web Site Administrator"** as the description and specify **"en"** as the language code.
4. Call the **PublishBusiness** method for the business entity of the XML Woodgrove XML Web Service.
  - a. Pass **"Woodgrove Online Bank"** as the *name* parameter.
  - b. Pass **"The Woodgrove Bank (Online)"** as the *description* parameter.
  - c. Pass the **Contact** object as the *contact* parameter.
5. Disable the button by setting the **Enabled** property to false.

► **Implement the btnContosoBusiness\_Click method**

1. Locate the **btnContosoBusiness\_Click** method.
2. Create a **Contact** object.
3. Set the **Contact** object properties similar to the way that you did for the **btnWoodgroveBusiness\_Click** method. Use the same values.
4. Call the **PublishBusiness** method for the business entity of the Contoso XML Web service.
  - a. Pass **"Contoso Micropayments"** as the *name* parameter.
  - b. Pass **"The Contoso Micropayments Service"** as the *description* parameter.
  - c. Pass the **Contact** object as the *contact* parameter.
5. Disable the button by setting the **Enabled** property to false.

### ► Implement the PublishService method

1. Locate the **PublishService** method.
2. Set the following the properties for the **Publish** class (which is in the **Microsoft.Uddi** assembly):
  - a. Set the **Publish.Url** property to **comboUrl.SelectedItem.C#** programmers will need to cast **comboUrl.SelectedItem** to a **string**.
  - b. Set the **Publish.AuthenticationMode** property to **AuthenticationMode.UddiAuthentication**.
  - c. Create a new instance of the class **NetworkCredential**. Pass the following values to the constructor:
 

Parameter	Value
Username	txtUser.Text
Password	txtPwd.Text
Domain	GLASGOW
  - d. Assign the instance of **NetworkCredential** to the **Publish.HttpClient.Credentials** property.
3. Create a **BindingTemplate** object.
4. Populate the **BindingTemplate** object.
  - Add a description to the **Descriptions** collection by using the *bindingDescription* parameter.
5. Create a new **AccessPoint** object.
6. Populate the **AccessPoint** object.
  - a. Set the **URLType** property to the value **URLTypeEnum.Http**.
  - b. Set the **Text** property to the value of the *urlAccess* parameter.
7. Assign the **AccessPoint** object to the **BindingTemplate.AccessPoint** property.
8. Create a **TModelInstanceInfo** object.
9. Populate the **TModelInstanceInfo** object.
  - a. Set the **TModelKey** property to the value of the *tModelKey* parameter.
  - b. Add a description to the **Descriptions** collection by using the *tModelDescription* parameter.
10. Add the **TModelInstanceInfo** object to the **BindingTemplate.TModelInstanceDetail.TModelInstanceInfos** collection.
11. Create a **BusinessService** object.
12. Populate the **BusinessService** object.
  - a. Set the **BusinessKey** property to the value of the *businessKey* parameter.
  - b. Set the **Name** property to the value of the *serviceName* parameter.
13. Add the **BindingTemplate** object to the **BusinessService.BindingTemplates** collection.



14. Create a **SaveService** object.
15. Add the **BusinessService** object to the **BusinessServices** collection of the **SaveService** object.
16. Call the **Send** method of the **SaveService** object.
17. Return the **ServiceDetail** object that the **Send** method returns.

► **Implement the btnWoodgroveService\_Click method**

1. Locate the **btnWoodgroveService\_Click** method.
2. Call the **PublishService** method for the business entity of the Woodgrove XML Web service.
  - a. Pass **"Online Bank Web Service"** as the *serviceName* parameter.
  - b. Pass the **bdWoodgrove.BusinessEntities(0).BusinessKey** field as the *businessKey* parameter.
  - c. Pass **tModelKeyWoodgrove** as the *tModelKey* parameter.
  - d. Pass the URL that specifies the endpoint of the Web service as the *urlAccess* parameter. The URL should have the following format:  
`http://machinename/woodgrove/bank.asmx`

---

**Tip** You can use the **Environment.MachineName** property to retrieve the name of your computer.

---

- e. Pass **"ASP.NET web access"** as the *bindingDescription* parameter.
  - f. Pass **"Woodgrove Style Web Service"** as the *tModelDescription* parameter.
3. Disable the button by setting the **Enabled** property to false.

► **Implement the `btnContosoService_Click` method**

1. Locate the `btnContosoService_Click` method.
2. Call the `PublishService` method for the business entity of the Contoso XML Web Service.
  - a. Pass "**Micropayment Web Service**" as the *serviceName* parameter.
  - b. Pass the `bdContoso.BusinessEntities(0).BusinessKey` field as the *businessKey* parameter.
  - c. Pass `tModelKeyContoso` as the *tModelKey* parameter.
  - d. Pass the URL that specifies the endpoint of the Web service as the *urlAccess* parameter. The URL should have the following format:  
`http://machinename/contoso/micropayment.asmx`

---

**Tip** You can use the `Environment.MachineName` property to retrieve the name of your computer.

---

- e. Pass "**ASP.NET web access**" as the *bindingDescription* parameter.
  - f. Pass "**Contoso Style Web Service**" as the *tModelDescription* parameter.
3. Disable the button by setting the `Enabled` property to false.

► **Test the application**

1. Build and run the application.

---

**Tip** It might take a few seconds for the registration code to execute. During this time the user interface will be disabled.

---

2. Click **Register Business** in the **Contoso** group box.
3. Click **Register Service** in the **Contoso** group box.
4. Click **Register Business** in the **Woodgrove** group box.
5. Click **Register Service** in the **Woodgrove** group box.
6. Click **Exit**.

## Exercise 2

### Implementing the FindWebServices Application

In this exercise, you will implement the functionality that is required to locate a WSDL document that is associated with a tModel, and the functionality that is required to locate the endpoint of an XML Web service. The starter application project is provided to you in the folder `<labroot>\Lab06\Starter\FindWebServices`.

#### ► Add references to required assemblies and import required namespaces

1. Add a reference to the **System.Web.Services.dll** assembly.
2. Open the following file:

C#	Visual Basic .NET
Find.cs	Find.vb

3. Import the **System.Net** namespace.

#### ► Implement the btnTName\_Click method

1. Locate the **btnTName\_Click** method.
2. In the **Inquire** class (which is in the **Microsoft.Uddi** assembly), set the following properties:
  - a. Set the **Inquire.Url** property to **http://glasgow/uddi/api/inquire.asmx**.
  - b. Set the **Inquire.AuthenticationMode** property to **AuthenticationMode.UddiAuthentication**.
  - c. Create a new instance of the class **NetworkCredential**. Pass the following values to the constructor:

Parameter	Value
Username	MOCUser
Password	MOC\$Pwd
Domain	GLASGOW

- d. Assign the instance of **NetworkCredential** to the **Inquire.HttpClient.Credentials** property.
3. Create a **FindTModel** object.
  4. Set the **Name** property of the **FindTModel** object to **txtTModel.Text**.
  5. Call the **Send** method of the **FindTModel** object and store the returned **TModelList** object.
  6. Clear the multiline **txtResults** edit control.
  7. Clear the **Items** collection property of the **lstAccess** list control.

8. For each **TModelInfo** object in the **TModelInfos** collection of the **TModelList** object, do the following:
  - a. Append the string representation of the **TModelInfo** to **txtResults.Text**.
  - b. Create a **GetTModelDetail** object.
  - c. Add a **tModelKey** to the **TModelKeys** collection of the **GetTModelDetail** object by using the **TModelInfo.TModelKey** property.
  - d. Call the **Send** method of the **GetTModelDetail** object.
  - e. Store the returned **TModelDetail** object.
  - f. For each **TModel** object in the **TModels** collection of the **TModelDetail** object, do the following:
    - i. Create a **TModelItem** object using the **TModel.OverviewDoc.OverviewURL** property and the **TModelInfo.TModelKey** property.
    - ii. Add the **TModelItem** object to the **lstAccess.Items** collection.

► **Implement the btnBKey\_Click method**

1. Locate the **btnBKey\_Click** method.
2. In the **Inquire** class (which is in the **Microsoft.Uddi** assembly), set the following properties:
  - a. Set the **Inquire.Url** property to **http://glasgow/uddi/api/inquire.asmx**.
  - b. Set the **Inquire.AuthenticationMode** property to **AuthenticationMode.UddiAuthentication**.
  - c. Create a new instance of the class **NetworkCredential**. Pass the following values to the constructor:

Parameter	Value
Username	MOCUser
Password	MOC\$Pwd
Domain	GLASGOW

- d. Assign the instance of **NetworkCredential** to the **Inquire.HttpClient.Credentials** property.
3. Create a **FindBusiness** object.
4. Add a **tModelKey** to the **TModelKeys** collection of the **FindBusiness** object by using **txtBusiness.Text**.
5. Call the **Send** method of the **FindBusiness** object.
6. Store the returned **BusinessList** object.
7. Clear the multiline **txtResults** edit control.

8. For each **BusinessInfo** object in the **BusinessList.BusinessInfos** collection, loop through the **ServiceInfos** collection. For each **ServiceInfo** object in this collection, do the following:
  - a. Create a **FindBinding** object.
  - b. Set the **ServiceKey** property to the value of the **ServiceKey** property of the **ServiceInfo** object.
  - c. Add a **tModelKey** to the **FindBinding.TModelKeys** collection by using **txtBusiness.Text**.  
The **txtBusiness.Text** property is populated when the end user double-clicks an item in the **lstAccess** list.
  - d. Call the **Send** method of the **FindBinding** object.
  - e. Store the returned **BindingDetail** object.
  - f. Loop through the **BindingDetail.BindingTemplates** collection to append the **BindingTemplate.AccessPoint.Text** property to the **txtResults** edit control.

► **Implement the btnBName\_Click method**

1. Locate the **btnBName\_Click** method.
2. In the **Inquire** class (which is in the **Microsoft.Uddi** assembly), set the following properties:
  - a. Set the **Inquire.Url** property to **http://glasgow/uddi/api/inquire.asmx**.
  - b. Set the **Inquire.AuthenticationMode** property to **AuthenticationMode.UddiAuthentication**.
  - c. Create a new instance of the class **NetworkCredential**. Pass the following values to the constructor:

Parameter	Value
Username	MOCUser
Password	MOC\$Pwd
Domain	GLASGOW

- d. Assign the instance of **NetworkCredential** to the **Inquire.HttpClient.Credentials** property.
3. Create a **FindBusiness** object.
4. Set the **FindBusiness.Name** property to the value of **txtBusiness.Text**.
5. Call the **Send** method of the **FindBusiness** object.
6. Store the returned **BusinessList** object.
7. Clear the multiline **txtResults** edit control.
8. Loop through the **BusinessList.BusinessInfos** collection to append the string representation of each **BusinessInfo** object to the multiline **txtResults** edit control.

► **Test the application**

1. Build and run the application.
2. To find the tModel of the Woodgrove XML Web service:
  - a. In the **Find tModel by...** box, type **Bank TModel**.
  - b. Click **Name**.
  - c. Verify that the **Access Points** list contains an entry that resembles the following:  
`http://computername/woodgrove/bank.asmx?WSDL`
  - d. Verify that the **Results** box contains an XML description of the tModel of the Woodgrove XML Web service.
3. To find the tModel of the Contoso XML Web service, repeat step 2 and provide **Micropayment TModel** as the name of the tModel.
4. To find the business entity of the Woodgrove XML Web service:
  - a. In the **Find business by...** box, type **Woodgrove Online Bank**.
  - b. Click **Name**.
  - c. Verify that the **Results** box contains an XML description of the business entity of the Woodgrove XML Web service.
5. To find the business entity of the Contoso XML Web service, repeat step 4 and provide **Contoso Micropayments** as the name of the business.
6. To find the business entity of the Woodgrove XML Web Service:
  - a. In the **Find business by...** box, type the value of the tModel key (including the **uuid:** prefix) for the Woodgrove XML Web Service.

---

**Tip** This value can be found in the **tModelKeyWoodgrove** field in the **frmMain** class of the **RegisterWebServices** project that you completed in Exercise 1.

or

If you search for a tModel as outlined in steps 2 and 3, you can double-click on the tModel in the **lstAccess** listbox. This will populate the **txtBusiness** edit control with the associated key.

---

- b. Click **TModelKey**.
  - c. Verify that the **Results** box contains the following:  
`http://computername/woodgrove/bank.asmx`
7. To find the business entity of the Contoso XML Web Service:
  - a. In the **Find business by...** box, type the value of the tModel key (including the **uuid:** prefix) for the Contoso XML Web Service. This value can be found in the **tModelKeyContoso** field in the **frmMain** class.
  - b. Click **TModelKey**.
  - c. Verify that the **Results** box contains the following:  
`http://computername/contoso/micropayment.asmx`
8. Click **Exit**.

# Review

- Overview of UDDI
- Publishing an XML Web Service
- Finding an XML Web Service
- Publishing an XML Web Service on an Intranet
- Configuring an XML Web Service

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

1. Which specification defines the operations that a UDDI registry supports?

**The UDDI Programmer's API Specification**

2. Which UDDI data structure is used to store the endpoint for an XML Web service?

**bindingTemplate**

3. How can you use UDDI to determine if two XML Web services are compatible?

**You can compare the tModelKey lists for each of the XML Web services. If the lists are identical, then the XML Web services are compatible.**

4. What is the disadvantage of hard-coded XML Web service endpoints?

**XML Web service consumers will not be able to handle a change in the location of an XML Web service.**

5. How can you localize your XML Web service?

**By using resource-only satellite assemblies.**





---

## Module 7: Securing XML Web Services

### Contents

Overview	1
Overview of Security	2
Built-In Authentication	10
Custom Authentication: SOAP Headers	18
Authorization: Role-Based Security	25
Authentication and Authorization with HttpModules	34
Authorization: Code Access Security	39
Encryption	46
Lab 7.1: Securing XML Web Services	54
Review	70
Course Evaluation	72



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, places or events 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, trademarks, 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.

© 2001-2002 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Active Directory, Authenticode, IntelliSense, FrontPage, Jscript, MSDN, PowerPoint, Visual C#, Visual Studio, Visual Basic, Windows NT, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

# Instructor Notes

**Presentation:**  
**120 Minutes**

This module teaches students how to use the security services of the Microsoft® Windows® operating system, Microsoft Internet Information Services (IIS), and the Microsoft .NET Framework and common language runtime to secure XML (Extensible Markup Language) Web services.

**Lab:**  
**60 Minutes**

After completing this module, students will be able to:

- Identify the differences between authentication and authorization.
- Explain how to use the security mechanisms that IIS and Windows provide for authentication.
- Use Simple Object Access Protocol (SOAP) headers for authentication in an XML Web service.
- Use role-based security and code access security for authorization in an XML Web service.
- Encrypt the communication between an XML Web service consumer and an XML Web service.

**Required Materials**

To teach this module, you need the Microsoft PowerPoint® file 2524B\_07.ppt.

**Preparation Tasks**

To prepare for this module:

- Read all of the materials for this module.
- Practice all of the demonstrations.
- Review the walkthrough code files in the *<install folder>\Democode\<language>\Mod07*.
- Complete the lab.

**Dual-language  
PowerPoint macros**

The PowerPoint file for this module contains macros that allow you to switch the displayed code between C# and Microsoft Visual Basic® .NET. To run the macros, you must install the full version of PowerPoint.

To switch a single slide to C#, perform the following steps:

1. Open the PowerPoint deck in PowerPoint.
2. On the **Slide Show** menu, click **View Show**.
3. Locate the slide that you want to switch to C#.
4. Click **C#** on the slide.

To switch a single slide to Visual Basic .NET, perform the following steps:

1. Open the PowerPoint deck in PowerPoint.
2. On the **Slide Show** menu, click **View Show**.
3. Locate the slide that you want to switch to Visual Basic .NET.
4. Click **Visual Basic .NET** on the slide.

---

**Note** You can switch a slide to C# or Visual Basic .NET at any time while displaying the slides. Just click **C#** or **Visual Basic .NET** to switch between the languages.

---

## Module Strategy

Use the following strategy to present this module:

- Overview of Security

Ensure that students understand the difference between authentication and authorization. Also, explain why encryption might be necessary even with authenticated clients. Explain that the .NET Framework can assist with authorization and authentication. Explain the limitations of using Windows authentication and the IIS-supported authentication mechanisms.

- Built-In Authentication

Explain the authentication support that is built into the Windows operating system and IIS. Explain the scenarios where Windows authentication is appropriate and where it is inappropriate for XML Web services.

- Custom Authentication: SOAP Headers

Explain how students can use SOAP headers to send authentication information to an XML Web service. Explain the mechanics of using SOAP headers. Be sure to explain how students can use SOAP headers to communicate information from a client to an XML Web service or from an XML Web service to a client. Also, point out that students can make a SOAP header optional for an XML Web service method.

- Authorization: Role-Based Security

In this module you will explain how to implement a custom role-based authorization mechanism. You need to explain why Windows discretionary access control list (DACL) based authorization is often not appropriate in XML Web service scenarios. Focus on how students can use **GenericPrincipal** and **GenericIdentity** objects to implement custom authorization mechanism and how this would be useful in the context of XML Web services.

- Authorization: Code Access Security

Explain why code access security is required and how students can use it in the context of XML Web services. Ensure that you explain how permissions are verified at load time and run time. Emphasize how the deployment environment for an XML Web service can affect the permissions that are granted to it.

- Encryption

Explain why encryption of the communication between an XML Web service and a consumer of the XML Web service might be necessary. Briefly discuss Secure Socket Layer (SSL), describing the performance impact. Explain how to use SOAP extensions to encrypt various parts of a SOAP message. In this context, explain the changes that students need to make to the XML Web service proxy. Emphasize that because the proxies are generated, any editing of the proxy class will be lost if the class is regenerated.



# Overview

- Overview of Security
- Built-In Authentication
- Custom Authentication: SOAP Headers
- Authorization: Role-Based Security
- Authorization: Code Access Security
- Encryption

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

Just like any Web-based application, you must make XML (Extensible Markup Language) Web services secure from accidental or deliberate misuse. There are three aspects of security that this module addresses in the context of XML Web service—authentication, authorization, and secure communication.

To adequately secure an XML Web service you must understand how to use the Microsoft® Windows® operating system and Microsoft Internet Information Services (IIS) to authenticate XML Web service consumers, how the Microsoft .NET Framework and common language runtime can assist in the task of authorization, and techniques for securing the messages that are exchanged between the XML Web service and a consumer by encrypting all or part of the messages.

## Objectives

After completing this module, you will be able to:

- Identify the differences between authentication and authorization.
- Explain how to use the security mechanisms that IIS and Windows provide for authentication.
- Use SOAP headers for authentication in an XML Web service.
- Use role-based security and code access security for authorization in an XML Web service.
- Encrypt the communication between an XML Web service consumer and an XML Web service.

## Overview of Security

- Authentication vs. Authorization
- Types of Authentication
- Types of Authorization
- Methods of Encryption

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Before looking at how to implement authentication, authorization, and encryption in an XML Web service, you must understand some of the concepts and terminology that are related to security. Also, you will look at some of the options that are available for providing authentication, authorization, and encryption in Microsoft ASP.NET Web Service applications.



## Authentication vs. Authorization

### ■ Authentication

- Process of finding and verifying the identity of a user
- Performed against an authentication authority

### ■ Authorization

- Process of determining if a user's request to perform some action is allowed to proceed
- Occurs after authentication
- Based on user's identity

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Before you can secure an XML Web service, you must understand the differences between authentication and authorization.

### Authentication

Authentication is the process of discovering and verifying the identity of a user by examining the user's credentials and then validating those credentials against some authentication authority. Currently, applications use a variety of authentication mechanisms, and you can use some of these mechanisms with the .NET Framework role-based security. Examples of commonly used mechanisms include the authentication mechanisms of the operating system (specific examples include NTLM and Kerberos version 5 authentications), Microsoft Passport, and application-defined mechanisms.

### Authorization

Authorization is the process of determining whether a user is allowed to perform a requested action. Authorization occurs after authentication and uses information about a user's identity and roles to determine the resources that a user can access. You can use .NET Framework role-based security to implement authorization.

## Types of Authentication

- IIS authentication
- ASP.NET authentication
- Forms authentication
- Passport authentication
- Custom SOAP header authentication

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

The first step in implementing security in any application is to authenticate users. Implementing a robust authentication mechanism is not easy, and if possible it is recommended that you use the authentication services that the platform provides. In this case, the platform you are looking at consists of the Windows operating system, IIS, and the .NET Framework and common language runtime.

### IIS authentication

IIS offers the following three mechanisms for authentication:

- *Basic* authentication
- *Digest* authentication
- *Integrated Windows* authentication

You will learn the details of these authentication mechanisms and how to use them to secure an XML Web service later in this module.

### ASP.NET authentication

ASP.NET supports two new authentication mechanisms, which ASP did not support:

- *Forms* authentication
- *Passport* authentication

### Forms authentication

Forms authentication is a mechanism by which unauthenticated requests are redirected to a Hypertext Markup Language (HTML) form using Hypertext Transfer Protocol (HTTP) client-side redirection. A user provides credentials in the form and submits it. If the Web application authenticates the request, the system issues a form, usually to a browser, which contains the credentials or a key for reacquiring the identity of the user. Subsequent requests are issued with the form in the request headers. An ASP.NET handler authenticates and authorizes these requests by using the validation method that the application developer specifies.

**Passport authentication**

Passport is a centralized authentication service that Microsoft provides and that offers a single logon feature and core profile services for member Web sites.

Both Forms authentication and Passport authentication are mentioned for the sake of completeness. You cannot easily use either of these mechanisms within an XML Web service. Both of these mechanisms present a logon screen that requires interaction with an end user, and both support logon timing out. But an XML Web service consumer cannot programmatically process the logon screen or handle a timed-out logon condition.

Because it is not recommended that you use Forms and Passport authentication within XML Web services, this module does not cover these authentications in any further detail. It may become possible to use Passport for XML Web service authentication in the future.

**Custom SOAP header authentication**

If you do not want to use built-in authentication mechanisms, then you can implement a custom authentication mechanism instead. You might not want to pass user credentials as part of the parameter list for every method in your XML Web service. In such a situation, you would need another way to pass the credentials. Simple Object Access Protocol (SOAP) headers are a convenient way to accomplish this task. An XML Web service consumer can add user credentials to the SOAP header. The XML Web service can then retrieve this information to perform custom authentication.

## Types of Authorization

- Windows NT security
- Role-based security
- Code access security
- Configuring authorization in an ASP.NET application

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

The ASP.NET environment, the .NET Framework, and the Windows platform provide several techniques for authorizing access to a system resource. The resources that can be accessed are the intersection of resources that are authorized to:

- A user by the Microsoft Windows NT® security system.
- The assembly by code access security.
- Optionally, the user's role by role-based security.

The folder that contains the root of an ASP.NET application is the root of a logical Universal Resource Identifier (URI) namespace. You can configure an ASP.NET application to further restrict access to the application's URI namespace based on user identity or role. For example, you could restrict access to subfolders below your application root.

This topic examines each of these authorization techniques in more detail.

### Windows NT security

Microsoft Windows NT provides security features that are based on user identity, and that prevent unauthorized access to system resources. These features are user authentication and object-based access control. It is important to note that with Windows security, after a user is authenticated, most code that that user runs has access to all of the resources that he or she can access.

Windows administrators can create discretionary access control lists (DACL) that control access to resources or objects on a network. Administrators can assign security descriptors that contain DACLs that list the users and groups that are granted access to objects such as a files, printers, or services.

### Role-based security

Role-based security is a security model where the specific identity of the user is not important. What is important are the logical roles that a user can assume. Role-based security uses the roles that are associated with a user to make decisions about security authorizations.

**Code access security**

Code access security is a security mechanism that you can use to prevent code from accessing protected resources. Just like role-based security, code access security requires that the user first be authenticated before code access security can operate.

**Configuring authorization in an ASP.NET application**

You can further control authorization to parts of an ASP.NET application's URI namespace with the **<authorization>** section of an ASP.NET application configuration file (Web.config). To use ASP.NET authorization, you place either a list of users or roles, or both, in the **allow** or **deny** elements of the **<authorization>** section of Web.config.

To define the conditions for accessing a particular folder, place a Web.config file that contains an **<authorization>** section in that folder. The conditions set for that folder also apply to its subdirectories, unless configuration files in a subdirectory override them. The syntax for the **<authorization>** section is as follows:

```
<[allow|deny] [users] [roles] [verbs] />
```

For an **allow** or **deny** element, you must specify either the **users** or the **roles** attribute. You can include both attributes, but both are not required together in an **allow** or **deny** element. The **verbs** attribute is optional.

The **allow** and **deny** elements grant and revoke access, respectively. Each of these elements support three attributes, which are defined in the following table.

Attribute	Description
Roles	Identifies a targeted role for this element.
Users	Identifies the targeted identities for this element.
Verbs	Defines the HTTP verbs to which the action applies, such as <b>GET</b> , <b>HEAD</b> , or <b>POST</b> .

The following example grants access to Mary, while denying it to Adam and all anonymous users ("?" indicates anonymous users):

```
<configuration>
  <system.web>
    <authorization>
      <allow users="CONTOSO\Mary"/>
      <deny users="CONTOSO\Adam" />
      <deny users="?" />
    </authorization>
  ...
```

## Methods of Encryption

- **Choosing what to encrypt**
  - Entire message,
  - Only body or header of message
  - Only selected messages
  - Partitioning
- **Some encryption options**
  - SSL
  - Custom SOAP extensions

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Although authentication and authorization together prevent unauthorized users from accessing system resources, neither prevents the interception of the data that is exchanged between the XML Web service consumer and the XML Web service. You use encryption to ensure the secure transfer of data.

### Choosing what to encrypt

Encryption is expensive and therefore you must take care in choosing the communication that must be encrypted. There are a number of options that you can consider:

- **Encrypt the whole message**

Although this is relatively easy to do, it provides very poor performance because it is unlikely that every communication requires absolute privacy.
- **Encrypt only the body of messages**

This is less computationally expensive than encrypting everything, but may still be more than what is required.
- **Encrypt only the headers of messages**

In XML Web services, authentication information is often provided in SOAP headers. You would not want this information to be visible except to the intended recipient, so encrypting the headers is a relatively cheap solution.
- **Encrypt only selected messages**

This requires the most work from the developer, but generally provides a well-tailored tradeoff between security and performance.

- No encryption

If the communicated data is not sensitive, you should not incur a performance penalty by encrypting any part of the communication.

- Partition your XML Web service

The idea here is to factor the service interface into groups of messages that require encryption and those that do not. You can then implement the interface by using two XML Web services, one for the methods that require encryption and one for the methods that do not. Using two XML Web services allows you to easily secure only the methods that require encryption, and avoid a performance penalty for those methods that do not.

### **Some encryption options**

There are many different options for encrypting communications, two of which are: Secure Socket Layer (SSL) and custom SOAP extensions.

- Secure Socket Layer

Using SSL is a simple way to encrypt the entire communication between an XML Web service consumer and an XML Web service.

- Custom SOAP extensions

If you need more detailed control, you can implement a custom SOAP extension to encrypt only sensitive data. You can transfer data that is not sensitive unencrypted, therefore providing better performance than SSL.

## Built-In Authentication

- Basic and Digest Authentication
- Integrated Windows Authentication
- Using IIS Authentication on a Web Server

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

IIS provides a number of built-in authentication mechanisms that XML Web services can use. To use any of the IIS authentication mechanisms, you must configure IIS and set the correct authentication mode in the Web.config file of an ASP.NET Web Service. For an XML Web service to authenticate its client, the client must programmatically provide the required user name and password credentials.

In this section, you will examine Basic, Digest, and Integrated Windows authentication.



## Basic and Digest Authentication

### ■ Basic authentication

- IIS prompts for a valid Windows user name/password
- Credentials sent via clear-text – not secure!
- Basic authentication with SSL hides password

### ■ Digest authentication

- Credentials hashed
- Supported by HTTP 1.1 clients only

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Before you learn how to use Basic and Digest authentication to secure an XML Web service, it is important to understand how these authentication mechanisms work.

### Basic authentication

The Basic authentication mechanism is a widely used, industry-standard method for collecting user name and password information. Basic authentication works in the following manner:

1. A Web browser on a client computer displays a dialog box where users can enter their previously assigned Windows 2000 account user names and passwords.
2. The browser then attempts to establish a connection with a Web server by using the supplied credentials. (The password is Base64 encoded before it is sent over the network.)
3. If the server rejects the credentials, the browser repeatedly displays the dialog box until the user either enters a valid user name and password, or closes the dialog box.
4. After the server verifies that the user name and password correspond to a valid Windows user account, a connection is established.

The advantage of Basic authentication is that it is part of the HTTP specification, and most browsers support it. The disadvantage is that Web browsers using Basic authentication transmit passwords in an unencrypted form. By monitoring communications on your network, someone could easily intercept and decipher these passwords by using publicly available tools. Therefore, Basic authentication is not recommended unless used in conjunction with Secure Socket Layer (SSL) or the connection between the client and the Web server is secure, such as a direct cable connection or a dedicated line.

---

**Note** Integrated Windows authentication takes precedence over Basic authentication. The browser will choose Integrated Windows authentication and will attempt to use the current Windows logon information before prompting the user for a user name and password.

---

### Digest authentication

The most recent industry standard development in Web security is the Digest authentication specification. Digest authentication is slated to be a replacement for the Basic authentication. The World Wide Web Consortium (W3C) endorsed Digest authentication to fix the security gaps in the Basic authentication mechanism.

Digest authentication uses a hashing algorithm to form a hexadecimal representation of a combination of user name, password, the requested resource, the HTTP method, and a randomly-generated value that the server returns.

*Hashing* is a one-way process of passing authentication credentials. The result of this process is called a *hash*, or message digest, and it is not feasible to decrypt it. That is, the original text cannot be deciphered from the hash.

Digest authentication is not as secure as Kerberos or a client-side key implementation, but it does represent a stronger form of security than Basic authentication.

Digest authentication is an HTTP 1.1 specification, which requires that a client be compliant with this specification. Because a hashing function must encrypt the user name and password, the browser must perform the hashing prior to submitting it to the server. If an IIS 5.0 virtual directory has Digest authentication enabled, a request from a browser that is not HTTP 1.1-compliant will generate an error in the client request. Microsoft Internet Explorer 4.0 was the first HTTP 1.1-compliant browser available from Microsoft.

## Integrated Windows Authentication

### ■ Characteristics

- Previously called NTLM or Windows NT Challenge/Response authentication
- Secure form of authentication because the user name and password are not sent across the network

### ■ Limitations

- Not all XML Web service clients support this authentication method
- Integrated Windows authentication does not work over HTTP proxy connections
- Additional TCP ports have to be opened in the firewall

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Many of the XML Web services that are created may not be publicly accessible. For such XML Web services, it is viable to use Integrated Windows authentication to secure the XML Web service.

### Characteristics

Integrated Windows authentication was previously known as NTLM or Windows NT Challenge/Response authentication. Integrated Windows authentication is a secure form of authentication because the user name and password are not sent across the network. When you enable Integrated Windows authentication, the client browser proves its identity by sending a hash of its credentials to the server.

Integrated Windows authentication can use both the Kerberos version 5 (v5) authentication protocol and its own challenge/response authentication protocol. If Active Directory® directory service is installed on the server, and the browser is compatible with the Kerberos v5 authentication protocol, both the Kerberos v5 protocol and the challenge/response protocol are used; otherwise only the challenge/response protocol is used.

---

**Note** The Kerberos v5 protocol is a network authentication protocol. It is designed to provide strong authentication for client/server applications by using secret-key cryptography. A free implementation of this protocol is available from the Massachusetts Institute of Technology. Kerberos protocol is also available in many commercial products, including Microsoft Windows 2000.

---

For Kerberos v5 authentication to be successful, both the client and server must have a trusted connection to a Key Distribution Center (KDC) and be Directory Services compatible. For more information about the Kerberos protocol, see the Windows 2000 documentation.

**Limitations**

Although integrated Windows authentication is secure, it has its limitations too.

- Not all XML Web service clients support integrated Windows authentication. However, XML Web service consumers using **SoapHttpClientProtocol**-derived proxy classes (this includes proxy classes created with Wsdl.exe or Microsoft Visual Studio® .NET) support integrated Windows authentication.
- Integrated Windows authentication does not work over HTTP proxy connections.
- Additional TCP ports have to be opened in the firewall, because Integrated Windows authentication does not use port 80.

For these reasons, integrated Windows authentication is best suited for an intranet environment, where both client (user) and Web server computers are in the same domain, and where administrators can ensure that all clients will be compliant.

## Using IIS Authentication on a Web Server

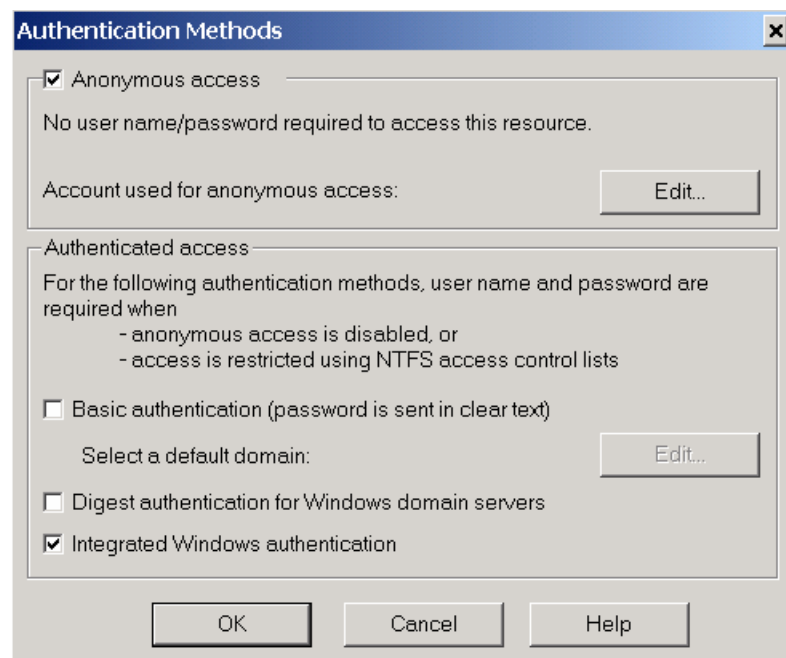
- Configuring authentication in IIS
- Configuring an ASP.NET XML Web service
- Accessing user identity in an XML Web service
- Providing credentials

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

To use IIS authentication to secure an ASP.NET XML Web Service, you must configure IIS in addition to the XML Web service.

### Configuring authentication in IIS

When securing an XML Web service, you can use any of the built-in IIS authentication mechanisms. The following illustration shows the dialog box in the Internet Service Manager from where you can select the authentication mechanism that you want to use:



If you select Basic authentication, you must ensure that the accounts that can access an XML Web service are granted permission to log on to the Web server that is hosting the service. This is necessary because Basic authentication impersonates a local user, and by default, domain accounts do not have permission to log on to a Web server.

If you select Digest authentication, the Windows account that you configure must be an account in a domain. The domain controller must also have a plain text copy of the password that is used because it must perform a hashing operation and compare the results with the hash that the browser sends.

If you select Integrated Windows authentication, the user will not be prompted for credentials unless the authentication fails. Remember that Integrated Windows authentication does not work across proxy servers or other firewall applications.

#### Configuring an ASP.NET XML Web service

To use Windows (Basic, Digest, or Integrated Windows) authentication with an XML Web service, the authentication mode in the Web.config file must be set to **Windows**, as shown in the following code:

```
<configuration>
  <system.web>
    <authentication mode = "Windows"/>
  ...
```

#### Accessing user identity in an XML Web service

Code in XML Web service methods can access identity information about an authenticated user by accessing the **Context.User.Identity** property. The following sample code shows how to access the authenticated user's name in an XML Web service method.

##### C#

---

```
[WebMethod]
public string HelloWorld()
{
    return "Hello" + Context.User.Identity.Name;
}
```

##### Microsoft Visual Basic® .NET

---

```
Public<WebMethod()> _
Function HelloWorld() As String
    Return "Hello" + Context.User.Identity.Name
End Function 'HelloWorld
```

#### Providing credentials

Proxy classes that are created by using the Web Services Description Language (WSDL) tool (Wsdl.exe), or created when adding a Web reference in Visual Studio .NET, derive from the **SoapHttpClientProtocol** class. These classes have a **Credentials** property, which can be used to obtain or set security credentials for XML Web service client authentication.

To use the **Credentials** property, an XML Web service client must create an instance of a class implementing the **ICredentials** interface, such as the **NetworkCredential** class. Then the client must set credentials that are specific to the authentication mechanism before making a call to an XML Web service method.

You can use the **NetworkCredential** class to set authentication credentials by using the Basic, Digest, or Integrated Windows authentication mechanisms. If authentication fails, the call to the XML Web service method will throw an exception of type **System.Net.WebException**.

The following code sample shows how to authenticate from a client application.

#### C#

```

1.      HelloService service = new HelloService();
2.      //Create a NetworkCredential object
3.      ICredentials credentials = new
        ↪NetworkCredential("Administrator", "password",
        ↪"woodgrovebank.com");
4.      //Set the client-side credentials using the Credentials
        ↪property
5.      service.Credentials = credentials;
6.      //Invoke the XML Web service method
7.      string s;
8.      try {
9.          s = service.HelloWorld();
10.     } catch
11.         Console.WriteLine("Authentication failed");
12.     }

```

#### Visual Basic .NET

```

1.      Dim service As New HelloService()
2.      'Create a NetworkCredential object
3.      Dim credentials = New NetworkCredential("Administrator",
        ↪"password", "woodgrovebank.com");")
4.      'Set the client-side credentials using the Credentials
        ↪property
5.      service.Credentials = credentials '
6.      'Invoke the XML Web service method
7.      Dim s As String
8.      Try
9.          s = service.HelloWorld()
10.     Catch
11.         Console.WriteLine("Authentication failed")
12.     End Try

```

The functionality of the preceding code can be described as follows:

- In line 1, an instance of an XML Web service proxy class is created.
- In line 3, a **NetworkCredentials** object is created and a user name, password, and domain information are supplied.
- In line 5, the **credentials** object is assigned to the **Credentials** property of the XML Web service proxy.
- In lines 8 through 12, the call to an XML Web service method is enclosed in a **try** block, so that if authentication fails, the **System.Net.WebException** exception can be caught.

## Custom Authentication: SOAP Headers

- Using a SOAP Header in an XML Web Service
- Using a SOAP Header in an XML Web Service Consumer

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Windows authentication works well for intranet scenarios in which you are authenticating a user in your own domain. If your XML Web service has Windows authentication mode set in Web.config, then you must create a local or domain account for each user. This is not a practical solution for applications with large numbers of users, and it is impossible for applications on the Internet.

For the Internet, you probably want to perform custom authentication and authorization, perhaps against a Structured Query Language (SQL) database. In that case, you should pass custom credentials (such as the user name and password) to your XML Web service and let it handle the authentication and authorization.

A convenient way to pass extra information along with a request to an XML Web service is a SOAP header. The XML Web service consumer adds user ID and password information to the SOAP header. The XML Web service methods retrieve this information and use it to perform custom authentication.

The only significant issue that you must resolve when using SOAP headers to transfer credentials is security. You can resolve this issue by strongly encrypting the identity information in the SOAP header.



## Using a SOAP Header in an XML Web Service

- Derive a class from SoapHeader
- Add a public field of the SoapHeader-derived type
- Apply the SoapHeader attribute

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

This topic examines how you can define a SOAP header for an XML Web service and use that header in the XML Web service method. The following code sample demonstrates this functionality.

### C#

```
1.      using System.Web.Services;
2.      using System.Web.Services.Protocols;
3.
4.      public class AuthHeader : SoapHeader
5.      {
6.          public string Username;
7.          public string Password;
8.      }
9.      public class AccountService : WebService
10.     {
11.         public AuthHeader sHeader;
12.         [WebMethod]
13.         [SoapHeader("sHeader",Required=false)]
14.         public decimal GetAcctBalance(string acctID) {
15.             ...
16.         }
17.     }
```

**Visual Basic .NET**

```
1. Imports System.Web.Services
2. Imports System.Web.Services.Protocols
3.
4. Public Class AuthHeader
5.     Inherits SoapHeader
6.     Public Username As String
7.     Public Password As String
8. End Class 'AuthHeader
9. Public Class AccountService
10.    Inherits WebService
11.    Public sHeader As AuthHeader
12.    <WebMethod(), _
13.    SoapHeader("sHeader", Required := False)> _
14.    Public Function GetAcctBalance(acctID As String) As Decimal
15.    '...
16.    End Function 'GetAcctBalance '
17. End Class 'AccountService
```

**Code explanation**

The functionality that the preceding code implements can be described as follows:

- In lines 4 through 8, a class named **AuthHeader**, which inherits from **SoapHeader**, is defined.
- In line 11, a field of type **AuthHeader** is added to the class that implements an XML Web service.
- In line 13, the **SoapHeader** attribute is applied to the XML Web service method.

Note that the name provided to the attribute constructor is the name of the field that was added in line 11.

XML Web services set the value of a header field for input headers before a method is called, and retrieve the value for output headers when the method returns.

**Disabling other authentication types**

To implement a custom authentication scheme by using SOAP headers, you must also disable other authentication types in the Web.config file for your XML Web service, as shown in the following code:

```
<configuration>
  <system.web>
    <authentication mode = "None"/>
    ...
  </system.web>
  ...
</configuration>
```

## Using a SOAP Header in an XML Web Service Consumer

- SOAP headers in WSDL
- XML Web service proxies and SOAP headers
- Using SOAP headers when calling XML Web services

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In the previous topic, you learned how to define a SOAP header and use it in an XML Web service method. In this topic, you will learn how a client can populate a SOAP header.

The SOAP header elements that an XML Web service requires are specified in the WSDL contract for the XML Web service. These elements are made available to a client when a proxy class is created from WSDL.exe or created by adding a Web reference within Visual Studio .NET.

### SOAP headers in WSDL

The following partial code listing shows how a SOAP header is defined in a WSDL file:

```
1.      <?xml version="1.0" encoding="utf-8" ?>
2.      <definitions namespaces deleted for brevity >
3.      ...
4.      - <types>
5.      ...
6.      <s:element name="WoodgroveAuthInfo"↵
7.          type="s0:WoodgroveAuthInfo" />
8.      - <s:complexType name="WoodgroveAuthInfo">
9.      - <s:sequence>
10.         <s:element minOccurs="1" maxOccurs="1" name="Username"↵
11.             nillable="true" type="s:string" />
12.         <s:element minOccurs="1" maxOccurs="1" name="Password"↵
13.             nillable="true" type="s:string" />
14.     </s:sequence>
15. </s:complexType>
16. - <s:element name="GetAccount">
17.     ...
18. </s:element>
19. - <s:element name="GetAccountResponse">
20.     ...
21. </s:element>
22.     ...
23. </types>
24. - <message name="GetAccountSoapIn">
25.     <part name="parameters" element="s0:GetAccount" />
26. </message>
27. - <message name="GetAccountSoapOut">
28.     <part name="parameters" element="s0:GetAccountResponse" />
29. </message>
30. - <message name="GetAccountWoodgroveAuthInfo">
31.     <part name="WoodgroveAuthInfo"↵
32.         element="s0:WoodgroveAuthInfo" />
33. </message>
34.     ...
35. - <portType name="Woodgrove Online BankSoap">
36.     ...
37. - <operation name="GetAccount">
38.     <input message="s0:GetAccountSoapIn" />
39.     <output message="s0:GetAccountSoapOut" />
40. </operation>
41.     ...
42. </portType>
43. - <binding name="Woodgrove Online BankSoap" ↵
44.     type="s0:Woodgrove Online BankSoap">
45.     <soap:binding↵
46.         transport="http://schemas.xmlsoap.org/soap/http" ↵
47.         style="document" />
48.     ...
49. - <operation name="GetAccount">
50.     <soap:operation soapAction="http://tempuri.org/GetAccount"↵
51.         style="document" />
52. - <input>
53.     <soap:body use="literal" />
54.     <soap:header message="s0:GetAccountWoodgroveAuthInfo"↵
55.         part="WoodgroveAuthInfo" use="literal" />
56. </input>
```

*(continued)*

```

57.
58.      - <output>
59.        <soap:body use="literal" />
60.      </output>
61.    </operation>
62.    ...
63.  </binding>
64.  - <service name="Woodgrove Online Bank">
65.    <documentation>Woodgrove banking services</documentation>
66.  - <port name="Woodgrove Online BankSoap" ↪
67.    binding="s0:Woodgrove Online BankSoap">
68.    <soap:address location="http://www.woodgrovebank.com/↪
69.      woodgrove/bank.asmx" />
70.  </port>
71.  ...
72. </service>
73. </definitions>

```

The definitions that the preceding code implements can be described as follows:

- In lines 8 through 15, a complex type named **WoodgroveAuthInfo** is defined. The type has two child elements, named **Username** and **Password**.
- In lines 6 through 7, an element named **WoodgroveAuthInfo** is of type **WoodgroveAuthInfo**.
- In lines 30 through 33, a message named **GetAccountWoodgroveAuthInfo** is defined.
- In lines 54 through 55, a SOAP header whose message is the **GetAccountWoodgroveAuthInfo** is added to the input communication of the **GetAccount** operation.

#### XML Web service proxies and SOAP headers

The following code shows part of the resulting proxy code that is generated for the previous WSDL file.

**C#**

---

```

[SoapHeaderAttribute("WoodgroveAuthInfoValue",
Required=false)]
//...other attributes omitted for brevity...
public Acct GetAccount(int acctID) {
...

```

**Visual Basic**

---

```

<SoapHeaderAttribute("WoodgroveAuthInfoValue", Required :=
↪False)> _
//...other attributes omitted for brevity...
Public Function GetAccount(int acctID) As Acct
...

```

In the preceding code, an attribute of type **SoapHeaderAttribute** is applied to the proxy method. Because the **Required** property is **false** (**False** for Visual Basic .NET), the SOAP header is optional. If the *Required* parameter is not specified, or has the value **true** (**True** for Visual Basic .NET), then the header is required to call this method.

**Using SOAP headers  
when invoking XML Web  
services**

After the proxy is generated, the consumer then directly sets the header for the proxy class before making a method call that requires it. The following example shows how the header is populated with credential details.

**C#**

---

```
WoodgroveOnlineBank bank = new WoodgroveOnlineBank();
WoodgroveAuthInfo authInfo = new WoodgroveAuthInfo ();
authInfo.Username = "Adam";
authInfo.Password = "password";
bank.WoodgroveAuthInfoValue = authInfo;
Acct acct = bank.GetAccount(1);
```

**Visual Basic .NET**

---

```
Dim bank As New WoodgroveOnlineBank()
Dim authInfo As New WoodgroveAuthInfo()
authInfo.Username = "Adam"
authInfo.Password = "password"
bank.WoodgroveAuthInfoValue = authInfo
Dim acct As Acct = bank.GetAccount(1)
```

## Authorization: Role-Based Security

- Identities
- Principals
- Using `WindowsIdentity` and `WindowsPrincipal` Objects
- Using `GenericIdentity` and `GenericPrincipal` Objects
- Authentication and Authorization with `HttpModules`

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

There are many different ways in which you can perform authorization. The .NET Framework and the common language runtime support two security models: role-based security and code access security. In this section, you will examine the role-based security model and how you can use it in XML Web services to implement authorization.

Microsoft Component Object Model (COM+) services introduced the concept of role-based security. COM+ services allow application developers to define roles which are meaningful within an application. In COM+ services role-based security, Windows accounts are added to roles. Adding or removing accounts to roles is done when configuring a COM+ application and this involves no code modifications. You use roles to control access to application functionality. For example, a human resources application can define the roles *Manager* and *Employee*. Users in the *Managers* role might have access to a **GetEmployeeSalary()** method, whereas users in the *Employee* role might not.

The .NET Framework extends the idea of role-based security by using two concepts extensively: *principals* and *identities*. This section introduces the **Principal** and **Identity** classes and explains how to use these classes in an XML Web service to verify the role of a user and provide access to resources that are based on that role.

You can use role-based security in conjunction with built-in authentication mechanisms like IIS Basic or Digest authentication, or in conjunction with a custom authentication mechanism. For example, you may want to authenticate a user by querying a database.

## Identities

- Windows identity
- Generic identity
- Custom identity

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

An identity object encapsulates information about a user or an entity that has been validated. Some examples of the encapsulated information are the user name and authentication type. The .NET Framework provides four identity types:

- **FormsIdentity**
- **GenericIdentity**
- **PassportIdentity**
- **WindowsIdentity**

Currently, you can use only **GenericIdentity** and **WindowsIdentity** in ASP.NET Web Services. You can also implement your own identity types.

### Windows identity

The **WindowsIdentity** class represents the identity of a user that is based on a method of authentication that Windows supports. A Windows identity provides the ability to impersonate a user other than one who is associated with the thread that is currently executing, so that resources can be accessed on behalf of that user.

### Generic identity

The **GenericIdentity** class represents the identity of a user based on a custom authentication method, which an application defines. For example, an application can perform a database lookup to authenticate a user.

### Custom identity

The **GenericIdentity** class can only store an authenticated user's name. If you decide that you need an identity that can hold custom-user information, you can create a class that implements the **IIdentity** interface. This kind of class is known as a custom identity.



# Principals

- **Principal represents the security context under which code is running**
- **What are roles?**
  - Named set of principals that have the same privileges with respect to security
- **Windows principal**
- **Generic principal**
- **Custom principal**
- **Principals and call context**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

A principal object represents the security context under which code is running. This includes the identity of the user as represented by an associated identity object, and the roles associated with the user. A principal can be a member of one or more roles. Therefore, applications can use role membership to determine whether a principal is authorized to perform a requested action.

## What are roles?

A role is a named set of principals that have the same privileges with respect to security (such as a teller or a manager). A role defines a group of related users of an application. For example, a banking application might impose limits on the withdrawal amounts that can be transacted, based on a role. In this scenario, tellers might be authorized to process withdrawals that are less than a specified amount, while only managers are allowed to process withdrawals in excess of that amount.

Role-based security in the .NET Framework supports two principal types:

- **WindowsPrincipal**
- **GenericPrincipal**

You can also define your own principal types.

## Windows principal

The **WindowsPrincipal** class represents Windows users and their roles. The roles are the Windows groups that a user is a member of.

## Generic principal

The **GenericPrincipal** class represents users and roles that exist independent of Windows users and their roles. Essentially, the generic principal provides a simple way for an application to perform custom authentication and authorization.

## Custom principal

A **GenericPrincipal** stores an identity and a list of roles to which the identity belongs. If you need to store application-specific role information, you can create a class that implements the **IPrincipal** interface. Any user-defined class that implements the **IPrincipal** interface is known as a custom principal.

**Principals and call context**

All .NET Framework applications are hosted in an application domain. Each application domain has an object of type **CallContext** created for it.

A principal object is bound to a call context. When a new thread is created, the call context will flow to the new thread. This means that the principal object reference is automatically copied to the new thread's call context. As a result, there is always a principal and identity available on whichever thread is the current thread to allow you to perform application-level authentication and authorization.

---

**Note** In role-based security, you can base the identity (and the principal it helps to define) on either a Windows account or the identity can be a custom identity unrelated to a Windows account. .NET applications can make authorization decisions based on a principal's identity or role membership, or both.

---

## Using WindowsIdentity and WindowsPrincipal Objects

- Using the Name property of the Identity object to control access to code based on Windows account

```
if (User.Identity.Name == "CONTOSO\\fred")
// Permit access to some code.
```

- Using the IsInRole method to control access to code

```
if (User.IsInRole("CONTOSO\\Administrators"))
// Permit access to some code.
```

C#

Visual Basic

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

You can control access to code based on the name of a user's identity, as shown in the first example on the slide. When an XML Web service uses Windows authentication, the authentication module attaches a **WindowsPrincipal** object to the application context. Also, the authentication module assigns a **WindowsPrincipal** object to the **User** property of the **HttpContext** class. The **Identity** property of this **WindowsPrincipal** references a **WindowsIdentity** object that represents a user's logon name and the user's domain.

For **WindowsPrincipal** objects, a role maps to a Windows group, including the domain. When checking for role membership in built-in Windows groups, you can use the **WindowsBuiltInRole** enumeration.

### C# and Visual Basic .NET code examples

The following example uses a hard-coded string in the call to determine if a user is a member of the built-in **Administrators** role.

#### C#

```
if (User.IsInRole("BUILTIN\\Administrators")) {
// only administrators can do this
}
```

#### Visual Basic .NET

```
If User.IsInRole("BUILTIN\\Administrators") Then
' only administrators can do this
End If
```

You can also verify the role membership of a principal object by calling the **IsInRole** method on that object, as shown in the preceding code. The preceding code works, but it cannot be easily localized. The following example uses the **WindowsBuiltInRole** enumeration instead, and can be more easily localized.

**C#**

---

```
if (User.IsInRole(WindowsBuiltInRole.Administrator)
{
// only administrators can do this
}
```

**Visual Basic .NET**

---

```
If User.IsInRole(WindowsBuiltInRole.Administrator) Then
    ' only administrators can do this
End If
```

The following XML Web service method returns identity and role information about an authenticated user.

**C#**

---

```
using System.Security.Principals;
// Required for role-based security
...
[WebMethod]
public string HelloWorld()
{
    // not required, but just to show the User property is an
    Identity
    IIdentity identity = Context.User.Identity;
    string name = User.Identity.Name;
    bool isAuth = User.Identity.IsAuthenticated;
    string identType = User.Identity.AuthenticationType;
    bool isAdmin = User.IsInRole("Domain\\Administrators");

    StringBuilder s;
    s.Append("Hello " + name + ", ");
    s.Append(isAuth ? "authenticated" : "not authenticated");
    s.Append(" using " + identType + ", ");
    s.Append("you are ");
    s.Append(isAdmin ? "an admin" : "not an admin");
    return s.ToString();
}
```

**Visual Basic .NET**

---

```
Imports System.Security.Principals
' Required for role-based security
,
'<WebMethod> _
Public Function HelloWorld() As String '
    ' Not required, but just to show the User property is an
Identity
    Dim identity As IIdentity = Context.User.Identity
    Dim name As String = User.Identity.Name
    Dim isAuthenticated As Boolean = User.Identity.IsAuthenticated
    Dim identType As String = User.Identity.AuthenticationType
    Dim isAdmin As Boolean =
    ↵User.IsInRole.__unknown("Domain\Administrators") '
    Dim s As StringBuilder
    s.Append("Hello " & name & ", ")
    If isAuthenticated Then
        s.Append("authenticated")
    Else
        s.Append("not authenticated")
    End If
    s.Append(" using " & identType & ", ")
    s.Append("you are ")
    If isAdmin Then
        s.Append("an admin")
    Else
        s.Append("not an admin")
    End If
    Return s.ToString()
End Function 'HelloWorld
```

## Using GenericIdentity and GenericPrincipal Objects

### ■ Creating and initializing a GenericIdentity object

```
Dim MyIdentity As GenericIdentity MyIdentity =  
    New GenericIdentity("User1")
```

### ■ Creating and initializing a GenericPrincipal object

```
Dim MyStringArray As String() = {"Manager", "Teller"}  
Dim MyPrincipal As  
    New GenericPrincipal(MyIdentity,  
    MyStringArray)
```

### ■ Saving the current principal

```
Dim save As GenericPrincipal = Thread.CurrentPrincipal
```

C#

Visual Basic

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

You can use the **GenericIdentity** class in conjunction with the **GenericPrincipal** class to implement role-based security that is independent of the Windows security system. For example, you can use method parameters representing a user's name and password, verify them against a database entry, and then create identity and principal objects that are based on the values in the database.

Typically, applications that use **GenericPrincipal** objects attach the created **GenericPrincipal** to the current thread by setting the **Thread.CurrentPrincipal** property. This makes the principal object readily available to the application for subsequent role-based security checks, and provides access to this object to any other assemblies that the application might call on the thread. Attaching the principal object also allows your code to use declarative role-based security checks and security checks using **PrincipalPermission** objects.

### Using the GenericPrincipal class in XML Web services

However, in ASP.NET Web Services, a single thread is shared by many Web sessions, and consequently by many different users, each with a unique identity. Therefore, in the case of XML Web services, it only makes sense to attach a single **GenericPrincipal** object to the thread for the duration of the exposed method call. The scenario in which it becomes important to do this is when your XML Web service method calls into an assembly which expects the **Thread.CurrentPrincipal** property to contain current principal information to do role-based security checks.

Further, the **Thread.CurrentPrincipal** property should be reset to its original value before the method returns, to prevent the code, which subsequently uses the thread, from having access to the generic principal identity.

**Implementing role-based security in an XML Web service**

The following procedure outlines the steps to implement role-based security within an XML Web service method by using **GenericIdentity** and **GenericPrincipal** objects.

To implement role-based security in an XML Web service method by using the **GenericIdentity** and **GenericPrincipal** objects:

1. Create a new instance of the **GenericIdentity** class and initialize it with a name that you want it to hold.
2. Create a new instance of the **GenericPrincipal** class and initialize it with the **GenericIdentity** object that is created in the preceding step, and an array of strings that represent the roles that you want to associate with this principal.
3. Save the principal that is attached to the current thread.
4. Attach the principal that you created in step 2 to the current thread.
5. Execute code that requires a current principal context.
6. Reset the principal to the original value.

# Authentication and Authorization with HttpModules

- **HttpApplication events and HttpModules**
- **Authentication using HttpModules**
- **Authorization after authentication**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

For every ASP.NET application, an instance of a class of type **HttpApplication** is created. The **HttpApplication** class exposes a number of events. This topic will focus on the **AuthenticateRequest** event.

## HttpApplication events and HttpModules

You can handle the **AuthenticateRequest** event in `Global.asax`. However, this would mean that each XML Web service that you implement would have to reimplement the authentication code in `Global.asax`.

A better method is to use the **HttpModule** class. An **HttpModule** class implements the **IHttpModule** interface. You can configure **HttpModules** to receive **HttpApplication** events. As a result, you can implement all of your custom authentication code in an **HttpModule** and reuse the **HttpModule** in whichever ASP.NET application that you desire.



**Authentication using HttpModules**

To perform authentication in an **HttpModule**, the **HttpModule** must subscribe to the **AuthenticateRequest** event. The following code shows how to do this.

**C#**


---

```
public sealed class CustomAuthenticationModule : IHttpModule
{
    public void Init(HttpApplication app)
    {
        app.AuthenticateRequest += new
        ↪EventHandler(this.OnAuthenticate);
        // Other initialization
    }
}
```

**Visual Basic .NET**


---

```
Public NotInheritable Class CustomAuthenticationModule
    Implements IHttpModule

    Sub Init(ByVal app As HttpApplication) Implements
    ↪IHttpModule.Init
        AddHandler app.AuthenticateRequest, AddressOf
    ↪Me.OnAuthenticate
        ' other initialization
    End Sub
```

To invoke your **HttpModule** for your XML Web service, the Web.config file must contain the correct configuration information. The following is an example of the required entries in Web.config.

```
<configuration>
  <system.web>
    <httpModules>
      <add name="CustomAuthn"
        type="WSHttpModule.CustomAuthenticationModule, WSHttpModule"
        />
    </httpModules>
  </system.web>
</configuration>
```

Finally you must handle the **AuthenticateRequest** event to set the principal for the current Web operation invocation. The following code gives an example of how this can be done.

C#

---

```
public void OnAuthenticate(Object src, EventArgs e)
{
    HttpApplication app = (HttpApplication)src;
    HttpContext ctx = app.Context;
    string soapUser;
    string soapPassword;
    XmlDocument dom = new XmlDocument();
    Stream httpStream = context.Request.InputStream;
    // Save the current position of stream.
    long posStream = httpStream.Position;
    try
    {
        dom.Load(httpStream);
        httpStream.Seek(posStream, System.IO.SeekOrigin.Begin);
        dom.Save(httpStream);
        // Bind to the Authentication header.
        soapUser =
        ↪dom.GetElementsByTagName("User").Item(0).InnerText;
        soapPassword =
        ↪dom.GetElementsByTagName("Password").Item(0).InnerText;
        // perform check for roles
        string [ ] roles;
        roles = GetRolesForUser(soapUser, soapPassword);
        ctx.User = new GenericPrincipal(new
        ↪GenericIdentity(soapUser,
            "MyAuthType"), roles);
    }
    catch (Exception e)
    {
        // Reset the position of stream.
        httpStream.Position = posStream;
        // Throw a SOAP exception.
        XmlQualifiedName name = new XmlQualifiedName("Load");
        SoapException soapException = new SoapException(
            "Unable to read SOAP request", name, e);
        throw soapException;
    }
}
```

**Visual Basic .NET**


---

```

Public Sub OnAuthenticate(src As Object, e As EventArgs)
    Dim app As HttpApplication = CType(src, HttpApplication)
    Dim ctx As HttpContext = app.Context
    Dim soapUser As String
    Dim soapPassword As String
    Dim dom As New XmlDocument()
    Dim httpStream As Stream = context.Request.InputStream
    ' Save the current position of stream.
    Dim posStream As Long = httpStream.Position
    Try
        dom.Load(httpStream)
        httpStream.Seek(posStream, System.IO.SeekOrigin.Begin)
        dom.Save(httpStream)
        ' Bind to the Authentication header.
        soapUser =
        ↪dom.GetElementsByTagName("User").Item(0).InnerText
        soapPassword =
        ↪dom.GetElementsByTagName("Password").Item(0).InnerText
        ' perform check for roles
        Dim roles() As String
        roles = GetRolesForUser(soapUser, soapPassword)
        ctx.User = New GenericPrincipal(New
        ↪GenericIdentity(soapUser, ↪
            "MyAuthType"), roles)
    Catch e As Exception
        ' Reset the position of stream.
        httpStream.Position = posStream
        ' Throw a SOAP exception.
        Dim name As New XmlQualifiedName("Load")
        Dim soapException As New SoapException("Unable to read
        ↪SOAP request", _
            name, e)
        Throw soapException
    End Try
End Sub 'OnAuthenticate

```

**Authorization after authentication**

After **HttpModule** has performed authentication, you can use .NET role-based security within the implementation of your XML Web service methods. The following code is an example of how this can be done.

**C#**

---

```
[WebMethod]
[SoapHeader("authentication", Required=false)]
public string ValidUser()
{
    if (!User.Identity.IsAuthenticated)
    {
        XmlQualifiedName name = new
        ↪ XmlQualifiedName("AuthError");
        SoapException soapException = new SoapException(
            "Request denied", name);
        throw soapException;
    }
    if (User.IsInRole("Customer"))
        return string.Format("{0} is a
        ↪ customer", User.Identity.Name);
    if (User.IsInRole("Admin"))
        return string.Format("{0} is an
        ↪ administrator", User.Identity.Name);
    return string.Format("{0} is a valid
    ↪ user", User.Identity.Name);
}
```

**Visual Basic .NET**

---

```
Public<WebMethod(), SoapHeader("authentication", Required :=
False)> _
Function ValidUser() As String
    If Not User.Identity.IsAuthenticated Then
        Dim name As New XmlQualifiedName("AuthError")
        Dim soapException As New SoapException("Request denied",
        ↪ name)
        Throw soapException
    End If
    If User.IsInRole("Customer") Then
        Return String.Format("{0} is a customer",
        ↪ User.Identity.Name)
    End If
    If User.IsInRole("Admin") Then
        Return String.Format("{0} is an administrator",
        ↪ User.Identity.Name)
    End If
    Return String.Format("{0} is a valid user",
    ↪ User.Identity.Name)
End Function 'ValidUser
```

## Authorization: Code Access Security

- Code Access Security Fundamentals
- Code Access Security in ASP.NET XML Web Services

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

The common language runtime includes an extensible code access security model. The goal of code access security is to prevent code from accessing protected resources that it has no right to access (such as undesired access to files or the registry). To fulfill this goal, the policy system for code access security grants code-specific permissions based on evidence, that is, characteristics of the code. The code access security model evaluates evidence and grants permissions at assembly level, and not at an application level.

This section introduces you to the concepts of code access security. Also, you will look at how the code access security mechanism determines what permissions to grant an assembly. You will review two scenarios where code access security affects XML Web service deployment.

## Code Access Security Fundamentals

- Evidence-based security
- Code access permissions
- Code groups
- Policy levels
- Named permission sets

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

Traditional authorization mechanisms authorize users based on their logon credentials (usually a password) and restrict resources (often folders and files) that a user is allowed to access. Code that executes on behalf of a user has the same permissions as that of the user. However, this approach to security is insufficient because it requires that every piece of code must be completely trusted before it is permitted to run.

There are many reasons why a code should not be trusted. A code might have bugs or vulnerabilities that can be exploited by another code that is malicious (example: a virus). A code might perform actions that a user is unaware of. As a result, computer systems can be damaged and private data can be compromised if users run malicious or buggy software.

The solution to these security problems is to provide a mechanism that allows trusted users to safely execute untrusted code, and to prevent trusted code from accidentally or intentionally compromising security. Code access security is one solution to these security problems.

### How does code access security work?

Rights to access resources are known as permissions. These permissions are typically organized into named permission sets. The named permission sets are associated with a code access group.

Before code is allowed to execute, the assembly in which the code resides is verified for membership in a specific code group. If the membership criterion is met, then the permissions associated with the code access group are granted to the assembly.

**Evidence-based security** *Evidence* is a set of information about the identity and origin of an assembly. Evidence might include any of the following:

- The assembly's strong name, consisting of a unique public key, a simple name, and a version.
- The assembly's publisher, which is obtained from the Microsoft Authenticode® signature.
- The zone from which the assembly originates, such as a local computer, intranet, or Internet zones.
- The location from which the assembly originates, which can be in the form of a URL, universal naming convention (UNC) path, or a local computer folder.
- The cryptographic hash of the assembly.

When a run-time host loads an assembly, it gathers evidence about the assembly and presents it to the code access security system. The code access security system uses this evidence about the assembly to determine the permissions to grant it based upon an existing security policy. When generating an assembly, you can include custom evidence with it. This evidence is evaluated only if you configure a security policy to use it.

**Code access permissions**

Code access permissions represent rights to access certain computing resources. Some examples of actions that permissions can restrict are reading and writing of files on the file system, accessing environment variables, and making calls to ADO.NET for database access.

The .NET Framework has many built-in code access permission classes that are designed to control access to system resources. For example, the **EnvironmentPermission** class controls access to environment variables; the **FileIOPermission** class controls access to files and folders on the file system; and the **PerformanceCounterPermission** class controls access to performance counters.

In addition to the built-in permission classes, you can add new permissions to the code access security system by implementing custom permissions.

**Code groups**

A code group consists of a membership condition (for example, a membership condition can be code from the Microsoft Web site), and a set of permissions that an assembly might be granted if it meets the membership condition. The runtime evaluates the membership condition that is specified in a code group against the evidence about an assembly. If the assembly meets the condition, it is eligible to receive the permission set that is associated with the code group.

---

**Note** When an assembly meets the membership condition for a code group, it is said to be a member of that code group.

---

A membership condition for a code group can specify that the publisher of an assembly is Microsoft. Therefore, only an assembly that provides evidence that it is published by Microsoft will satisfy the membership condition and be eligible to receive the permission set that is associated with the code group. For this example, the permission set might represent the right to access the C:\Temp directory and the **USERNAME** environment variable.

---

**Note** The membership conditions closely match the evidence available for assemblies, as shown in the preceding example.

---

### Policy levels

Security policy is organized into different policy levels. Each policy level contains a hierarchy of code groups that determine the conditions for permission grants. There are four policy levels:

- *Enterprise-level policy.* The network administrator specifies the enterprise-level policy. This policy contains a code group hierarchy that applies to all of the managed code on the entire network.
- *Machine-level policy.* The administrator of a local computer specifies the machine-level policy. This policy contains a code group hierarchy that applies to all managed code on the computer.
- *User-level policy.* The administrator or a user of a local computer specifies the user-level policy. This policy contains a code group hierarchy that applies to all managed code that a specific user runs.
- *Application domain-level policy.* An application domain host can specify security policy to be applied to a code within an application domain, provided that that host has been granted permission from the **SecurityPermission** class. You can set application domain-level policy only once for any application domain. After an application domain policy is set, further attempts to set this policy fail.

In each security policy level, code groups are organized hierarchically. The root of the hierarchy is a code group that matches all of the code. This root code group has child code groups, and in turn those code groups can have children, and so on. When an assembly is loaded, the common language runtime checks the membership condition for a code group. If the evidence that the assembly provides satisfies the membership condition of a parent code group, then each child code group of that parent will be evaluated against the assembly to see if their membership conditions satisfy the assembly. However, if an assembly does not meet the membership condition for a parent code group, the conditions of the descendants of that group will not be checked. Therefore, each level of children in the hierarchy implies an **AND** condition with its parent in the hierarchy.

When an assembly is loaded, the membership conditions for each code group within each policy level are verified against the evidence of an assembly. For most code groups, the union of the permission sets for the code groups that have their conditions met is computed. This union of permission sets represents the assembly's permission set for that policy level. The intersection of these permission sets across all the policy levels determines the final permission set that an assembly receives.

### Named permission sets

An administrator can associate a set of permissions with a code group by using a named permission set. A named permission set must consist of at least one permission; must have a name; and must have a description. You can associate more than one code group with a named permission set.

The common language runtime provides the following built-in named permission sets: **Nothing**, **Execution**, **Internet**, **LocalIntranet**, **Everything**, and **FullTrust**. You can also create custom-named permission sets.



## Code Access Security in ASP.NET XML Web Services

- Local deployment vs. ISP deployment
- Identifying permissions required by your code
- Requesting permissions in your code
  - RequestMinimum
  - RequestOptional
  - RequestRefused

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Having looked at the concepts of code access security, this topic will examine how you can use it in ASP.NET Web Services.

### Local deployment vs. ISP deployment

ASP.NET Web Service assemblies execute as local applications. When the code access security system interrogates an ASP.NET Web Service assembly, it presents the evidence that **Zone** is equal to **My Computer**. With the default .NET Framework installation, this evidence matches the membership condition of the built-in code group named **My\_Computer\_Zone**. The **My\_Computer\_Zone** code group receives the built-in permission set named **FullTrust**. The **FullTrust** permission set provides full access to all of the resources that the permissions protect.

If your XML Web service has been granted the **FullTrust** permission, the code access security system cannot impose any security restrictions when the default security policy is in use. It is unlikely that your XML Web service will be granted the **FullTrust** permission except in a development environment, or if the XML Web service is deployed in an environment where you have full control over administering the security policy.

However, you might create XML Web services that are deployed at locations, such as an ISP, where you do not have control over administering the security policy. An ISP might implement a more restrictive security policy, which could be problematic for your XML Web services. Most importantly, you must understand the permissions that your XML Web service code requires.

### Identifying permissions required by your code

You can identify the permissions that your code requires by using several different approaches.

The .NET Framework SDK documentation lists the permissions that are required to use each of the .NET Framework classes. By referring to the documentation for the classes that are used in your XML Web service, you can have a good idea of the permissions that your XML Web service code will need.

Also, a familiarity with the built-in permission classes will help in determining the permissions that your code requires. For example, if you know that a **MessageQueue** permission exists, you can predict that your code will need this permission to use the Microsoft Message Queue libraries.

Another means of determining the permissions that your code will need is to test it with a limited set, such as the **Internet** permission set, and see the security exceptions that the code throws.

After determining the permissions that your assembly needs, you can negotiate with ISP administrators for a more liberal security policy specifically for your assembly.

### Requesting permissions in your code

It is possible to specify permission requests for your assembly by using the **SecurityPermission** attribute. This attribute is stored in the metadata of your assembly. The assembly permission requests are examined when an assembly is loaded. The .NET Framework proceeds based on the kind of permission request the assembly makes. The three kinds of permission requests are:

- **Minimum permissions (RequestMinimum)**

The permissions in these requests represent the minimum set of permissions that an assembly needs to work effectively. If these permissions are not available to an assembly when it is loaded, the .NET Framework will not execute the code in that assembly and returns an exception to the caller. A minimum permission request for an assembly documents the required permissions for that assembly. Not making a minimum permission request for an assembly is the equivalent of making a minimum permission request of **Nothing**.

- **Optional permissions (RequestOptional)**

The permissions in these requests represent permissions that an assembly can use, but the assembly can still run effectively without these permissions. These permissions are granted to an assembly if they are available to that assembly, but if they are not available, the assembly is still allowed to run. Not making an optional permission request for an assembly is the equivalent of making an optional permission request of **FullTrust**.

- **Refused permissions (RequestRefused)**

The permissions in these requests represent permissions that an assembly can never be granted, even if a security policy allows these permissions to be granted. Not making a refused permission request is the equivalent of making a refused permission request of **Nothing**.

You can request permissions by adding assembly-level attributes to your code. For example, to request a minimum permission set that grants your assembly the ability to execute unmanaged code, you can add the following attribute to your assembly.

**C#**

---

```
[assembly:SecurityPermissionAttribute(SecurityAction.RequestMinimum,
    UnmangedCode = true)]
```

**Visual Basic .NET**

---

```
<assembly:
SecurityPermissionAttribute(SecurityAction.RequestMinimum, ↵
    UnmangedCode := True)>
```

Typically, you add the preceding code to the AssemblyInfo file in your XML Web service project.

By using reflection, an administrator can view the permission set that an assembly minimally requires and set the security policies accordingly.

It is a good practice to annotate your assembly by setting the **SecurityPermission** attribute to a minimum permission set. This can be particularly helpful if you deploy your assembly on a system where you do not have any control over setting the security policy.

# Encryption

- Using SSL
- Using Custom SOAP Extensions
- Code Walkthrough: Implementing a SOAP Extension
- Code Walkthrough: Implementing a Custom Attribute
- Code Walkthrough: Using a Custom Attribute

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Apart from securing XML Web services through authentication and authorization, you also must secure the messages that are exchanged between an XML Web service and its consumer. You can secure messages by encrypting them before they are sent.

In this section, you will look at the advantages and disadvantages of using Secure Socket Layer (SSL) for XML Web service encryption versus using custom SOAP extensions to perform encryption. Also, you will look at the process of enabling connections through SSL. Then, you will examine the mechanics of using encryption to provide secure communication between an XML Web service and its client.

## Using SSL

- What are X.509 certificates?
- How to enable SSL on a Web server

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### What are X.509 certificates?

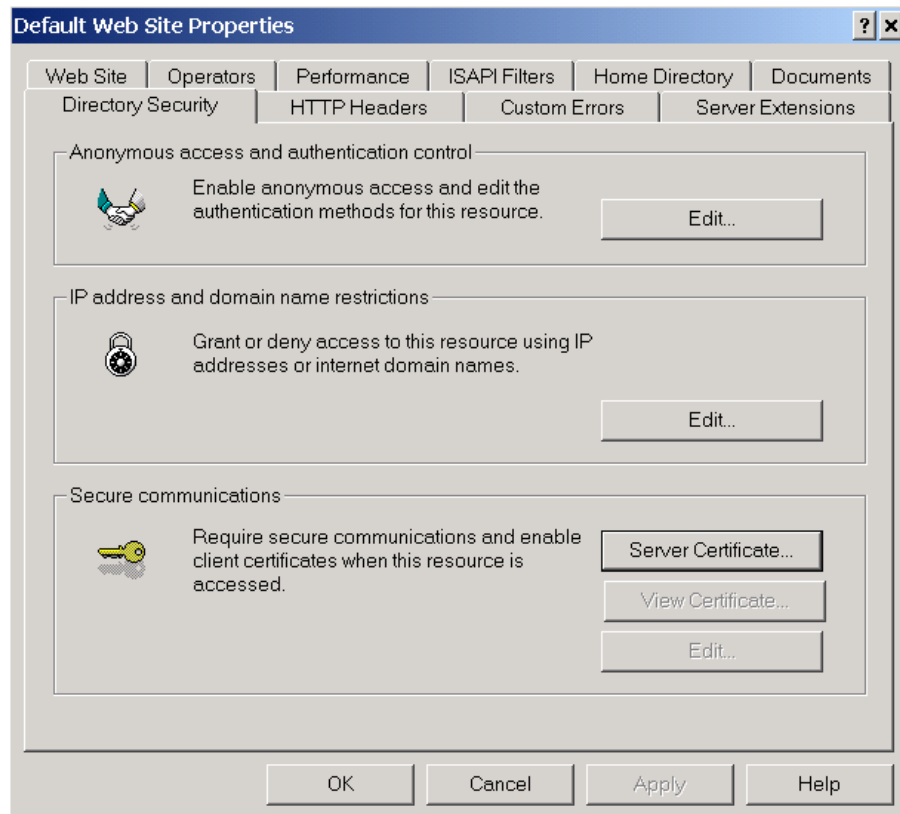
To enable IIS to support Secure Socket Layer (SSL) connections, you must obtain an X.509 certificate and install it on the Web server running IIS.

A certificate is a structure that contains information about its subject, issuer's name, validity period, and other characteristics. Each certificate is related to a pair of private and public keys that are used in SSL encryption. SSL always uses X.509 certificates to authenticate a Web server.

You obtain a certificate by making a request to a Certificate Authority (CA). When a CA issues a certificate to a subject (an entity that made the request), it verifies that the subject is who it claims to be and signs the new certificate with its private key.

**How to enable SSL on a Web server**

After you obtain a certificate, if you want to enable SSL on you Web server, you must install the certificate. You install the certificate from the **Directory Security** tab in the **Properties** dialog box for a Web site or a virtual directory, as shown in the illustration:



Typically, the activities of obtaining a certificate and configuring the Web server for SSL are the responsibility of the network administrator, and not the developer. For this reason, we will not discuss this administrative function further. For more information, see the Knowledge Base article with the title "How to Import a Server Certificate for Use in Internet Information Services 5.0" on Microsoft MSDN®.

## Using Custom SOAP Extensions

- The **DESCryptoServiceProvider** class
- The **CryptoStream** class
- Encryption using SOAP extensions

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

Both HTTP and SOAP are text-based protocols. This makes it easy for an unauthorized entity to see the contents of the messages that are exchanged between an XML Web service consumer and an XML Web service.

The problem with SSL is that any communication is always encrypted. Using a SOAP extension is a viable option, if you want to:

- Encrypt only some of the requests or responses.
- Encrypt only some parts of a request or response.
- Use a solution that does not have node affinity.
- Control the encryption mechanism that is used.

Before using a SOAP extension for encryption, you must be familiar with some of the .NET Framework classes that will help in implementing such an extension.

### The **DESCryptoServiceProvider** class

The **DESCryptoServiceProvider** class provides access to the Cryptographic Service Provider (CSP) version of the Data Encryption Standard (DES) algorithm. Using this encryption service provider, you can create an encryptor, given a key and an initialization vector (IV).

```
encryptor = des.CreateEncryptor( key, IV )
```

You can also create a decryptor by using **DESCryptoServiceProvider**.

```
decryptor = des.CreateDecryptor( key, IV )
```

---

**Note** The **DESCryptoServiceProvider** is just one encryption service provider class within the .NET Cryptographic Services.

---

**The CryptoStream Class**

A **CryptoStream** is a stream that links data streams to cryptographic transformations. By simply writing or reading to this stream, you can encrypt or decrypt data. You can create a **CryptoStream** object after you have an **encryptor** or **decryptor** object. The following code shows how to do this.

**C#**

---

```
ICryptoTransform encryptor;  
ICryptoTransform decryptor;  
encryptor = des.CreateEncryptor( key, IV );  
decryptor = des.CreateDecryptor( key, IV );  
  
CryptoStream cs;  
cs = new CryptoStream(ms, encryptor, CryptoStreamMode.Write);  
...  
cs = new CryptoStream(ms, decryptor, CryptoStreamMode.Read);
```

**Visual Basic .NET**

---

```
Dim encryptor As ICryptoTransform  
Dim decryptor As ICryptoTransform  
encryptor = des.CreateEncryptor(key, IV)  
decryptor = des.CreateDecryptor(key, IV)  
  
Dim cs As CryptoStream  
cs = New CryptoStream(ms, encryptor, CryptoStreamMode.Write)  
...  
cs = New CryptoStream(ms, decryptor, CryptoStreamMode.Read)
```

**Encryption using SOAP Extensions**

The .NET Framework makes it possible to hook into the serializing and deserializing process for SOAP messages. You hook into these processes by implementing:

- A class derived from the **SoapExtension** class, which is found in the **System.Web.Services.Protocols** namespace.
- A custom attribute that references the custom SOAP extension class.

To encrypt and decrypt messages by using SOAP extensions, you must apply the custom attribute that references the SOAP extension to the appropriate XML Web service methods.

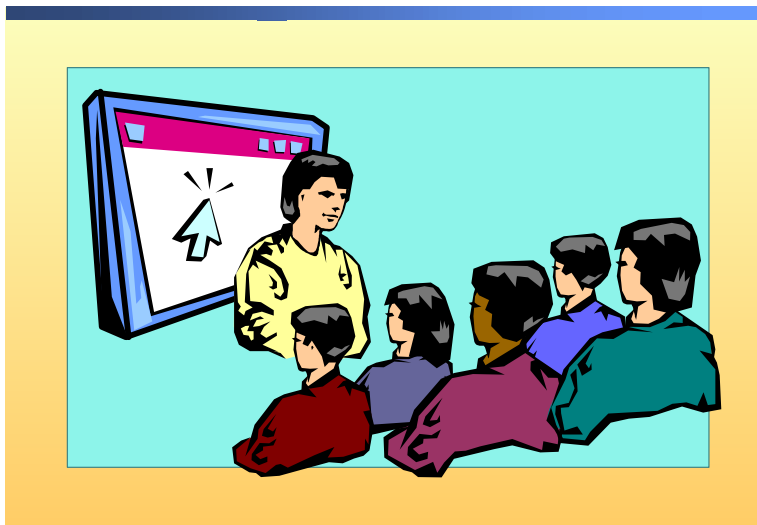
---

**Note** If you are implementing an XML Web service consumer by using the .NET Framework, then you can also apply the custom attribute to the proxy class methods that correspond to the XML Web service methods that have the custom attribute applied.

---



## Code Walkthrough: Implementing a SOAP Extension



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

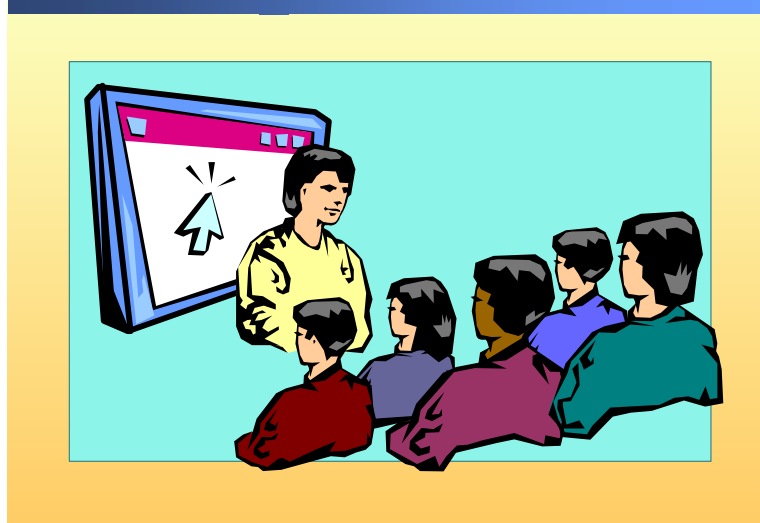
In this walkthrough, you will learn how to implement a SOAP extension named **EncryptExtension**. This SOAP extension will encrypt SOAP headers before they are serialized, and decrypt them after they have deserialized.

---

**Note** You can find the code that is used for this walkthrough in the file *<install folder>\Democode\<language>\Mod07\EncryptionExtension (.cs or .vb)*.

---

## Code Walkthrough: Implementing a Custom Attribute



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

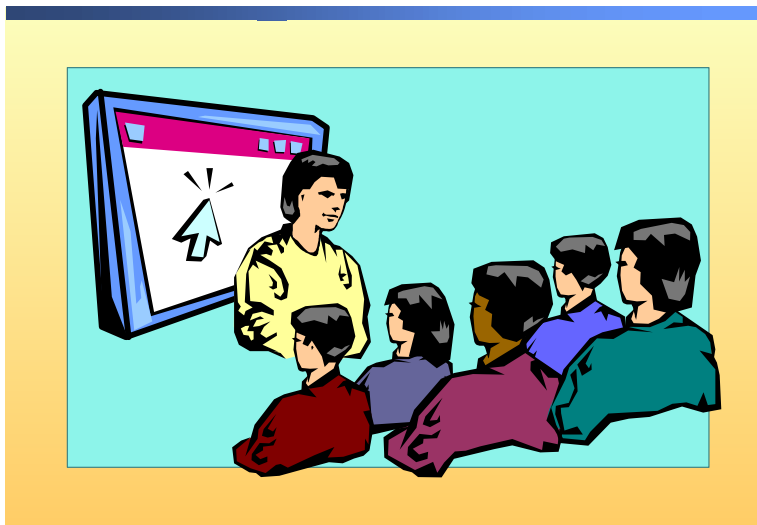
In this walkthrough, you will learn how to implement a custom attribute that is derived from the **SoapExtensionAttribute**. You will use this custom attribute to associate a custom SOAP extension with the methods in an XML Web service and an XML Web service proxy in the next code walkthrough.

---

**Note** You can find the code that is used for this walkthrough in the file *<install folder>\Democode\<language>\Mod07\EncryptionExtensionAttribute (.cs or .vb)*.

---

## Code Walkthrough: Using a Custom Attribute



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

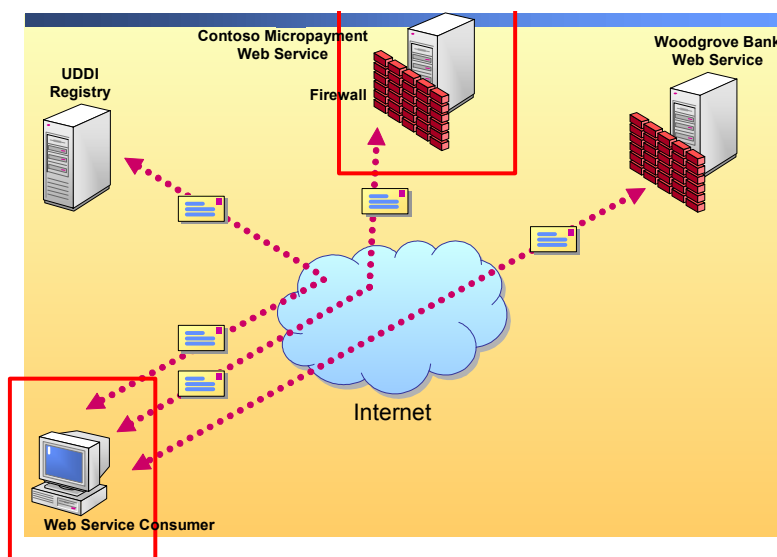
In this walkthrough, you will learn how to apply the custom attribute that was created in the previous walkthrough to the methods of an XML Web service and the corresponding methods of an XML Web service proxy class.

---

**Note** You can find the code that is used for this walkthrough in the file *<install folder>\Democode\<language>\Mod07\UseEncryptAttribute.txt*.

---

## Lab 7.1: Securing XML Web Services



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Objectives

After completing this lab, you will be able to:

- Explain and use role-based security in an XML Web service.
- Use SOAP headers for authentication in an XML Web service.

**Note** This lab focuses on the concepts in this module and as a result may not comply with Microsoft security recommendations. For instance, this lab does not comply with the recommendation that you do not use weak passwords or that you do not hard code passwords into an application.

### Lab Setup

There are starter and solution files that are associated with this lab. The starter files are in the folder `<labroot>\Lab07\Starter`. The solution files are in the folder `<labroot>\Lab07\Solution`.

If you completed Lab 5.1, Implementing a Simple XML Web Service, in Module 5, “Implementing a Simple XML Web Service,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*, skip the following procedures in this section.

#### ► Set up Contoso XML Web Service solution, if you did not complete Lab 5.1

- Copy the `<labroot>\Lab05\Solution\Contoso` folder to `C:\Inetpub\Wwwroot` folder, and overwrite the Contoso project that you created in Lab 5.1.

**► Set up Contoso Web Service solution, if you did not start Lab 5.1**

1. Copy the <labroot>\Lab05\Solution\Contoso folder to the C:\Inetpub\Wwwroot folder.
2. Create the virtual directory for the Contoso XML Web Service:
  - a. Click **Start**, point to **Programs**, point to **Administrative Tools**, and then click **Internet Services Manager**.
  - b. Click the plus sign to expand the tree.
  - c. Right-click **Default Web Site**, point to **New**, and then click **Virtual Directory**.
  - d. Complete the **Virtual Directory Creation Wizard** by using the information in the following table.

On this wizard page	Do this
<b>Welcome to the Virtual Directory Creation Wizard</b>	Click <b>Next</b> .
<b>Virtual Directory Alias</b>	In the <b>Alias</b> box, type <b>Contoso</b> and click <b>Next</b> .
<b>Web Site Content Directory</b>	In the <b>Directory</b> box, type <b>C:\inetpub\wwwroot\Contoso</b> and click <b>Next</b> .
<b>Access Permissions</b>	Select <b>Browse</b> , and click <b>Next</b> .
<b>You have successfully completed the Virtual Directory Creation Wizard.</b>	Click <b>Finish</b> .

**Scenario**

In this lab, you will modify the Contoso Micropayment Service to use SOAP headers to contain authentication information. You will also modify the Contoso XML Web service to encrypt these headers by using a SOAP Extension that is provided to you.

You will also modify the Woodgrove and Contoso Account Manager application to add encrypted SOAP headers to XML Web service method calls to both the Contoso Micropayment Service and the Woodgrove Online Bank XML Web service.

**Estimated time to  
complete this lab: 60  
minutes**

## Exercise 1

### Authenticating Using SOAP Headers

In this exercise, you will add the **SoapHeader** attribute to Contoso Micropayment XML Web service methods to include authentication information in a SOAP header. You will also modify the Woodgrove and Contoso Account Manager to add authentication information to SOAP headers for the **Woodgrove** and **Contoso** XML Web service methods.

► **Add the SoapHeader attribute to Contoso XML Web service methods**

1. In Visual Studio .NET, open the Contoso project that you created in Lab 5.1.

---

**Important** If you did not start or complete Lab 5.1, refer to the Lab Setup section at the beginning of this lab for additional instructions.

---

2. Open the code behind file for Micropayment.asmx.
3. Locate the **GetAccount** method:
4. Add the following attribute to the method, after the **WebMethod** attribute.

C#	Visual Basic .NET
<code>[SoapHeader("authInfo", Required=false)]</code>	<code>&lt;SoapHeader("authInfo", Required := ↪ False)&gt;</code>

Notice that the **SoapHeader** attribute refers to the **authInfo** data member of the **Micropayment** class. The **authInfo** data member is of type **ContosoAuthInfo**. The **ContosoAuthInfo** data type is defined as follows.

C#	Visual Basic .NET
<pre>public class ContosoAuthInfo :     SoapHeader {     public string Username;     public string Password; }</pre>	<pre>Public Class ContosoAuthInfo     Inherits SoapHeader     Public Username As String     Public Password As String End Class 'ContosoAuthInfo</pre>

Also, notice that the **SoapHeader** attribute **Required** property is set to indicate that the SOAP header is optional.

---

**Tip** Defining the SOAP header as optional aids testing, because if the header is not required in an XML Web service method, then the method can support the HTTP GET protocol. If the method supports the HTTP GET protocol, then you will be able to use the built-in test page. However, in a real-world scenario, you would want to require that the SOAP header be present to perform authentication.

---

5. Locate the **GetTransactionHistory** method. Add the same **SoapHeader** authentication information to this method.
6. Build the Contoso Micropayment Service.

► **Add SOAP headers to XML Web service method calls in the Woodgrove and Contoso Account Manager application**

1. In Visual Studio .NET, open the Woodgrove and Contoso Account Manager project that you worked on in Lab 5.1.

---

**Important** If you did not start or complete Lab 5.1, then use the Woodgrove and Contoso Account Manager project found in the folder `<labroot>\Lab07\Starter`. Otherwise, continue to use the Woodgrove and Contoso Account Manager project that you worked on in Lab 5.1.

---

2. Open the following file and examine the code.

C#	Visual Basic .NET
WebServiceClientForm.cs	WebServiceClientForm.vb

3. In Solution Explorer, under the **Web References** category, right-click **Micropayment Web reference**.
4. On the shortcut menu, click **Update Web Reference** to rebuild the Contoso client proxy class.

---

**Note** The Web reference must be updated. This allows the proxy class generator to detect that the XML Web service methods **GetAccount** and **GetTransactionHistory** now support a SOAP header. The proxy class **GetAccount** and **GetTransactionHistory** will be recreated to also support SOAP headers.

---

5. Open the **Class View** and expand the **WebServiceClient.Micropayment** namespace.

Notice that the updated proxy namespace contains a definition for the **ContosoAuthInfo** class, which is the class that is referenced by the **SoapHeader** attribute that you applied to the **GetAccount** and **GetTransactionHistory** XML Web service methods.

6. Locate the **buttonContosoGetTransactions\_Click** method.
7. Create an instance of the **ContosoAuthInfo** class.

Notice that the **Form1** class defines the following private data members.

C#	Visual Basic .NET
private string contosoUserID = "John";	Private contosoUserID As String = "John"
private string contosoPassword =	Private contosoPassword As String =
↵ "password";	↵ "password"
private string woodgroveUserID = "John";	Private woodgroveUserID As String =
private string woodgrovePassword =	↵ "John"
↵ "password";	Private woodgrovePassword As String =
	↵ "password"

---

**Important** The code for steps 7 through 10 must be inserted before the **GetTransactionHistory** method call.

---

8. Assign the **ContosoAuthInfo.Username** property the value of **contosoUserID**.

9. Assign the **ContosoAuthInfo.Password** property the value of **contosoPassword**.
10. Assign the **ContosoAuthInfo** object to the **ContosoMicropaymentService.ContosoAuthInfoValue** property.
11. Repeat steps 7 through 10 in the **GetContosoAccountInfo** method. Ensure that you insert the code for steps 6 through 9 before the invocation of the **GetAccount** method.
12. Locate the **buttonWoodgroveGetTransactions\_Click** method.
13. Create an instance of the **WoodgroveAuthInfo** class.

---

**Important** The code for steps 13 through 16 must be inserted before the **GetTransactionHistory** method call.

---

14. Assign the **WoodgroveAuthInfo.Username** property the value of **woodgroveUserID**.
15. Assign the **WoodgroveAuthInfo.Password** property the value of **woodgrovePassword**.
16. Assign the **woodgrovePassword** object to the **WoodgroveOnlineBank.WoodgroveAuthInfoValue** property.
17. Repeat steps 13 through 16 in the **GetWoodgroveAccountInfo** method. Ensure that you insert the code for these steps before the invocation of the **GetAccount** method.
18. Repeat steps 13 through 16 in the **GetWoodgroveAccountList** method. Ensure that you insert the code for these steps before the invocation of the **GetAllAccounts** method.

► **Enable logon in the Woodgrove and Contoso Account Manager application**

1. Assign the **menuItem1.Enabled** property the following value.

C#	Visual Basic .NET
<b>true</b>	<b>True</b>

This enables the **Set Woodgrove Logon** and **Set Contoso Logon** menu items on the **Logon** menu, allowing a user to select different Woodgrove and Contoso accounts to test the authentication logic.

The implementations of the **menuItemWoodgroveLogon\_Click** and **menuItemContosoLogon\_Click** event handlers are already provided to you.



2. Modify the existing code to prompt the user for Woodgrove and Contoso authentication information. To do this:
  - a. Remove the following lines of code at the beginning of the **Form1\_Load** event handler method.

**C#****Visual Basic .NET**

---

this.GetWoodgroveAccountList();	Me.GetWoodgroveAccountList()
this.GetContosoAccountInfo();	Me.GetContosoAccountInfo()

---

- b. Remove the comments from the following code in the **Form1\_Load** event handler code.

**C#**

---

```
if (GetLogonInfo("Woodgrove", ref woodgroveUserID, ref woodgrovePassword))
{
    this.GetWoodgroveAccountList();
}

if (GetLogonInfo("Contoso", ref contosoUserID, ref contosoPassword))
{
    this.GetContosoAccountInfo();
}
```

**Visual Basic .NET**

---

```
If GetLogonInfo("Woodgrove", woodgroveUserID, woodgrovePassword) Then
    Me.GetWoodgroveAccountList()
End If

If GetLogonInfo("Contoso", contosoUserID, contosoPassword) Then
    Me.GetContosoAccountInfo()
End If
```

3. Build the application.

► **Test the SOAP header authentication**

1. Enter different Contoso and Woodgrove account authentication information when prompted by the **LogonForm** dialog box when the application loads. Or, reset the Woodgrove or Contoso authentication information from the **Logon** menu.

- a. Use the following valid Woodgrove user IDs to verify that the Web method authentication takes place.

UserID	Password
John	Password
Jane	Password

- b. Similarly, use the following valid Contoso user IDs to verify that authentication takes place.

UserID	Password
John	Password
Jane	Password

- c. To invoke the **Woodgrove** and **Contoso** XML Web service methods that use SOAP header authentication, click **Contoso Update Account Info** and **Get Transaction History**.
  - d. Enter invalid user IDs or passwords for both Woodgrove and Contoso. The XML Web service methods will cause an exception to be thrown.

## Exercise 2

### Encrypting Using the SOAP Encryption Extension

In this exercise, you will add functionality to encrypt SOAP authentication headers that are sent from the client to the Contoso XML Web service and decrypt the headers that the XML Web service receives.

#### ► Encrypt SOAP headers sent from the Woodgrove and Contoso Account Manager to the Contoso XML Web service

1. Open the Woodgrove and Contoso Account Manager project.
2. Add a reference to the provided **EncryptionExtension** assembly.
  - a. On the **Project** menu, click **Add Reference**.
  - b. In the **Add Reference** dialog box, on the **.NET** tab, click **Browse**.
  - c. Browse to <labroot>\Lab07\Starter\EncryptionExtension\Bin\Debug.
  - d. Click **EncryptionExtension.dll**.
  - e. Click **OK** to add the reference.
3. Import the **EncryptionExtension** namespace.
4. Open **Class View**.
5. Expand the **ContosoMicropaymentService** class in the **WebServiceClient.Micropayment** namespace.
6. Double-click the **GetAccount** method to open the proxy class implementation for this method.
7. Add the following method attribute.

C#

---

```
[EncryptionExtension(Encrypt=EncryptMode.Request,
SOAPTarget=Target.Header)]
```

Visual Basic .NET

---

```
<EncryptionExtension(Encrypt := EncryptMode.Request,
SOAPTarget := Target.Header)> _
```

The **Encrypt** and **Target** properties specify that the encryption target is the SOAP header, which must be encrypted when an XML Web service method request is sent.

Note that the **EncryptionExtension** attribute must be applied after the **WebMethod** attribute. For example, the following code shows the **EncryptionExtension** attribute that is applied after the **WebMethod** attribute to the **GetAccount** XML Web service method.

C#

---

```
[WebMethod]
[SoapHeader("authInfo", Required=false)]
[EncryptionExtension(Decrypt=DecryptMode.Request,
↪SOAPTarget=Target.Header)]
public AccountDataSet GetAccount()
```

**Visual Basic .NET**

---

```
<WebMethod(), _  
SoapHeader("authInfo", Required := False), _  
EncryptionExtension(Decrypt := DecryptMode.Request,  
↪SOAPTarget := Target.Header)> _  
Public Function GetAccount() As AccountDataSet '
```

8. Add the same attribute to the **GetTransactionHistory** method of the **ContosoMicropaymentService** proxy class.
9. Build the application.

**► Decrypt the SOAP headers that the Contoso XML Web service receives**

1. Open the Contoso Micropayment XML Web service project.
2. Add a reference to the provided **EncryptionExtension** assembly.
  - a. On the **Project** menu, click **Add Reference**.
  - b. In the **Add Reference** dialog box, on the **.NET** tab, click **Browse**.
  - c. Browse to <labroot>\Lab07\Starter\EncryptionExtension\Bin\Debug.
  - d. Click EncryptionExtension.dll.
  - e. Click **Open**.
  - f. Click **OK** to add the reference.
3. Open the code behind file for Micropayment.asmx.
4. Add the following method attribute to the **GetAccount** and **GetTransactionHistory** XML Web service methods:

**C#**

---

```
[EncryptionExtension(Decrypt=DecryptMode.Request,  
↪SOAPTarget=Target.Header)]
```

**Visual Basic .NET**

---

```
<EncryptionExtension(Decrypt := DecryptMode.Request,  
↪SOAPTarget := Target.Header)> _
```

The **Decrypt** and **Target** properties specify that the decryption target is the SOAP header, which must be decrypted when an XML Web service method request is received.

5. Build the application.
6. Run the Woodgrove and Contoso Account Manager application.
7. To test the Contoso Micropayment Service functionality, click **Contoso Update Account Info** and **Get Transaction History**.

## Exercise 3

### Using Role-Based Security

In this exercise, you will add a new method named **DeleteAccount** to the Contoso Micropayment Service. Only Woodgrove administrators will be authorized to delete a user account by using this method. The **DeleteAccount** method will authenticate against the database and use role-based security to authorize access to the code that deletes the account.

The **DeleteAccount** method will call the **\_AuthenticateAdmin** stored procedure to authenticate an administrator against a table in the Contoso database. This stored procedure returns a set of application-defined roles that are associated with the authenticated administrator. You will use these roles to create a **GenericPrincipal** object and implement role-based security. You will then associate this **GenericPrincipal** object with the current thread and call a method in an external assembly, named ContosoAdminDB.dll, to actually delete the account. The method in the external assembly will test that the user associated with the current thread has the correct role membership authorized to delete an account.

---

**Note** The ContosoAdminDB.dll provided as a part of the starter files will not actually delete the accounts.

---

#### ► Add the DeleteAccount method

1. Open the Contoso Micropayment Service project.
2. Add a reference to the assembly at <labroot>\Lab07\Starter\ContosoAdminDB\Bin\Debug\ContosoAdminDB.dll.
3. Open the code behind file for Micropayment.asmx.
4. Add an XML Web service method with the following signature to delete an account.

C#

---

```
public bool DeleteAccount(string AdminUserName, string
    ↪Password, int AccountID)
```

Visual Basic .NET

---

```
Public Function DeleteAccount(AdminUserName As String,
    ↪Password As String, _
    AccountID As Integer) As Boolean
```

Note that this method must not include a SOAP header or encryption attribute. Remember to expose this method by adding the **WebMethod** attribute.

### ► Add a new database connection to Server Explorer

1. Open Server Explorer.
2. Right-click **Data Connections** and click **Add Connection**.
3. Complete the **Data Link Properties** by using the information in the following table.

On this wizard page	Do this
<b>Connection tab of the Data Link Properties Dialog box</b>	<p>For the numbered fields, type the following values:</p> <ol style="list-style-type: none"> <li>1. <i>The name of your computer</i>\MOC</li> <li>2. User name: <b>sa</b> Password: <b>Course_2524</b> Select the <b>Allow saving password</b> check box.</li> <li>3. <b>Contoso</b></li> </ol> <p>To verify that the connection information is correct, click <b>Test Connection</b>.</p>
<b>Microsoft Datalink</b>	Click <b>OK</b> .

4. A dialog box will be displayed warning that your connection information is not encrypted. Click **OK**. Generally, this is not a safe practice, but in the classroom it is convenient.

### ► Add a RolesDataSet typed dataset

1. In Solution Explorer, right-click the **Contoso** project and click **Add** and then **Add New Item** on the shortcut menu.
2. From the list of available templates, click **Data Set**.
3. In the **Name** field, rename the file to **RolesDataSet.xsd**.

### ► Generate a typed RolesDataSet

1. Expand the Stored Procedures node under the newly added connection in the Server Explorer.
2. Click the **\_AuthenticateAdmin** stored procedure and drag it to the designer surface for **RolesDataSet.xsd**.

### ► Complete the DeleteAccount method

1. Open the code behind file for Micropayment.asmx.
2. Locate the **DeleteAccount** method.
3. Add and instantiate local variables for a **SqlCommand**, **SqlConnection**, and **SqlDataAdapter**.

4. Initialize the **SqlCommand** object as shown in the following example.

#### C#

---

```
cmd.CommandText = "_ AuthenticateAdmin ";
cmd.CommandType = CommandType.StoredProcedure;
cmd.Connection = conn;
cmd.Parameters.Add(new SqlParameter("@RETURN_VALUE", SqlDbType.Int, 4,
ParameterDirection.ReturnValue, true, ((Byte)(10)), ((Byte)(0)), "",
DataRowVersion.Current, null));
cmd.Parameters.Add(new SqlParameter("@AdminUserName", SqlDbType.NVarChar, 16,
ParameterDirection.Input, true, ((Byte)(10)), ((Byte)(0)), "",
DataRowVersion.Current, AdminUserName));
cmd.Parameters.Add(new SqlParameter("@AdminPassword ", SqlDbType.NVarChar, 16,
ParameterDirection.Input, true, ((Byte)(10)), ((Byte)(0)), "",
DataRowVersion.Current, Password));
```

#### Visual Basic .NET

---

With cmd

```
.CommandText = "_AuthenticateAdmin "
.CommandType = CommandType.StoredProcedure
.Connection = conn
.Parameters.Add(New SqlParameter("@RETURN_VALUE", SqlDbType.Int, 4,
➤ParameterDirection.ReturnValue, True, CType(10,Byte), CType(0,Byte), "",
➤DataRowVersion.Current, Nothing))
.Parameters.Add(New SqlParameter("@AdminUserName", SqlDbType.NVarChar, 16,
➤ParameterDirection.Input, True, CType(10,Byte), CType(0,Byte), "",
DataRowVersion.Current, AdminUserName))
.Parameters.Add(New SqlParameter("@AdminPassword", SqlDbType.NVarChar, 16,
➤ParameterDirection.Input, True, CType(10, System.Byte), CType(0, System.Byte), "",
➤DataRowVersion.Current, Password))
```

End With

5. Initialize the **SqlConnection** object as shown in the following example.

#### C#

---

```
conn.ConnectionString =
➤(string)ConfigurationSettings.AppSettings["connectStringContoso"];
conn.Open();
```

#### Visual Basic .NET

---

```
conn.ConnectionString = ConfigurationSettings.AppSettings("connectStringContoso")
conn.Open()
```

6. Initialize the **SqlAdapter** as shown in the following example.

#### C#

---

```
adapter.SelectCommand = cmd;
```

#### Visual Basic .NET

---

```
adapter.SelectCommand = cmd
```

7. Create an instance of the **RolesDataSet** typed dataset, using the **SqlDataAdapter** object to fill it. The following code is an example of how to do this.

**C#**

---

```
RolesDataSet ds = new RolesDataSet ();
int nRecords = adapter.Fill(ds,"_AuthenticateAdmin ");
conn.Close();
if (nRecords == 0)
    return false;
return ds;
```

**Visual Basic .NET**

---

```
Dim ds As New RolesDataSet ()
Dim nRecords As Integer = adapter.Fill(ds,
↳ "_AuthenticateAdmin ")
conn.Close()
If nRecords = 0 Then
    Return False
End If
Return ds
```

8. If the returned dataset contains records, allocate an array of strings that is the length of the numbers of records.
9. Loop through the **RolesDataSet** and add an element in the string array for each role in the **RolesDataSet**.

The role is specified by the **Role** property of the item in the **RolesDataSet.AuthenticateAdmin** collection. The following code is an example of how to store the list of roles in a string array

**C#**

---

```
// Create array of string with roles
string [] roles;
roles = new string[nRecords];
for (int i = 0; i < nRecords; i++)
{
    roles[i] = ds._AuthenticateAdmin[i].Role;
}
```

**Visual Basic .NET**

---

```
' Create array of string with roles
Dim roles() as String
roles = New String(nRecords-1){}
Dim i As Integer
For i = 0 To nRecords - 1
    roles(i) = ds._AuthenticateAdmin(i).Role
Next i
```



**► Use role-based security**

1. Import the System.Security.Principal namespace. This namespace defines the GenericIdentity and GenericPrincipal objects.
2. Create a GenericIdentity object by passing the AdminUserName parameter to the GenericIdentity constructor.
3. Create a GenericPrincipal object by passing the GenericIdentity object and the string array of roles to the GenericPrincipal constructor.
4. Save the current thread principal to a temporary IPPrincipal reference.
5. Set the current thread principal to the GenericPrincipal object.
6. Call the static method AdminUtil.DeleteAccount from the ContosoAdminDB.dll assembly passing a string representation of AccountID.
7. The AdminUtil.DeleteAccount method requires the caller to be a member of the application-defined Supervisor role. If the caller is not a member of this role, an exception is thrown.
8. Enclose the code added in steps 5 and 6 in a try-catch block.
9. Catch any exceptions that the AdminUtil.DeleteAccount method throws. If an exception is caught, return the following value.

C#	Visual Basic .NET
false	False

10. If there is no exception, return the following value.

C#	Visual Basic .NET
true	True

11. Add a **finally** clause that restores the current thread principal to the saved reference.

The following code is an example of how to perform these steps.

**C#**

---

```
GenericIdentity ident = new GenericIdentity(AdminUserName);
GenericPrincipal principal = new GenericPrincipal(ident,
    ↪roles);
```

```
IPrincipal principalSave =
    ↪System.Threading.Thread.CurrentPrincipal;
```

```
try
{
    System.Threading.Thread.CurrentPrincipal = principal;
    AdminUtil.DeleteAccount(AccountID.ToString());
    return true;
}
catch(Exception e)
{
    return false;
}
finally
{
    System.Threading.Thread.CurrentPrincipal =
principalSave;
}
```

**Visual Basic .NET**

---

```
Dim ident As New GenericIdentity(AdminUserName)
Dim principal As New GenericPrincipal(ident, roles)
```

```
Dim principalSave As IPrincipal =
System.Threading.Thread.CurrentPrincipal
```

```
Try
    System.Threading.Thread.CurrentPrincipal = principal
    AdminUtil.DeleteAccount(AccountID.ToString())
    Return True
Catch e As Exception
    Return False
Finally
    System.Threading.Thread.CurrentPrincipal = principalSave
End Try
```

12. Build the Contoso Micropayment Service.

**► Test the DeleteAccount method**

1. Open the test page <http://localhost/Contoso/Micropayment.asmx>.
2. Click the **DeleteAccount** method to navigate to the DeleteAccount Service Method Help page.

The following table shows the administrators and roles included in the Contoso database.

Administrator User Name	Password	Role(s)
Kelly	password	Supervisor, SuperUser, ServiceRep
Sue	password	ServiceRep
Bob	password	ServiceRep

3. Invoke **DeleteAccount** for Kelly, setting the AccountID parameter to **1**.  
Verify that the XML Web service method returns **true** because Kelly is a member of the Supervisor role.
4. Invoke **DeleteAccount** for Bob or Sue, setting the AccountID parameter to **1**.

Verify that the XML Web service method returns **false** because these administrators are not members of the Supervisor role.

## Review

- Overview of Security
- Built-In Authentication
- Custom Authentication: SOAP Headers
- Authorization: Role-Based Security
- Authorization: Code Access Security
- Encryption

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

1. Which one of the following occurs first: authentication or authorization?  
**authentication**
2. What are the three authentication mechanisms that IIS provides?  
**Basic, Digest, and Integrated Windows authentication**
3. If the users of your XML Web service do not have Windows accounts, how can you provide authentication information to the XML Web service?  
**By defining a SOAP header representing the required information and requiring that a SOAP header be used in your XML Web service.**
4. When implementing role-based security, which type of object stores a list of roles for an authenticated user?  
**Principal**

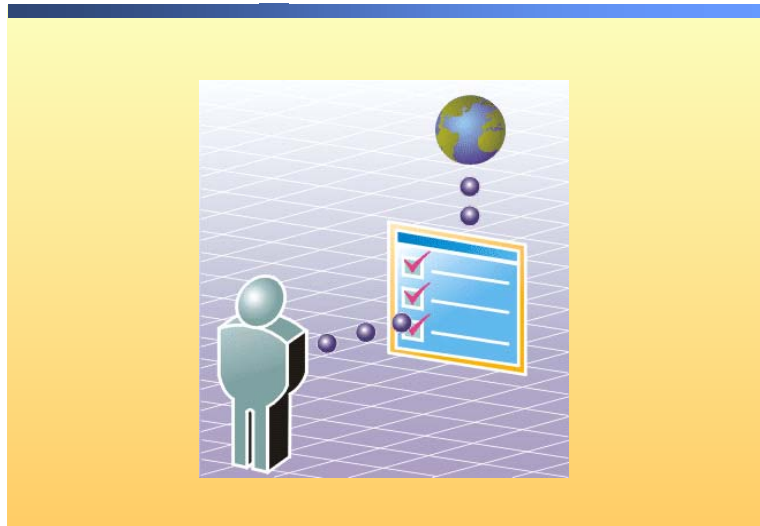
5. Which attribute should you use to request permissions for the assemblies in your XML Web service?

**SecurityPermission**

6. Which class does the .NET Framework provide to allow the developer to hook into the SOAP serialization/deserialization process?

**SoapExtension**

## Course Evaluation



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

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

At a convenient time between now and the end of the course, please complete a course evaluation, which is available at <http://www.metricsthatmatter.com/survey>.

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

---

## Module 8: Designing XML Web Services

### Contents

Overview	1
Data Type Constraints	2
Performance	11
Lab 8.1: Implementing Caching in an XML Web Service	28
Reliability	33
Versioning	37
HTML Screen Scraping XML Web Services	39
Aggregating XML Web Services	47
Demonstration: Example of an Aggregated XML Web Service	52
Lab 8.2: Implementing an Aggregated XML Web Service	53
Review	67



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, places or events 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, trademarks, 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.

© 2001-2002 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Active Directory, Authenticode, IntelliSense, FrontPage, Jscript, MSDN, PowerPoint, Visual C#, Visual Studio, Visual Basic, Windows NT, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.



# Instructor Notes

**Presentation:**  
**60 Minutes****Lab:**  
**45 Minutes**

This module teaches students which design issues to consider when designing real-world XML Web services. The issues discussed are related to data type constraints, performance, reliability, versioning, deployment in Internet Service Provider (ISP) and Application Service Provider (ASP) scenarios, and aggregating XML Web services. The module also discusses HTML screen scraping as a pseudo XML Web service.

After completing this module, you will be able to:

- Identify the restrictions that the various XML Web services protocols impose on data types.
- Explain how the use of **Application** and **Session** state can affect the performance and scaling of XML Web services.
- Explain how to use output and data caching to improve XML Web service performance.
- Explain how to implement asynchronous Web methods.
- Explain the need for instrumenting XML Web services.
- Identify the components of an XML Web service that can be versioned.
- Explain how to implement a virtual XML Web service by using screen scraping.
- Implement an XML Web service that uses multiple XML Web services.
- Identify the tradeoffs in the techniques that are used for exposing aggregated XML Web services.

**Required Materials**

To teach this module, you need the Microsoft® PowerPoint® file 2524B\_08.ppt.

**Preparation Tasks**

To prepare for this module:

- Read all of the materials for this module.
- Practice all of the demonstrations.
- Review the walkthrough code files in the *<install folder>\Democode\<language>\Mod08*.
- Complete the labs.

This section provides demonstration procedures that are not appropriate for the student notes.

### HTML Screen Scraping

#### ► Setting up the demonstration

1. Create a virtual directory with the alias Scrape and a directory of `<installroot>\Democode\<language>\Mod08\Scrape\VDir`.
2. Show the file sales.htm in the browser.

#### ► Explain the structure of the WSDL document

1. Open the file sales.wsdl in the browser.
2. Explain the output element in the document.

#### ► Demonstrate the use of screen scraping

1. Open the solution, Scrape.sln in the folder `<installroot>\Democode\<language>\Mod08\Scrape`.
2. Open the salesclient (.cs or .vb) file.
3. Explain the existing code.
4. Add a Web Reference using the URL `http://localhost/scrape/sales.wsdl`.
5. Rename the reference from localhost to Scrape.
6. Import the ScreenScrape.Scrape namespace.
7. Execute the application and discuss the output.

### Example of an Aggregated XML Web Service

#### ► Demonstrate the NorthwindClient application

1. Start the application NorthwindClient.exe, which can be found in the folder `<install folder>\Labfiles\CS\Lab08_2\Solution\NorthwindClient\bin\Debug` or `<install folder>\Labfiles\VB\Lab08_2\Solution\NorthwindClient\bin`.
2. In the **From** list, click the last **Woodgrove Online Bank** entry. In the status bar the URL for this entry will be displayed. It should be `http://instructor/woodgrove/bank.asmx`. If the correct URL is not displayed systematically check each **Woodgrove Online Bank** entry until the URL for the instructor computer is located.
3. In the **To** list, click **Contoso Micropayments** entry. In the status bar the URL for this entry will be displayed. It should be `http://instructor/contoso/Micropayment.asmx`. If the correct URL is not displayed systematically check each **Contoso Micropayments** entry until the URL for the instructor computer is located.
4. Click **Transfer**.
5. Explain that \$100 has been transferred from an account at the Woodgrove bank to an account at the micropayment service, named Contoso.
6. Explain that the Northwind Traders XML Web service addressed all of the details of managing the transfer, including retrieving routing numbers, and so on.

► **Explain the Northwind Traders XML Web service implementation**

1. In Microsoft Visual Studio® .NET, open the *<install folder>\WebServicesSolution\Northwind\Northwind* project.
2. Open *Traders.asmx.cs*.
3. Explain the implementation of **GetTModelEndpoints**, **GetTransferSinks** and **GetTransferSources** methods.
4. Explain the **EFTTransfer** method.
  - a. Describe how the **Northwind Traders** XML Web service interacts with the **Contoso** and **Woodgrove** XML Web services.
  - b. Explain how to use the binding information.

► **Explain the implementation of the CreditAccount method in the Contoso XML Web service**

1. Open the *<install folder>\Labfiles\<language>\WebServicesSolution\Contoso\Contoso* project.
2. Open the code behind file for *Micropayment.asmx*.
3. Explain the implementation of the **CreditAccount** method.
  - a. Explain routing information. (It is information required by a financial institution for electronically transferring funds to an account at another financial institution).
  - b. Explain that the Contoso XML Web service is a consumer of the Woodgrove XML Web service.

► **Explain the implementation of the AuthorizeFundsTransfer method in the Woodgrove XML Web service**

1. Open the *<install folder>\Labfiles\<language>\WebServicesSolution\Woodgrove\Woodgrove* project.
2. Open the code behind file for *Bank.asmx*.
3. Explain the implementation of the **AuthorizeFundsTransfer** method.
  - Explain the information that is contained in the **EFTBindingInfo** class.

► **Show that money is transferred between the accounts**

1. Run *<install folder>\Labfiles\<language>\Lab07\Solution\Woodgrove and Contoso Account Manager\bin\Debug\WebServiceClient.exe*.  
Leave the default accounts selected and note the account balance.
2. Switch to the *NorthwindClient.exe* application.
3. Transfer funds from the Woodgrove Online Bank to the Contoso Micropayment Service.
4. Switch to the *WebServiceClient.exe* application.
5. Click **Update Account Info**.
6. Point out that the balance has been reduced by \$100.

## Module Strategy

Use the following strategy to present this module:

- **Data Type Constraints**

Explain that even though Microsoft ASP.NET XML Web services support a rich set of data types, not all of the Microsoft .NET Framework data types are appropriate for XML Web services. Tell the students that Simple Object Access Protocol (SOAP) is the preferred protocol for XML Web services. Explain how using Hypertext Transfer Protocol (HTTP)-GET and POST methods limit the types of data a XML Web service can support. Teach the module as a set of tradeoffs. Explain to the students that circumstances may dictate different choices in different scenarios.

- **Performance**

Start out by discussing general performance considerations. Then, examine the issues that the students must consider when implementing XML Web services by using Microsoft ASP.NET. Finally, discuss the ASP.NET output and data caching capabilities and how they can be used in XML Web services.

- **Reliability**

In this topic, discuss how the .NET common language runtime enables better application reliability without compromising on performance. Also, explain how the ASP.NET process recovery model enhances reliability by supporting automatic restart of applications if the `Aspnet_wp.exe` worker process fails, and by allowing scheduled restarts to reclaim leaked resources. Also, emphasize the importance of having the ability to monitor the health of a running XML Web service to support early detection of impending application failure.

- **Versioning**

Explain that all developers of XML Web services will have to address issues with versioning. Emphasize that XML Web Services Description Language (WSDL) documents should not be versioned for production XML Web services. Discuss strategies for handling evolving interfaces by using generic Extensible Markup Language (XML) fragments. Do not get into an in-depth discussion on assembly versioning. Rather, refer the students to Course 2350A, *Securing and Deploying Microsoft .NET Assemblies*.

- HTML Screen Scraping XML Web Services

For many students, the concept of a virtual XML Web service will not be intuitive. Explain that this section is important because it is unlikely that owners of most of the data on the Internet will ever provide access to their data through XML Web services. However, making this data accessible to clients through an XML Web service is a useful paradigm. Because consumers interact with an XML Web service through a proxy class, if the proxy class simply retrieves the raw data and then parses the data locally, the consumer need not be aware that the processing takes place on the client and not on the server. A detail that you should emphasize is that screen scraping XML Web service proxies can only communicate by using the HTTP-GET protocol.

- Aggregating XML Web Services

Explain that XML Web services can be viewed as sets of functionality. There is no reason why these sets of functionality should not be aggregated to provide richer functionality. The module discusses a number of models for aggregating XML Web services. Analyze each model and discuss its areas of application. You must teach this section by basing it on the final lab scenario.



# Overview

- Data Type Constraints
- Performance
- Reliability
- Versioning
- HTML Screen Scraping XML Web Services
- Aggregating XML Web Services

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

It is relatively easy to implement a simple XML (Extensible Markup Language) Web service. However, if you implement an XML Web service that must be interoperable, scalable, reliable, and able to perform well, then there are a number of issues that you must consider when designing your XML Web service.

In this module, you will examine some of the important issues that you must consider when designing a real-world XML Web service.

## Objectives

After completing this module, you will be able to:

- Identify the restrictions imposed on data types by the various XML Web services protocols.
- Explain how the use of **Application** and **Session** state can affect the performance and scaling of XML Web services.
- Explain how output and data caching can be used to improve XML Web service performance.
- Explain how to implement asynchronous web methods.
- Explain the need for instrumenting XML Web services.
- Identify the components of an XML Web service that can be versioned.
- Explain how to implement a virtual XML Web service by using screen scraping.
- Implement an XML Web service that uses multiple XML Web services.
- Identify the tradeoffs in the techniques used for exposing aggregated XML Web services.

# Data Type Constraints

- Protocol constraints
- Structures vs. classes
- Typed vs. untyped datasets
- Arrays vs. collections
- Exceptions

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

Although the Microsoft® .NET Framework supports a rich set of data types, not all data types can or should be used as part of the interface of an XML Web service. In this topic, you will first look at how the choice of protocol for your XML Web service can limit the data types that you can use in the service. You will then look at some guidelines for choosing between similar data types, such as, structures and classes. Finally, you will look at how XML Web services expose exceptions, and the results of throwing custom exception types.

## Protocol constraints

Although Microsoft ASP.NET XML Web services automatically support Hypertext Transfer Protocol (HTTP)-GET/POST and Simple Object Access Protocol (SOAP), these protocols are not equally capable when it comes to the data types that they support. The following table summarizes the data types that you can use for each of the protocols that XML Web services support.



Type	Description	SOAP	POST	GET
Primitive types	Standard primitive types, which are as follows: <b>String, Int32, Byte, Boolean, Int16, Int64, Single, Double, Decimal, DateTime, and XmlQualifiedName</b>	✓	✓	✓
Enumeration types		✓	✓	✓
Enumeration types and arrays of primitives		✓	✓	✓
Classes and structures	Class and structure types with public fields or properties. The public properties and fields are serialized.	✓	✗	✗
Arrays of classes and structures		✓	✗	✗
<b>XmlNode</b>	Represents an XML node in an XML document.	✓	✗	✗
Arrays of <b>XmlNode</b>		✓	✗	✗

The key points to remember when choosing the protocol for your XML Web service are:

- GET and POST support only primitive data types (**int**, **string**, and so on), enumerations, and arrays of primitives.  
These data types are passed as name/value pairs.
- SOAP supports rich data types by packaging data in XML and standardizing the call format.
- SOAP provides for a rich extensibility mechanism.

It might seem obvious that it is best to choose SOAP as the protocol for your XML Web service, but there are a few other issues to consider:

- HTTP-GET and HTTP-POST are much more widely used than SOAP, and therefore, you are limiting the clients that can use your XML Web service.
- SOAP requests use a lot more bytes to transmit data than HTTP-GET or HTTP-POST requests.
- By default, ASP.NET-based XML Web services support all of the three protocols as long as you restrict the XML Web service methods to use only simple data types.

If you want to restrict the protocols that your XML Web service will support, you can make an entry in Web.config. The following code shows how to remove HTTP-GET and HTTP-POST protocols, so that the XML Web service will support only SOAP.

```
<configuration>
  <system.web>
    <webServices>
      <protocols>
        <remove name="HttpGet" />
        <remove name="HttpPost" />
      </protocols>
    </webServices>
  </system.web>
</configuration>
```

---

**Note** If you remove the HTTP-GET protocol, you will not be able to test your XML Web service methods by using the Web Service Method Description page provided for an XML Web service project in Microsoft Visual Studio® .NET.

---

### Structures vs. classes

It is very important to remember that XML Web services are *not* about object remoting. When an XML Web service method returns an object, the public fields and properties of the object are remoted. None of the functionality of the class is remoted. As a result, from an XML Web service consumer's perspective, classes and structures are superficially indistinguishable.

The following code demonstrates how the XML Schema Definition Language (XSD) allows you to define derived data types.

**C#**

---

```
public class Acct
{
    public string description;
    public string number;
    public string type;
    public decimal balance;
    public string status;
}

public class CreditCardAcct : Acct
{
    public int payPeriod;
}
```

**Visual Basic .NET**

```

Public Class Acct
    Public description As String
    Public number As String
    Public type As String
    Public balance As Decimal
    Public status As String
End Class 'Acct

Public Class CreditCardAcct
    Inherits Acct
    Public payPeriod As Integer
End Class 'CreditCardAcct

```

You can represent the preceding class in XSD as follows:

```

<s:complexType name="Acct">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="description" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="number" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="type" type="s:string" />
    <s:element minOccurs="1" maxOccurs="1" name="balance" type="s:decimal" />
  </s:sequence>
  <s:attribute name="status" type="s:string" />
</s:complexType>

<s:complexType name="CreditCardAcct">
  <s:complexContent mixed="false">
    <s:extension base="s0:Acct">
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="payperiod" type="s:int" />
      </s:sequence>
    </s:extension>
  </s:complexContent>
</s:complexType>

```

Even though you can define derived types by using XSD in the .NET environment, you may not derive a new type from a structure. Because a class and a structure are so similar, and structures have additional restrictions, classes are generally preferred to structures as data types in XML Web service methods.

**Typed vs. untyped datasets**

Datasets can be typed or untyped. A typed dataset is a dataset that is derived from a base **DataSet** class and has an XML schema file (an .xsd file) that defines the tables and fields that the dataset encapsulates. You typically use the schema when generating a typed dataset to define strongly-typed accessors for the tables and fields that the dataset encapsulates.

Because a typed **DataSet** class inherits from a base **DataSet** class, the typed class inherits all of the functionality of the base class and can be used with methods that take a dataset as a parameter.

In contrast, an untyped dataset has no corresponding built-in schema. Similar to a typed dataset, an untyped dataset contains tables, columns, and so on, but these are exposed only as collections.

You can use either typed or untyped datasets in your XML Web service. However, using typed datasets makes programming with datasets easier and less error-prone.

#### Data access using typed vs. untyped datasets

The class for a typed dataset has an object model in which its tables and columns become first-class objects in the object model. For example, if you are working with a typed dataset, you can refer to a column in a table contained in the dataset, using code similar to the following.

**C#**

---

```
string s;  
s = dsCustomersOrders.Customers[0].CustomerID;
```

**Visual Basic .NET**

---

```
Dim s As String  
s = dsCustomersOrders.Customers(0).CustomerID
```

In contrast, if you are working with an untyped dataset, the equivalent code is the following.

**C#**

---

```
string s;  
s =  
↪(string)dsCustomersOrders.Tables["Customers"].Rows[0]["CustomerID"];
```

**Visual Basic .NET**

---

```
string s;  
s =  
↪CStr(dsCustomersOrders.Tables("Customers").Rows(0)("CustomerID"))
```

#### Advantages of typed datasets

Using a typed dataset has the following advantages:

- Code is easier to read.
- Microsoft IntelliSense® fully supports typed datasets in the Visual Studio Code Editor.
- The syntax for the typed dataset provides type checking at compile time.

Accessing tables and columns in a typed dataset is also slightly faster at run time because the access path is determined at compile time. This eliminates the need to look up tables and columns in collections at run time.

**When to use untyped datasets**

Even though typed datasets have many advantages, there are a variety of circumstances in which an untyped dataset is useful. Some of these circumstances are when:

- There is no schema available for the dataset.  
There may not be an available schema if you are using a third-party component that returns a dataset.
- The data you are working with does not have a static, predictable structure.
- A dataset might be created dynamically and you do not want to define a schema.

**Arrays vs. collections**

The .NET Framework provides a number of collection classes like **HashTable**, **ArrayList**, and so on. A collection is just an unordered set of object references. When you return an instance of a collection class to an XML Web service consumer, it is no different than returning **object[]**.

In general, it is preferable to make your XML Web service interface as strongly typed as possible. If you decide to use a collection in your XML Web service, it is still very easy to return an array of the stored type.

The following code returns strongly-typed arrays of type **Acct** instead of type **object**.

C#

```
[WebMethod]
[XmlInclude(typeof(CreditCardAcct))]
[XmlInclude(typeof(SavingsAcct))]
[return:XmlArray("AccountList")]
[return:XmlArrayItem("Account")]
public Acct[] GetAllAccounts()
{
    SavingsAcct a = new SavingsAcct();
    CreditCardAcct cc = new CreditCardAcct();

    ArrayList listOfAccts = new ArrayList();
    listOfAccts.Add(a);
    listOfAccts.Add(cc);

    return (Acct[])listOfAccts.ToArray(typeof(Acct));
}
```

**Visual Basic .NET**

```

Public<WebMethod(), XmlInclude(GetType(CreditCardAcct)), _
    XmlInclude(GetType(SavingsAcct)), return:
    ↪XmlArray("AccountList")> _
Function GetAllAccounts() As<XmlArrayItem("Account")> Acct()
    Dim a As New SavingsAcct()
    Dim cc As New CreditCardAcct()
    ...
    Dim listOfAccts As New ArrayList()
    listOfAccts.Add(a)
    listOfAccts.Add(cc)

    Return CType(listOfAccts.ToArray(GetType(Acct)), Acct())
End Function 'GetAllAccounts

```

Almost the only time there is a need to return collections or arrays of type **object** is when a collection contains a heterogeneous list of objects. However, this is not a very common design pattern and in such cases, the returned collection can be an instance of a class or structure.

**Exceptions**

When implementing an XML Web service method, if an error occurs, you can either return an error code or throw an exception. Depending on the error handling method that you choose, there are certain advantages and disadvantages.

If you simply return error codes, the code values will be reliably returned, but developers often fail to check return codes.

On the other hand, if you throw an exception, and the protocol that is used to access the XML Web service is not SOAP, the XML Web service consumer will just receive an HTTP error with the error code value of 500.

If an XML Web service consumer invokes a method by using SOAP, the exception is caught on the server and wrapped inside a new object of a type **SoapException**. A **SoapException** can either be thrown by the common language runtime or by an XML Web service method. The common language runtime can throw a **SoapException** if a response to a request is not formatted correctly. XML Web service methods can generate a **SoapException** by simply throwing an exception within the method. The following code shows an XML Web service method that throws a custom exception.

**C#**


---

```
[WebMethod]
public double LoanPayments(double loanAmount, double
↳annualInterestRate,
    int numberOfYears, int numberOfPaymentsPerYear)
{
    int numberOfMonths;
    if (numberOfPaymentsPerYear == 0)
        throw new ContosoException("You cannot specify 0 for the
↳number" +
            " of payments per year.");

    int nPeriods = numberOfYears*numberOfPaymentsPerYear;
    Excel.Application xl = new Excel.Application();
    double amt = xl.WorksheetFunction.Pmt(

        annualInterestRate/numberOfPaymentsPerYear,
            nPeriods,loanAmount,0,0);
    return amt;
}
```

**Visual Basic .NET**


---

```
Public<WebMethod()> _
Function LoanPayments(loanAmount As Double, annualInterestRate
As Double, _
    numberOfYears As Integer, numberOfPaymentsPerYear As
↳Integer) As Double
    Dim numberOfMonths As Integer
    If numberOfPaymentsPerYear = 0 Then
        Throw New ContosoException("You cannot specify 0 for
the↳ number" & _
            " of payments per year.")
    End If
    Dim nPeriods As Integer = numberOfYears *
↳numberOfPaymentsPerYear
    Dim xl As New Excel.Application()
    Dim amt As Double = xl.WorksheetFunction.Pmt( _
        annualInterestRate / numberOfPaymentsPerYear, nPeriods,
↳loanAmount, 0, 0)
    Return amt
End Function 'LoanPayments
```

The following code shows an XML Web service consumer that catches the exception. Note that the exception is a **SoapException** and is *not* a **ContosoException**.

C#

---

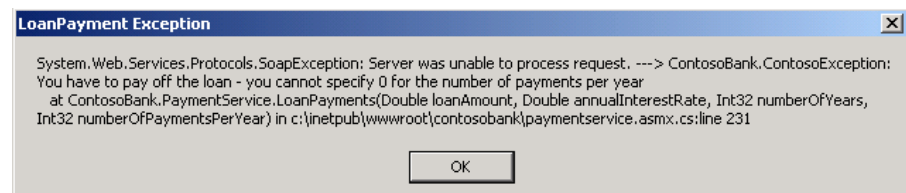
```
ContosoBank.PaymentService ps;  
ps = new ContosoBank.PaymentService();  
try {  
    double amt = ps.LoanPayments(205000.0, 0.0712, 30, 0);  
    Console.WriteLine("Monthly payments are {0}",amt);  
} catch (SoapException se) {  
    MessageBox.Show(se.Message,"LoanPayment Exception");  
}
```

Visual Basic .NET

---

```
Dim ps As ContosoBank.PaymentService  
ps = New ContosoBank.PaymentService()  
Try  
    Dim amt As Double = ps.LoanPayments(205000.0, 0.0712, 30,  
    ↪0)  
    Console.WriteLine("Monthly payments are {0}", amt)  
Catch se As SoapException  
    MessageBox.Show(se.Message, "LoanPayment Exception")  
End Try
```

Also, note that the exception that the XML Web service throws is not lost. The **Message** property of the custom exception is propagated as part of the **Message** property of the **SoapException**.





# Performance

- General Considerations
- .NET-Specific Considerations
- Caching in XML Web Services
- Asynchronous Server-Side Methods

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this section, you will begin by looking at a few universally applicable performance considerations, and then you will examine some of the ways in which you can use the capabilities of the .NET Framework to improve the performance of your XML Web service. Specifically, you will learn about caching and state management for XML Web services.

## General Considerations

- Caching
- Locking
- Making asynchronous calls
- Measuring performance and behavior
- Unnecessary code

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

There are a number of performance-related principles that you can apply to almost any situation where high performance is required.

### Caching

You can use software caching in many scenarios. For example, if performing calculations that are either time consuming or CPU intensive, or both, you may choose to store the calculated result for future use instead of recalculating each time a request for the result is received.

Here are a few guidelines on caching:

- If you cache the wrong data, then you waste memory.
- If you cache too much data, then you have less memory for other operations.
- If you cache too little data, then the cache will be ineffective because you will have to reprocess the data that was not cached.
- If you cache time-sensitive data for too long, then it will become outdated. With servers, you typically care more about speed of processing than storage, and therefore caching is done more aggressively on servers than desktop systems. This can lead to stale caches if you do not have good algorithms for detecting stale data.
- If you do not periodically invalidate and flush unused cache entries, then you will be making unnecessary demands on system memory. This could affect the performance of the Web server or other applications that are hosted on the same computer.

### Locking

Locking is not usually a problem in XML Web services unless you are using the **Application** object to store updateable state. The locks that are acquired on resources, such as database tables, will probably cause more problems.

The easiest way to protect read/write data is to use a single lock for all of the data that you need to protect. This is a simple approach, but results in serialization problems. Every thread that attempts to manipulate the data will have to wait in line to acquire the lock. If a thread is blocked on a lock, it is not doing useful work. If a server is underutilized, multiple threads that are blocked on a lock are seldom a problem, because only one thread is likely to acquire the lock at a time. If a server is under a heavy load, a high-contention lock can become a huge problem.

There are several techniques that can reduce lock contention:

- Do not be overprotective of your data, that is, do not lock data when it is not required. Hold locks for the duration necessary, and no longer. It is important not to hold locks unnecessarily for large sections of code or frequently executed sections of code.
- Partition your data so that you can use a disjoint set of locks to protect the partitioned data. For example, you could partition a customer list on the first letter of the customer's company name, and you could use a separate lock to protect each partition.
- Use the **Interlocked** class in the **System.Threading** namespace to atomically modify data without acquiring a lock.
- Use multireader/single-writer locks when the data is not modified often. Using this strategy, you can achieve better concurrency, though the lock operations will be more expensive and you risk starving writers.

### Making asynchronous calls

If you must execute an operation that takes a significant amount of time, then use asynchronous calls to execute the operation. For example, if your XML Web service calls another XML Web service in a method named **GetAccountHistory**, then you should use the **Beginxxxx/Endxxxx** methods of the proxy for the second XML Web service. If the **GetAccountHistory** method does not immediately require the data from the second XML Web service, then you could call the **Beginxxxx** method at the start of the **GetAccountHistory** method, and then call the **Endxxxx** method at the point that you require the data.

The following code demonstrates how an asynchronous call to an XML Web service method can be made from within another XML Web service method.

**C#**

```
Contoso.PaymentService ps = new Contoso.PaymentService();
IAsyncResult res = ps.BeginGetAccountInfo(1234, callback, null);
Account acct;
```

```
if (res.IsCompleted)
    acct = ps.EndGetAccountInfo(res);
```

**Visual Basic .NET**

```
Dim ps As New Contoso.PaymentService()
Dim res As IAsyncResult = ps.BeginGetAccountInfo(1234,
    ↳callback, Nothing)
Dim acct As Account
```

```
If res.IsCompleted Then
    acct = ps.EndGetAccountInfo(res)
End If
```

**Measuring performance and behavior**

Without measurements, you do not understand your application's behavior. The .NET Framework provides a number of tools for measuring the performance and behavior of your XML Web service.

Measurement encompasses black-box measurement and profiling. Black-box measurement is gathering the numbers that are exposed by external testing tools (throughput, response times) and performance counters (memory usage, context switches, CPU utilization). Many of the metrics that are measured are external to your code, but it is a good practice to instrument your code to allow for performance measurement, even when your XML Web service is deployed.

When you consider how much instrumentation to build into your XML Web service, there are a number of tradeoffs. It is true that instrumenting your code will result in some performance degradation. However, the possibility of performance degradation must be balanced with an evaluation of the risk inherent in not instrumenting your code. Instrumentation in code is not just a feature of debug builds of applications. Even after your XML Web service is deployed into production, you will need to monitor the performance of the XML Web service.

**Unnecessary code**

It is common for developers to not revisit code that has been written if it functions correctly. However, this often means that unnecessary code and inefficient code is left in the application. The following code contrasts two functionally equivalent calls to the **Trace.Write** method. In the second call, the **WriteIf** function is called whether or not the message is written. In this example, the code that tests the switch before making the call to **Trace.Write** is preferred.

**C#**

---

```
if (mySwitch.TraceVerbose) Trace.Write("Deposit failed")
Trace.WriteIf(mySwitch.TraceVerbose, "Deposit failed")
```

**Visual Basic .NET**

---

```
If mySwitch.TraceVerbose Then
    Trace.Write("Deposit failed")
End If
Trace.WriteIf(mySwitch.TraceVerbose, "Deposit failed")
```

## .NET-Specific Considerations

- **Disable session state**
- **Choose an appropriate state provider**
- **Avoid exceptions**
- **Use native database providers**
  - Stored procedures vs. ad-hoc queries
- **Use ASP.NET Web gardening**
- **Disable debug mode**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

If you are not careful with the use of **Session** state object, then you might introduce scaling or performance problems. In spite of this risk, many developers still prefer to use the session state management facilities of ASP.NET.

The following guidelines list specific techniques that you can use to avoid performance and scaling problems.

### Disable session state

Disable session state when you are not using it. To disable session state for a page, set the **EnableSessionState** attribute in the **@Page** directive to **false**.

```
<%@ Page EnableSessionState="false" %>.
```

---

**Note** If only read-only access to session state is required, set the **EnableSessionState** attribute in the **@Page** directive to **ReadOnly**.

---

You can also disable session state for each XML Web service method. To disable session state for an application, set the **mode** attribute to **off** in the **sessionstate** configuration section in the application's Web.config file.

```
<sessionstate mode="off" />.
```

**Choose an appropriate state provider**

Choose your session state provider carefully.

ASP.NET provides three distinct ways to store session data for your application:

- In-process session state.
- Out-of-process session state as a Microsoft Windows® service.
- Out-of-process session state in a Microsoft SQL Server™ database.

If you need your XML Web service to scale out across multiple processors, multiple computers, or when you cannot lose data if a server or a process is restarted, then you must not use the in-process session state provider.

Instead, you should use one of the out-of-process session state providers.

**Avoid exceptions**

Because exceptions cause significant performance degradations, you should never use them as a way to control normal program flow. If it is possible to detect a condition that would cause an exception in your code, then you should do that instead of waiting to catch the exception itself before handling that condition.

**Use native database providers**

The .NET Framework provides two data providers: the OLE DB data provider and the SQL) data provider. The SQL data provider is recommended for building high-performance Web applications.

**Stored procedures vs. ad-hoc queries**

When using the managed Microsoft SQL Server provider, you can receive an additional performance boost by using compiled stored procedures instead of ad-hoc queries. For information about using SQL Server stored procedures, see Course 2389B, *Programming with ADO.NET*.

Use the **SqlDataReader** class for a fast forward-only data cursor. The **SqlDataReader** class provides a means to read a forward-only data stream that is retrieved from a SQL Server database. While creating an ASP.NET XML Web service, you might encounter situations that allow you to use **SqlDataReader**. The **SqlDataReader** class offers higher performance than the **DataSet** class. This is because **SqlDataReader** uses the Tabular Data Stream (TDS) protocol to read data directly from a database connection.

**Use ASP.NET Web gardening**

The ASP.NET process recovery model enables scalability on multiprocessor computers by distributing work to several processes, one per CPU, each with processor affinity set to its CPU. This technique is called Web gardening, and can dramatically improve the performance of certain applications.

**Disable debug mode**

Always remember to disable debug mode before deploying a production application or conducting any performance measurements. The following example code shows how to disable the debug mode:

```
<configuration>
  <system.web>
    <compilation defaultLanguage="c#" debug="false" />
  </system.web>
</configuration>
```

## Caching in XML Web Services

- Output caching
- Data caching
- Controlling caching
  - File and key-based dependencies
  - Expiration policies
  - Item priorities
  - Removal notification
- Scenarios

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

ASP.NET provides two types of caching that you can use to create high-performance XML Web services:

- *Output caching* allows you to store responses that are generated by an XML Web service method. On subsequent requests, rather than re-executing the method, the cached output is used to service the request.
- *Data caching* programmatically stores arbitrary objects in memory, so that your application can save the time and resources that it takes to re-create them.

Both of these forms of caching are available to an XML Web service developer.

### Output caching

Output caching of an entire .asmx page does not make sense, because the .asmx page is just a convenient entry point to the XML Web service. In the context of XML Web services, output caching is done on a per-XML Web service method basis.

Output caching for an XML Web service method is enabled by adding the **CacheDuration** property to the **WebMethod** attribute. The following code shows how to cache the results of a method call for 60 seconds.

C#	Visual Basic .NET
<pre>[WebMethod(CacheDuration=60)] public Acct[] GetAccounts(int acctID) {</pre>	<pre>Public &lt;WebMethod(CacheDuration := 60)&gt; _ Function GetAccounts(acctID As Integer) _     As Acct()</pre>

In output caching, note that each unique set of input parameters results in a unique cached result.

**Data caching**

Data caching is implemented by the **Cache** class in the **System.Web.Caching** namespace. Each ASP.NET application has an instance of the **Cache** class that is private to that application, and whose lifetime is bounded by the lifetime of the application. When an application is restarted, the instance of its **Cache** object is re-created.

You can place items in a cache and later retrieve them by using key-value pairs. In an XML Web service, the **Cache** object can be accessed through the **Context** object. The following code example demonstrates this functionality.

**C#**

---

```
Context.Cache["ListOfSources"] = dsSources;
```

**Visual Basic .NET**

---

```
Context.Cache("ListOfSources") = dsSources
```

**Controlling caching**

Although the **Cache** class offers a simple interface for you to customize cache settings, it also offers powerful features that allow you to customize how items are cached and how long they are cached.

For example, when system memory becomes scarce, the cache automatically removes seldom used or unimportant items to free memory for processing requests. This technique is called *scavenging*. It is one of the ways by which the cache ensures that data that is not frequently used does not waste valuable server resources.

---

**Note** A cache has no information about the contents of the items that it contains. It merely holds a reference to those objects. A caching system can also provide a way to track the dependencies of those objects and set expiration policies.

---

**File and key-based dependencies**

You can define the validity of a cached item based on an external file, a directory (file dependencies), or another cached item (key dependencies). If a dependency changes, the cached item is invalidated and removed from a cache. You can use this technique to remove items from the cache when their data source changes. For example, if you store the list of current employees in an XML file for a Human Resources application, you can store a reference to an **XmlDocument** object that represents the XML file in the cache. When the source file is updated, the reference is removed from the cache. When required, the application reads the XML file and a new version of the XML document is inserted into the cache.

You can add an item to a cache by passing an instance of the **CacheDependency** class to the **Add** or **Insert** method. The following example demonstrates how to use the **Insert** method to add an item to the cache with a dependency on an XML file.



---

**C#**

```
XmlDocument doc = new XmlDocument();
Doc.Load(Server.MapPath("employees.xml"));
CacheDependency depend = new
CacheDependency(Server.MapPath("employees.xml"));
Context.Cache.Insert("Employees", doc, depend);
```

---

**Visual Basic .NET**

```
Dim doc As New XmlDocument()
Doc.Load(Server.MapPath("employees.xml"))
Dim depend As New
CacheDependency(Server.MapPath("employees.xml"))
Context.Cache.Insert("Employees", doc, depend)
```

The **CacheDependency** class also allows you to monitor arrays of files and directories, cache keys, or both.

### Expiration policies

You can add an item to the cache and include an expiration time. You can set absolute time-out periods or timeouts relative to the last access of the cached item. The following code uses the **Insert** method to add an item to the cache with an absolute expiration of two minutes from the time of insertion into the cache.

---

**C#**

```
Cache.Insert("Employees", doc, null,
    ↪DateTime.Now.AddMinutes(2),
    NoSlidingExpiration);
```

---

**Visual Basic .NET**

```
Cache.Insert("Employees", doc, Nothing,
    ↪DateTime.Now.AddMinutes(2), _
    NoSlidingExpiration)
```

The following code uses the **Insert** method to add an item to the cache with a sliding expiration of 30 seconds.

---

**C#**

```
Cache.Insert("Employees", doc, null,    NoAbsoluteExpiration,
    TimeSpan.FromSeconds(30));
```

---

**Visual Basic .NET**

```
Cache.Insert("Employees", doc, Nothing, NoAbsoluteExpiration,
    ↪_
    TimeSpan.FromSeconds(30))
```

---

**Note** You can either define an absolute expiration or a sliding expiration, but not both.

---

Whichever expiration parameter you use, you must set the other parameter to zero. The **Cache** class defines two fields that do this automatically: **NoAbsoluteExpiration** and **NoSlidingExpiration**. When you define an absolute or a sliding expiration, set the appropriate parameter to its corresponding field value.

**Item priorities**

All items in the cache will eventually be removed from the cache. This may happen because the specified duration for the lifetime of an object in the cache has exceeded, or because of memory shortage. If items must be removed from the cache because of memory shortage, the common language runtime uses a least-recently used (LRU) algorithm to decide which items must be removed first. If you think that not all of the items in the cache are of equal importance, then you might prefer to have the less important cached items removed first from the cache during a memory shortage. You can accomplish this by assigning priority values (from the **CacheItemPriority** enumeration) to the items when they are cached. As a result, the lower priority items are released first from the cache.

The following example uses the **Add** method to add an item to the cache with a priority value of **High**.

**C#**

---

```
Context.Cache.Add("Employee", doc,
    null, NoAbsoluteExpiration,
    TimeSpan.FromSeconds(30),
    CacheItemPriority.High,
    null);
```

**Visual Basic .NET**

---

```
Context.Cache.Add("Employee", doc, _
    Nothing, NoAbsoluteExpiration, _
    TimeSpan.FromSeconds(30), _
    CacheItemPriority.High, _
    Nothing)
```

**Removal notification**

An item may be removed from the cache for any of the following reasons:

- The item's timeout expired.
- An object that the cached item was dependent on changed.
- The item was explicitly removed by using the **Remove** method of the **Cache** class.
- The common language runtime was scavenging for memory and the item became eligible for removal.

When an item expires, it is removed from the cache. Attempts to retrieve its value will return null unless the item has been added to the cache again. When an item is removed from the cache, a specified function can execute. The reason for removal is provided as an argument to this function, and in this function, you can perform some appropriate action. For example, you might want to update the data that was cached and re-cache it.

Using removal notification in XML Web services requires a little additional work. Because the object that encapsulates your XML Web service might be recycled at the end of each XML Web service method, the callback function will not necessarily be invoked on a thread that is in the middle of responding to a request. Therefore, the callback function cannot be an instance method. Instead, it must be a static method. However, the **Context** object, which gives you access to the **Cache** object, is only available in instance methods. As a result, you cannot re-cache items in the callback function, unless you find a way to pass a reference to the **Cache** object into the callback function.

The following code shows how you might implement removal notification. All of the following code would be part of the class implementing an XML Web service.

```
1.      internal class CacheData {
2.          internal XmlDocument doc;
3.          internal string path;
4.      }
5.      private static CacheItemRemovedCallback onRemove = null;
6.
7.      public static void RemovedCallback(string key, Object v,
8.          CacheItemRemovedReason r) {
9.          CacheData cd = (CacheData) v;
10.         cd.doc.Load(cd.path);
11.         HttpRuntime.Cache.Insert(key, cd,
12.             new CacheDependency(cd.path),
13.             Cache.NoAbsoluteExpiration, Cache.NoSlidingExpiration,
14.             CacheItemPriority.Default,
15.             new CacheItemRemovedCallback(
16.                 Service1.RemovedCallback));
17.     }
18.
19.     [WebMethod]
20.     public string GetEmployees()
21.     {
22.         CacheData cd = (CacheData)Context.Cache["Employees"];
23.         return cd.doc.DocumentElement.OuterXml;
24.     }
```

```
1.            Friend Class CacheData
2.                Friend doc As XmlDocument
3.                Friend path As String
4.            End Class 'CacheData
5.
6.            Private Shared onRemove As CacheItemRemovedCallback = Nothing
7.
8.            Public Shared Sub RemovedCallback(key As String, v As
9.                ↵[Object], r As CacheItemRemovedReason)
10.                Dim cd As CacheData = CType(v, CacheData)
11.                cd.doc.Load(cd.path)
12.                HttpRuntime.Cache.Insert(key, cd, _
13.                    New CacheDependency(cd.path), _
14.                    Cache.NoAbsoluteExpiration, Cache.NoSlidingExpiration, _
15.                    CacheItemPriority.Default, _
16.                    New CacheItemRemovedCallback( _
17.                        AddressOf Service1.RemovedCallback))
18.            End Sub 'RemovedCallback
19.
20.            Public<WebMethod()> _
21.                Function GetEmployees() As String
22.                    Dim cd As CacheData = CType(Context.Cache("Employees"),
23.                        ↵CacheData)
24.                    Return cd.doc.DocumentElement.OuterXml
25.                End Function 'GetEmployees
```

The functionality of the preceding code can be described as follows:

- In lines 1 through 4, a class named **CacheData** is defined. In this example, a reference to an XML document is cached, but the class also has a field named **path** that will store the path to the document that is being cached.
- In line 6, a static reference to a delegate of type **CacheItemRemovedCallback** is defined.
- In lines 8 through 17, a function named **RemovedCallback** is defined. This function will be called when the item is removed from the cache.
- In line 9, the **CacheData** object is recovered.
- In line 10, the modified XML document is reloaded.
- In line 11 through 16, the updated **CacheData** object is stored back in the cache by using the same key when it was first stored.
- In line 22, the **CacheData** object that was cached is retrieved.

The following code provides an example of how to initialize the cache when the application starts. The event handler is found in Global.asax.

#### C# code example

```

1.      protected void Application_Start(Object sender, EventArgs e) {
2.          XmlDocument doc = new XmlDocument();
3.          CacheData cd = new CacheData();
4.          cd.c = Context.Cache;
5.          cd.path = Server.MapPath("employees.xml");
6.          doc.Load(cd.path);
7.          cd.doc = doc;
8.          onRemove = new CacheItemRemovedCallback(
9.              Service1.RemovedCallback);
10.         Context.Cache.Insert("Employees", cd,
11.             new CacheDependency(cd.path),
12.             Cache.NoAbsoluteExpiration, Cache.NoSlidingExpiration,
13.             CacheItemPriority.Default,
14.             CacheItemPriorityDecay.Never,
15.             onRemove);
16.     }

```

#### Visual Basic .NET code example

```

1.      Protected Sub Application_Start(sender As [Object], e As
2.          EventArgs)
3.          Dim doc As New XmlDocument()
4.          Dim cd As New CacheData()
5.          cd.c = Context.Cache
6.          cd.path = Server.MapPath("employees.xml")
7.          doc.Load(cd.path)
8.          cd.doc = doc
9.          onRemove = New CacheItemRemovedCallback( _
10.              AddressOf Service1.RemovedCallback)
11.         Context.Cache.Insert("Employees", cd, _
12.             New CacheDependency(cd.path), _
13.             Cache.NoAbsoluteExpiration, Cache.NoSlidingExpiration, _
14.             CacheItemPriority.Default, CacheItemPriorityDecay.Never,
15.             _
16.             onRemove)
17.     End Sub 'Application_Start

```

The functionality of the preceding code can be described as follows:

- In lines 4 through 7, the **CacheData** object is initialized.
- In lines 8 through 9, the delegate reference **onRemove** is initialized. Note that the callback function is the method named **RemovedCallback**.
- In lines 10 through 14, the **CacheData** object is inserted into the cache. Note that the **onRemove** delegate is supplied, so that the callback function is called when the item is removed from the cache.

**Scenarios**

Not all data is suitable for caching. Typically, it is recommended that you only cache data that changes relatively infrequently. For example, the list of your organization's customers, organized by region, might be a relatively slow-changing list, and therefore would make a good candidate for caching.

XML Web service methods, whose results depend on specific input parameters, and where the possible range of values for the parameters is not restricted, are not suitable for caching. For example, if you implemented a method to add two numbers and return the result, it would not be a good idea to cache the result.

## Asynchronous Server-Side Methods

- **Implementing asynchronous WebMethods**
  - C# implementation
  - Visual Basic .NET implementation
- **Appropriate uses**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

We have seen that XML Web service client proxies expose asynchronous versions of the XML Web service methods via a pair of **Begin/End** methods. It is also possible to implement asynchronous XML Web service methods by using a similar pattern to the client proxy pattern.

### Implementing asynchronous WebMethods

Implementing an XML Web service method asynchronously requires that the synchronous version of the method be split into two methods. The two methods must be named **Beginxxx** and **Endxxx**.

#### C# Implementation

The **Beginxxx** method signature includes all the *in* and *ref* parameters plus two additional parameters. The additional parameters are of type **AsyncCallback** and **object**. The method returns an object of type **AsyncResult**.

The **Endxxx** method signature includes all of the *ref* and *out* parameters plus an additional parameter of type **AsyncResult**. The method returns the same type that the synchronous version of the method returned.

**C#**

---

```
[WebMethod]
IAsyncResult BeginGetAccounts(AsyncCallback callback, object
asyncState)
{
    WoodgroveOnlineBank bank = new WoodgroveOnlineBank();
    asyncState = bank;
    IAsyncResult ar =
    ↪ bank.BeginGetAllAccounts(callback, asyncState);
    return ar;
}

Acct[] EndGetAccounts(IAsyncResult result)
{
    WoodgroveOnlineBank bank =
    ↪ (WoodgroveOnlineBank) result.AsyncState;
    return bank.EndGetAllAccounts(result);
}
```

**Microsoft Visual Basic®  
.NET Implementation**

The **Beginxxx** method signature includes all the *ByVal* and *ByRef* parameters plus two additional parameters. The additional parameters are of type **AsyncCallback** and **Object**. The method returns an object of type **IAsyncResult**.

The **Endxxx** method signature includes all the *ByRef* parameters plus an additional parameter of type *IAsyncResult*. The method returns the same type that the synchronous version of the method returned.

**Visual Basic .NET**

---

```
<WebMethod()> _
Function BeginGetAccounts(callback As AsyncCallback,
    ↪ asyncState As Object) As IAsyncResult
    Dim bank As New WoodgroveOnlineBank()
    asyncState = bank
    Dim ar As IAsyncResult = bank.BeginGetAllAccounts(callback,
    ↪ asyncState)
    Return ar
End Function 'BeginGetAccounts

Function EndGetAccounts(result As IAsyncResult) As Acct()
    Dim bank As WoodgroveOnlineBank = CType(result.AsyncState,
    ↪ WoodgroveOnlineBank)
    Return bank.EndGetAllAccounts(result)
End Function 'EndGetAccounts
```



**Appropriate uses**

It is recommended that you consider using an asynchronous implementation of an XML Web service method in the following situations:

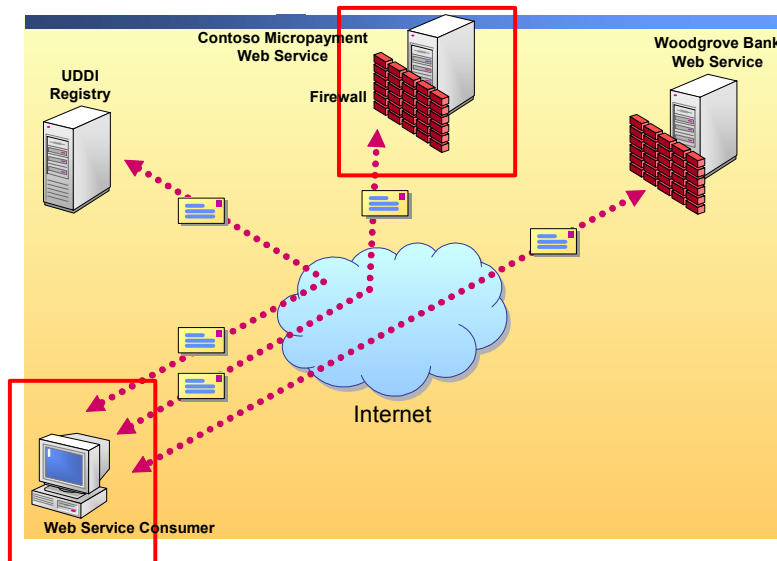
- The XML Web service method will be invoking another XML Web service.
- The XML Web service method will be performing file input/output (I/O).
- The XML Web service is performing any I/O that could take a long time and is based on Microsoft Win32® kernel object handles (for example, socket I/O).

---

**Note** Even if you implement an XML Web service method as a pair of asynchronous functions, the method is still exposed as a single XML Web service operation.

---

## Lab 8.1: Implementing Caching in an XML Web Service



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Objective

After completing this lab, you will be able to implement caching in an XML Web service.

**Note** This lab focuses on the concepts in this module and as a result may not comply with Microsoft security recommendations

### Lab Setup

There are starter and solution files that are associated with this lab. The starter files are in the folder `<labroot>\Labfiles\Lab08_1\Starter`. The solution files for this lab are in the folder `<labroot>\Labfiles\Lab08_1\Solution`.

If you did not complete Lab 7.1, Securing XML Web Services, in Module 7, “Securing XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*, copy the `<labfolder>\Lab07\Solution\Contoso` project to the `C:\Inetpub\Wwwroot` folder, overwriting your existing Contoso project.

### Scenario

In this lab, you will implement a method on the Contoso XML Web service that returns an XML document with information about the different membership levels that the Contoso Micropayment service supports. The XML document containing the information will be cached by using the .NET Framework caching infrastructure. If the underlying XML document is modified, the cache will be invalidated and the XML document must be recached.

You will test the solution to this lab by using the Service Help page for the Contoso XML Web service.

**Estimated time to complete this lab: 40 minutes**

## Exercise 1

### Extending the Contoso Micropayment XML Web Service

In this exercise, you will add a Web method named **GetMembershipInformation** to the Contoso Micropayment XML Web service. This method will retrieve a cached XML document and return its contents to the client. You will also add code to *Global.asax* to initialize the cache and refresh it when the underlying XML document is modified.

#### ► Initialize the cache

1. Open the Contoso project in Microsoft Visual Studio® .NET.

---

**Important** If you did not complete the modifications to the Contoso project in Lab 7.1, refer to the Lab Setup section at the beginning of this lab for additional instructions.

---

2. Open the code behind file for *Global.asax*.
3. Import the **System.Xml** and **System.Web.Caching** namespaces.
4. Add the following class definition above the class **Global**.

C#	Visual Basic .NET
<code>internal class CacheData {</code>	<code>Friend Class CacheData</code>
<code>internal XmlDocument doc;</code>	<code>Friend doc As XmlDocument</code>
<code>internal string path;</code>	<code>Friend path As String</code>
<code>}</code>	<code>End Class 'CacheData</code>

5. Add an object of type **CacheItemRemovedCallback** to the **Global** class. The following code shows how to do this.

C#

---

```
private static CacheItemRemovedCallback onRemove = null;
```

Visual Basic .NET

---

```
Private Shared onRemove As CacheItemRemovedCallback =  
    Nothing
```

6. Add code to the **Application\_Start** method to load the XML document named **memberships.xml** from the XML Web service folder. The following code shows how to do this.

C#

---

```
XmlDocument doc = new XmlDocument();  
doc.Load(Server.MapPath("memberships.xml"));
```

Visual Basic .NET

---

```
Dim doc As XmlDocument = New XmlDocument()  
doc.Load(Server.MapPath("memberships.xml"))
```

7. Initialize an instance of the **CacheData** class named **cd** with the XML document object and the path to the XML document. The following code shows how to do this.

**C#**

---

```
CacheData cd = new CacheData();
cd.path = Server.MapPath("memberships.xml");
cd.doc = doc;
```

**Visual Basic .NET**

---

```
Dim cd As New CacheData()
cd.path = Server.MapPath("memberships.xml")
cd.doc = doc
```

8. Add the **CacheData** object to the cache. The following code shows how to do this.

**C#**

---

```
onRemove = new CacheItemRemovedCallback(
    Global.RemovedCallback);
Context.Cache.Insert("MembershipTypes", cd,
    new CacheDependency(cd.path),
    Cache.NoAbsoluteExpiration, Cache.NoSlidingExpiration,
    CacheItemPriority.Default,
    onRemove);
```

**Visual Basic .NET**

---

```
onRemove = New CacheItemRemovedCallback( _
    AddressOf Global.RemovedCallback)
Context.Cache.Insert("MembershipTypes", cd, _
    New CacheDependency(cd.path), _
    Cache.NoAbsoluteExpiration, Cache.NoSlidingExpiration,
    ↪_
    CacheItemPriority.Default, _
    onRemove)
```

### ► Implement the removal notification function

1. Add a method named **RemovedCallback** to the class **Global**.
  - a. In the method, cast the object that has been removed to an object of type **CacheData**.
  - b. Reload the XML document by using the path that is stored in the cached object.
  - c. Reinsert the cached object into the cache.

The following code shows how to do this.

**C#**


---

```
public static void RemovedCallback(string key, Object v,
    CacheItemRemovedReason r)
{
    CacheData cd = (CacheData) v;
    cd.doc.Load(cd.path);
    HttpRuntime.Cache.Insert(key, cd,
        new CacheDependency(cd.path),
        Cache.NoAbsoluteExpiration, Cache.NoSlidingExpiration,
        CacheItemPriority.Default,
        new CacheItemRemovedCallback(Global.RemovedCallback));
}
```

**Visual Basic .NET**


---

```
Public Shared Sub RemovedCallback(key As String, v As
[Object], r As CacheItemRemovedReason)
    Dim cd As CacheData = CType(v, CacheData)
    cd.doc.Load(cd.path)
    HttpRuntime.Cache.Insert(key, cd, _
        New CacheDependency(cd.path), _
        Cache.NoAbsoluteExpiration, Cache.NoSlidingExpiration,
        _
        CacheItemPriority.Default, _
        New CacheItemRemovedCallback( _
            AddressOf Global.RemovedCallback))
End Sub 'RemovedCallback
```

**► Add the GetMembershipInformation method**

1. Open the code behind file for Micropayment.asmx.
2. Import the **System.Xml** and **System.Web.Caching** namespaces.
3. Add a method named **GetMembershipInformation** to the **MicroPaymentService** class. This method accepts no arguments and returns an **XmlNode**.
4. Add a **WebMethod** attribute to the method.
5. In the method, use the **HttpRuntime** class to retrieve the cached data, and return the cached XML document to the client. The following code shows how to do this.

**C#**


---

```
CacheData cd =
(CacheData)HttpRuntime.Cache["MembershipTypes"];
return cd.doc.DocumentElement;
```

**Visual Basic .NET**


---

```
Dim cd As CacheData =
CType(HttpRuntime.Cache("MembershipTypes"), CacheData)
Return cd.doc.DocumentElement
```

► **Test the caching behavior**

1. Copy the file **memberships.xml** from <labroot>\Lab08\_1\Starter to the folder c:\Inetpub\wwwroot\Contoso.
2. Open the code behind file for Global.asax.
3. Set a breakpoint in the **RemovedCallback** method.
4. Run the XML Web service in debug mode.
5. When the Service Help page is displayed, click **GetMembershipInformation**.
6. On the Service Help page, click **Invoke**.  
You will see the membership information displayed.
7. Open the file **memberships.xml** in the folder c:\Inetpub\wwwroot\Contoso by using Notepad.exe.
8. Change the **cost** attribute for the Gold membership to 3500 and save the file.  
The debugger should break at the breakpoint set in the **GetMembershipInformation** method.
9. Press the F5 key.

Switch back to the Service Help Page and click **Invoke**. You will see the updated membership information displayed.

# Reliability

- **Availability**
  - Process isolation
  - Process recovery model
- **Measurement**
  - Performance counters
  - WMI

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

Developing reliable applications is a difficult task. For example, small memory leaks may go undetected in a testing environment, but they accumulate over time. Also, your application might maintain a linked list that continues growing. The memory leaks and the growing linked list might result in performance degradation over time, which is unacceptable. As performance degrades, and your XML Web service is frequently unavailable, clients will perceive the application as unreliable.

## Availability

Your XML Web services might become unavailable for the following reasons:

- Individual processes might fail.
- A significant number of XML Web services fail frequently if you are running a Web farm.

The common language runtime helps you address the preceding issues in a number of ways.

## Process isolation

Most developers implement applications with little or no regard for the other applications that run on the same computer. This can lead to unintended interactions. One way to reduce the risk of unintended interactions is to isolate all applications in separate processes. However, this solution makes more demands on system resources than when applications share processes. As a result, there is a tradeoff between reliability and performance.

With application domains in common language runtime, you are able to achieve both reliability and performance in your XML Web service. Application domains provide application isolation within the same process boundary. Because of this feature, it is often a good idea to migrate any unmanaged code that your XML Web service uses to the managed environment. Migrating unmanaged code will reduce the performance penalty incurred when your XML Web service calls a component outside of the application domain.

**Process recovery model**      All ASP.NET code runs in an external worker process named **aspnet\_wp.exe**. This process can automatically restart your application domain if it fails. Any memory is reclaimed during the garbage collection for the common language runtime, and if a deadlock is detected, the runtime performs deadlock recovery.

You can also configure the worker process to proactively reset itself based on a timer or on-demand. Proactively resetting the process is a useful preventive measure because it minimizes the chances of your application experiencing a slowdown because of resource depletion, or experiencing problems with counter roll-overs, and so on.

**Measurement**      Earlier, you learned about a few general considerations for measuring performance and behavior of an application. This section examines the .NET Framework support for instrumenting XML Web services.

**Performance counters**      Windows performance counters allow your applications and components to publish, capture, and analyze the performance data that applications, services, and drivers provide. You can use this information to determine system bottlenecks, and fine-tune system and application performance. For example, you might use a performance counter to track the amount of time that is required to process an order, or query a database. The decision about how to instrument your XML Web service is a tradeoff between the performance impact that writing to performance counters imposes and the utility gained by instrumenting the XML Web service. If your XML Web service will be deployed by an Internet Service Provider (ISP) or an Application Service Provider (ASP), it is vital that you instrument the XML Web service.

You can use the **PerformanceCounter** class for both reading predefined or custom counters and writing performance data to custom counters.

To read from a performance counter:

1. Create an instance of the **PerformanceCounter** class.
2. Set the **CategoryName**, **CounterName**, and, optionally, the **InstanceName** or **MachineName** properties.
3. Call the **NextValue** method to get the reading.

**WMI**      Windows Management Instrumentation (WMI) provides a rich set of system management services that are built into the Microsoft Windows operating system. A broad spectrum of applications, services, and devices are available that use WMI to provide extensive management features for information technology (IT) operations and product support organizations. The use of WMI-based management systems leads to more robust computing environments and greater system reliability, which allows savings for corporations.



WMI provides extensive instrumentation to accomplish almost any management task for many high-end applications, such as Microsoft Exchange Server, Microsoft SQL Server, and Microsoft Internet Information Services (IIS). An administrator can perform the following tasks:

- Monitor the health of the applications.
- Detect bottlenecks or failures.
- Manage and configure applications.
- Query application data (use the traversal and querying of object relationships).
- Perform seamless local or remote management operations.

The WMI architecture consists of the following three tiers:

- *Clients*. Software components that perform operations by using WMI (for example, reading management details, configuring systems, and subscribing to events).
- *Object manager*. A broker between providers and clients that provides some key services, such as standard event publication and subscription, event filtering, query engine, and so on.
- *Providers*. Software components that capture and return live data to the client applications, process method invocations from the clients, and link the client to the infrastructure that is managed.

The provision of data and events, and the ability to configure systems are provided seamlessly to clients and applications through a well-defined schema. In the .NET Framework, the **System.Management** namespace provides common classes to traverse the WMI schema.

The following code shows how to define a WMI **Instance** class.

C#

---

```
[InstrumentationClass(InstrumentationType.Abstract)]
public class AbstractClass : Instance {
    public string Property_Name;
}

[InstrumentationClass(InstrumentationType.Instance)]
public class InstanceClass : AbstractClass {
    public int Sample_Number;
}

...
InstanceClass instClass = new InstanceClass();
instClass.Property_Name = "Hello";
instClass.Sample_Number = 888;
instClass.Published = true;
```

**Visual Basic .NET**

---

```
<InstrumentationClass(InstrumentationType.Abstract)> _  
Public Class AbstractClass  
    Inherits Instance  
    Public Property_Name As String  
End Class 'AbstractClass  
<InstrumentationClass(InstrumentationType.Instance)> _  
  
Public Class InstanceClass  
    Inherits AbstractClass  
    Public Sample_Number As Integer  
End Class 'InstanceClass  
...  
Dim instClass As New InstanceClass()  
instClass.Property_Name = "Hello"  
instClass.Sample_Number = 888  
instClass.Published = True
```

# Versioning

- Versioning assemblies
- Versioning WSDL documents
- Versioning request or response payload

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Any code deployed into production is likely to evolve through multiple versions. XML Web services are not any different. As a developer of an ASP.NET XML Web service, you must consider a number of issues that are specific to XML Web services.

## Versioning assemblies

You manage versioning in the .NET Framework at the assembly level and use it only when locating strong-named assemblies. By default, the common language runtime loads the assembly whose version number matches the version number in the metadata of the calling assembly. However, you can instruct the runtime to load a different version of an assembly by using version policy.

Version policy is the set of rules that specify which version of a dependent assembly to bind to. Version policy is defined in the policy configuration files. When used in conjunction with the global assembly cache (GAC), versioning and version policy make side-by-side deployment and execution of assemblies much easier.

From the perspective of XML Web services, the fact that you can have side-by-side deployment of different versions of an assembly means that you can deploy different versions of an XML Web service.

For more information about assembly versioning, see Course 2350A, *Securing and Deploying Microsoft .NET Assemblies*.

## Versioning WSDL documents

You define the interface to your XML Web service by using Web Services Description Language (WSDL). If you decide to modify the interface to your XML Web service, it is recommended that you modify any registration information in Universal Description, Discovery, and Integration (UDDI) to indicate that your business service no longer supports the **tModel** that is associated with the previous version of the WSDL document. It is vital that you do not simply modify the WSDL document associated with a previously registered **tModel**, because existing consumers of your XML Web service may no longer function correctly with the modified interface.

**Versioning request or response payload**

If any parameter to an XML Web service method or the return value of a method changes its structure over time, then you can handle the versioning issues by defining the parameter or return type to be of the type **XmlElement** or **XmlNode**. You can then apply the **XmlAnyElement** or **XmlAnyAttribute** to the parameter or the field. These attributes instruct the common language runtime to capture all elements and attributes that are found in an XML document. The following code shows how to do this.

**C#**

---

```
[WebMethod]
public class Company
{
    [return:XmlAnyElement]
    XmlNode GetOrganizationStructure() {
        ...
    }
}
```

**Visual Basic .NET**

---

```
<WebMethod()> _
Public Class Company
    ...
    Function GetOrganizationStructure() As<XmlAnyElement()>
        ↪XmlNode
    ...
    End Function 'GetOrganizationStructure
End Class 'Company
```

## HTML Screen Scraping XML Web Services

- Regular Expressions in the .NET Framework
- Using Regular Expressions in WSDL Documents
- Demonstration: Screen Scraping an HTML Document

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

It is not always realistic to expect that data will be conveniently parsed and stored in a database for easy retrieval. Often, data is dumped to flat files. If you want to use a portion of this data, then you must retrieve the document and parse it to extract the information of interest.

Writing code to parse data is a tedious and error-prone process if the data is not organized into well-defined fields. A useful way to reduce the effort that is involved is to use an engine that can support regular expressions, which you can use to parse data.

Microsoft has combined regular expression parsing technology with Web Service Description Language (WSDL) and the .NET XML Web service proxy technology to enable you to build virtual XML Web services on top of static documents. When applied to Web pages, this technique is known as screen scraping.

## Regular Expressions in the .NET Framework

- The **Regex** class
- The **Match** class
- The **MatchCollection** class

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

Regular expressions provide a powerful, flexible, and efficient method for processing text. The extensive pattern-matching notation of regular expressions allows you to:

- Quickly parse large amounts of text to find specific character patterns.
- Extract, edit, replace, or delete text substrings.
- Add the extracted strings to a collection to generate a report.

Regular expressions are an indispensable tool for many applications that manipulate strings, such as HTML processing, log file parsing, and HTTP header parsing.

The regular expressions of the Microsoft .NET Framework are designed to be compatible with Perl 5.0 regular expressions. You can find the regular expression classes for the .NET Framework in the **System.Text.RegularExpressions** namespace.

### The **Regex** class

The **Regex** class represents a read-only regular expression. The following example creates an instance of the **Regex** class and defines a simple regular expression when the object is initialized.

#### C#

```
Regex r;  
r = new Regex(@"\s2000");
```

#### Visual Basic .NET

```
Regex r;  
r = new Regex("\s2000");
```

**The Match class**

The **Match** class represents the results of a regular expression match. The following example uses the **Match** method of the **Regex** class to return an object of type **Match** to find the first match in the input string. The example uses the **Success** property of the **Match** class to find out if a match was found.

```
Regex r = new Regex("abc");
Match m = r.Match("123abc456");
if (m.Success)
{
    Console.WriteLine("Found match at position " + m.Index);
}
```

**Escaped characters**

Most of the important regular expression language operators are unescaped single characters. The escape character “\” (a single backslash) notifies the regular expression parser that the character following the backslash is not an operator. The following table lists the character escapes that are recognized in regular expressions.

Escaped character	Meaning
ordinary characters	Characters other than ., \$, ^, {, [, (,  , ), *, +, ?, and \ match themselves.
\a	Matches a bell alarm (\u0007).
\b	Matches backspace (\u0008) if in [ ] character classes; otherwise, in a regular expression, \b denotes a word boundary (between \w and \W characters)
\t	Matches a tab (\u0009).
\r	Matches a carriage return (\u000D).
\v	Matches a vertical tab (\u000B).
\f	Matches a form feed (\u000C).
\n	Matches a new line (\u000A).
\e	Matches an escape (\u001B).
\040	Matches an ASCII character as octal (up to three digits). For example, \040 represents a space.
\x20	Matches an ASCII character by using hexadecimal representation (exactly two digits).
\cC	Matches an ASCII control character; for example, \cC is CTRL+C.
\u0020	Matches a Unicode character by using hexadecimal representation (exactly four digits).
\	When followed by a character that is not recognized as an escaped character, matches that character. For example, \* is the same as \x2A.

**Character classes**

Character classes are the set of characters that define which substring to match. The following table summarizes character matching syntax.

Character class	Meaning
.	Matches any character except \n unless modified by the <b>Singleline</b> option.
[aeiou]	Matches a single character included in the specified set of characters.
[^aeiou]	Matches any single character that is not in the specified set of characters.
[0-9a-zA-F]	Matches any character in the specified ranges.
\w	Matches any word character. Same as [a-zA-Z_0-9].
\W	Matches any nonword character. Same as [^a-zA-Z_0-9].
\s	Matches any white space character. Same as [\f\n\r\t\v].
\S	Matches any nonwhite space character. Same as [^\f\n\r\t\v].
\d	Decimal digit. Same as [0-9].
\D	Nondigit. Same as [^0-9].

For more information, see the topic “Regular Expression Language Elements” in the .NET Framework SDK documentation.

**The MatchCollection Class**

The **MatchCollection** class represents a sequence of successful nonoverlapping matches when applying a regular expression to an input string. The **Regex.Matches** property returns instances of **MatchCollection**.

The following example uses the **Matches** method of the **Regex** class to fill an instance of the **MatchCollection** class with all of the matches that are found in the input string. The following example code copies the match collection to a string array that holds all of the matches and an integer array that indicates the position of each match.

**C#**

---

```
MatchCollection mc;
string[] results = new String[20];
int[] matchposition = new int[20];

Regex r = new Regex("abc");
mc = r.Matches("123abc4abcd");
for (int i = 0; i < mc.Count; i++)
{
    results[i] = mc[i].Value;
    matchposition[i] = mc[i].Index;
}
```



**Visual Basic .NET**

---

```
Dim mc As MatchCollection
Dim results() As String = New [String](20) {}
Dim matchposition(20) As Integer

Dim r As New Regex("abc")
mc = r.Matches("123abc4abcd")
Dim i As Integer
For i = 0 To mc.Count - 1
    results(i) = mc(i).Value
    matchposition(i) = mc(i).Index
Next i
```

## Using Regular Expressions in WSDL Documents

- Create a WSDL document manually
- Specify the output message format in the output element within the binding element

```
<urt:text>
  <urt:match name="nameOfElement"
    pattern="regular expression ..."/>
  ....
</urt:text>
```

### ■ Example

```
<urt:text>
  <urt:match name="LineItem"
    pattern="&lt;tr&gt;(.*)?&lt;/tr&gt;"
    repeats="*" />
  </urt:match>
</urt:text>
```

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

If you want to consume a flat file as if it were an XML Web service by creating an XML Web service proxy, you need a service description. In the case of flat files screen scraping, no WSDL can be generated automatically. Therefore, you have to construct the WSDL document manually.

The interesting part of a manually created WSDL document is the format of the output, which is specified in the **binding** element. The output is simply an XML document with the following structure:

```
<urt:text>
  <urt:match name="nameOfElement"
    pattern="regular expression ...">

</urt:text>
```

The **pattern** attribute of the **match** element is a Perl 5.0 compatible regular expression. The **match** elements can also have nested **match** elements to perform pattern matching within a match.

### Example

Consider the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<family>
  <parents>
    <mother>Clair</mother>
    <father>Peter</father>
  </parents>
  <children>
    <child>John</child>
    <child>Robert</child>
    <child>Jim</child>
    <child>Amy</child>
  </children>
</family>
```

To mimic an XML Web service that can provide a list of parents and a list of children, you can have the following WSDL document:

```
<?xml version="1.0"?>
<definitions standard namespaces omitted for brevity
  xmlns:urt=↵
    "http://microsoft.com/wsdl/mime/textMatching/">
  <types/>
  <message name="GetFamilyInfoHttpGetIn"/>
  <message name="GetFamilyInfoHttpGetOut"/>
  <portType name="FamilyHttpGet">
    <operation name="GetFamilyInfo">
      <input message="s0:GetFamilyInfoHttpGetIn"/>
      <output message="s0:GetFamilyInfoHttpGetOut"/>
    </operation>
  </portType>
  <binding name="FamilyHttpGet" type="s0:FamilyHttpGet">
    <http:binding verb="GET"/>
    <operation name="GetFamilyInfo">
      <http:operation location=""/>
      <input>
        <http:urlEncoded/>
      </input>
      <output>
        <urt:text>
          <urt:match name="listOfChildren"
            pattern="(&lt;children&gt;(.*)?&lt;/children&gt;)"
            ignoreCase="1">
          <urt:match name="children"
            pattern="&lt;child&gt;(.*)?&lt;/child&gt;"
            repeats="*" />
          </urt:match>
        </urt:text>
      </output>
    </operation>
  </binding>
  <service name="Family">
    <port name="FamilyHttpGet" binding="s0:FamilyHttpGet">
      <http:address
        location="http://www.cpandl.com/Scrape/family.xml"/>
    </port>
  </service>
</definitions>
```

You can process this WSDL document by using Wsdl.exe to produce a proxy class. You can use this proxy class like any other proxy to an XML Web service.

## Demonstration: Screen Scraping an HTML Document

Region	Q1	Q2	Q3	Q4
NorthWest	10,000	13,500	14,200	11,000
Midwest	81,100	76,500	82,500	81,000
South	66,400	69,100	72,500	75,300
NorthEast	97,100	99,400	102,900	98,200

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this demonstration, you will see how data can be extracted from an HTML page and presented to the client as if it were an XML Web service that was providing the data.

## Aggregating XML Web Services

- Aggregated XML Web Service Scenarios
- Designing an XML Web Service for Aggregation
- Demonstration: Example of an Aggregated XML Web Service

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

Developers are constantly looking for ways to reuse code. XML Web services present another opportunity for code reuse. The reuse model is different than code libraries because of the loose coupling between XML Web services and their consumers. The reuse scenarios for XML Web services encompass interorganization workflows and service aggregation. Although a detailed discussion of workflow applications is beyond the scope of this course, it is useful to examine service aggregation as a way of reusing XML Web services.

## Aggregated XML Web Service Scenarios

- Gateways to XML Web services
- Simple interfaces to complex XML Web services
- Portals to XML Web services

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

There are many possible ways that the services that an aggregated XML Web service provides can be made available to consumers.

### Gateway to XML Web services

An organization has an internal XML Web service behind a corporate firewall. The XML Web service uses integrated Windows authentication. A corporate decision is made to allow external clients (for example trading partners) to use the XML Web service. The external clients must be authenticated by using a different authentication mechanism because they may be traversing a proxy server before accessing the XML Web service.

A simple solution for this scenario is to implement the XML Web service with exactly the same interface as the internal XML Web service, and place the new XML Web service outside the firewall. The new XML Web service would provide a secure gateway to the XML Web service behind the firewall by authenticating all requests and then forwarding them to the internal XML Web service.

### Simple interfaces to complex XML Web services

Some XML Web services may expose very rich functionality. The interface to the XML Web service may be very complex. For example, the Microsoft TerraServer .NET XML Web service is able to return photographic images, topographic maps, and relief maps by using IDs to entities called tiles. The tiles can be located by latitude and longitude. Suppose you wanted to retrieve an image of the city of Portland, Oregon, in the United States. You would first have to find the latitude and longitude information, determine the tile ID, and then retrieve the image. However, the TerraServer XML Web service does not currently provide the capability of determining the latitude and longitude of a city based on its name. It would be useful to have an XML Web service that had an operation that took a city, state or province, and country as arguments and returned the image of the appropriate tile. The XML Web service would retrieve the latitude and longitude from one XML Web service and use the information to find the tile ID and then image at the TerraServer XML Web service.

**Portals to XML Web services**

In lab 8.2 associated with this module, you will implement an XML Web service, named Northwind Traders, that performs an electronic funds transfer. It does this transfer by acting as a proxy for the consumer and handles the details of obtaining routing information, and initiating the electronic funds transfer. The XML Web service communicates with two other XML Web services that act as the source and recipient of the funds being transferred. The only thing that a client of the Northwind Traders XML Web service must do is select the source and destination financial institution, and supply the amount for the transfer and security credentials for the source financial institution. This is considerably simpler than having to request routing information from one financial institution and deliver that information to another financial institution manually.

The previous scenario is an example of how an XML Web service that aggregates other XML Web services can provide additional value to the consumer. Instead of the consumer having to interact with multiple XML Web services, the aggregating XML Web service can act as a portal to other XML Web services.

## Designing an XML Web Service for Aggregation

- Protocol considerations
- Handling non-interactive clients
- Designing for ISPs
- Designing for ASPs
  - Monitoring and metering
- Self-repair and remote repair

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

One of the biggest sources of problems for Application Service Providers (ASPs) is that very little third-party software is designed for Internet delivery. You need to consider what kinds of design constraints will be imposed by the environment where you expect your XML Web service to be deployed. In this topic, you will examine some of these issues as a guide to designing an XML Web service that you can use in a variety of scenarios.

<b>Protocol considerations</b>	The choice of protocol affects what data types your XML Web service methods can use. It is recommended that you select SOAP as your protocol of choice. Not only does SOAP support a very rich set of data types, but also, SOAP is not tied to HTTP as an underlying protocol. In fact SOAP will soon be usable over File Transfer Protocol (FTP) and Simple Mail Transfer Protocol (SMTP).
<b>Handling noninteractive clients</b>	If your XML Web service is designed to expect interactive clients, then it will probably not perform well in a scenario where other XML Web services are aggregating your XML Web service.
<b>Designing for ISPs</b>	<p>Many organizations are outsourcing the physical hosting of their Web sites to Internet Service Providers (ISPs). The challenge for the XML Web service developer is that the hardware on which the XML Web service is deployed will probably be shared with other customers also. Consequently, there will most likely be much stricter security policies in place to ensure that applications for different customers do not accidentally or deliberately interfere with each other. XML Web service developers must ensure that all of the required permissions are requested so that the ISP can correctly configure the security policy for the XML Web service.</p> <p>Another issue that you must address is that the physical location of the XML Web service will be inaccessible to you. Therefore, if the XML Web service must be configurable after it is deployed to the ISPs infrastructure, you must provide a secure administrative interface to your XML Web service.</p>
<b>Designing for ASPs</b>	ASPs sell subscriptions to software. They can only charge customers for service usage that can be tracked.



**Monitoring and metering**

ASPs want the capability to isolate and monitor individual transactions within the application. They want the ability to configure what aspects of an application will be monitored. The capability to closely monitor many system metrics makes it easier for the ASP to correctly determine which clients are using which services and the exact actions that are being performed. This is the basis for application usage metering.

Another desirable attribute of an application from the ASP perspective is the ability of applications to automatically generate alerts to notify key personnel. You can implement much of the required eventing and monitoring infrastructure by using the WMI support in the

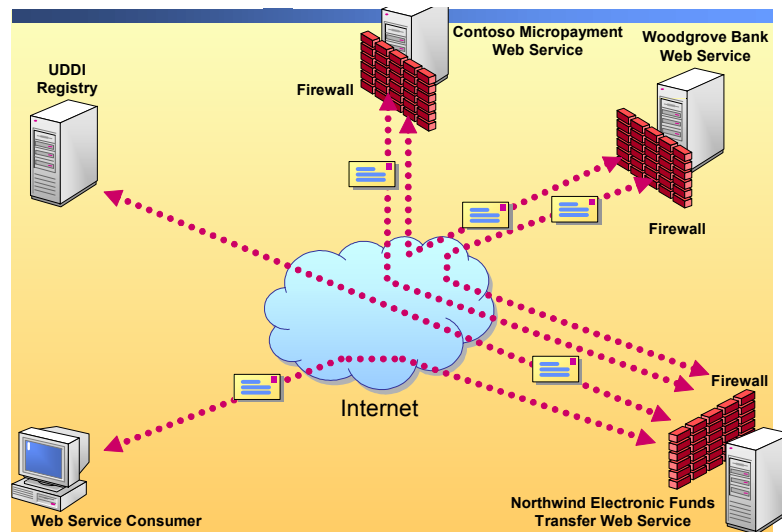
**System.Management.Instrumentation** namespace and by using the **Performance Counter** classes to write to the Windows eventLogs.

**Self-repair and remote repair**

It is recommended that applications also take some measures to perform self-repair if problems are diagnosed. For example, if an XML Web service is a client of a second XML Web service and it detects that the second XML Web service is unavailable, it should attempt to locate a backup service provider.

If your application cannot perform self-diagnosis and repair, you should also provide an administrative interface that will allow you to configure settings like database connection strings in Web.config.

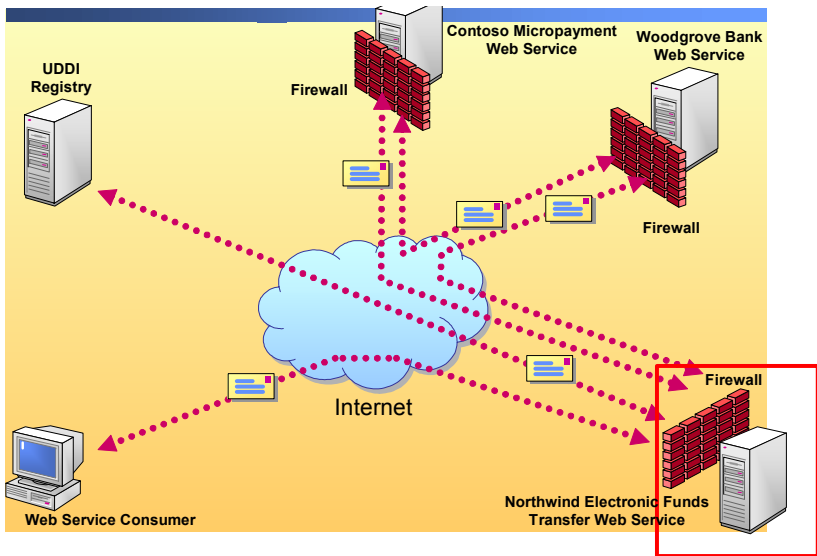
## Demonstration: Example of an Aggregated XML Web Service



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

In this demonstration, you will see an example of an XML Web service named Northwind that aggregates two other XML Web services (Contoso and Woodgrove) to implement an electronic funds transfer portal. This demonstration illustrates the solution to the lab associated with this module.

## Lab 8.2: Implementing an Aggregated XML Web Service



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Objective

After completing this lab, you will be able to implement an aggregating XML Web service that uses multiple XML Web services.

**Note** This lab focuses on the concepts in this module and, as a result, may not comply with Microsoft security recommendations.

### Lab Setup

There are starter and solution files that are associated with this lab. The starter files are in the folder `<labroot>\Labfiles\Lab08_2\Starter`. The solution files for this lab are in the folder `<labroot>\Labfiles\Lab08_2\Solution`.

If you did not complete Lab 8.1, Implementing Caching in an XML Web Service, in Module 8, “Designing XML Web Services,” in Course 2524B, *Developing XML Web Services Using Microsoft ASP.NET*, copy the `<labfolder>\Lab08_1\Solution\Contoso` project to the `C:\Inetpub\Wwwroot` folder, overwriting your existing Contoso project.

### Scenario

In this lab, you will implement an aggregating XML Web service named Northwind Traders and a client to consume this service. The Northwind Traders XML Web service will be a portal to facilitate electronic funds transfers between financial institutions. It will locate XML Web services for financial institutions that can be either a source or a destination of the funds transfers. It will allow a funds transfer between any of the located financial institutions.

You will also implement a client that uses the Northwind Trader XML Web service.

**Estimated time to complete this lab: 90 minutes**

## Exercise 1

### Extending the Contoso Micropayment XML Web Service

In this exercise, you will add a **CreditAccount** method to the Contoso Micropayment XML Web service. This method will invoke the Woodgrove Online Bank **TransferFunds** method, which transfers funds from a Woodgrove account to a Contoso account.

#### ► Add the **CreditAccount** method

1. Open the Contoso project in Microsoft Visual Studio .NET.

---

**Important** If you did not complete the modifications to the Contoso project in Lab 7.1, refer to the Lab Setup section at the beginning of this lab for additional instructions.

---

2. Open the code behind file for Micropayment.asmx.
3. To provide information to allow Contoso XML Web service to bind to the Woodgrove XML Web service, add the following structure.

C#	Visual Basic .NET
<pre>public struct EFTBindingInfo {     public string token;     public string endPoint; }</pre>	<pre>Public Structure EFTBindingInfo     Public token As String     Public endPoint As String End Structure 'EFTBindingInfo</pre>

The **endPoint** string provides the URL that the **CreditAccount** method will use to call the Woodgrove **TransferFunds** method. The **token** string provides an encrypted string that the Woodgrove **AuthorizeFundsTransfer** method returns. The **TransferFunds** method uses this token to verify that the transfer has been authorized, and that the authorization has not expired.

4. Begin creating the **CreditAccount** method by adding the following method signature to the **MicroPaymentService** class.

C#
<pre>public void CreditAccount(EFTBindingInfo bindingInfo,     ↳decimal amount)</pre>
Visual Basic .NET
<pre>Public Sub CreditAccount(bindingInfo As EFTBindingInfo,     ↳amount As Decimal)</pre>

5. In the **CreditAccount** method, add attributes that do the following:
  - a. Expose the method as an XML Web service method.
  - b. Use a SoapHeader that uses the **authInfo** data member in the header.
  - c. Decrypt the request header.

► **Call the Woodgrove TransferFunds method in CreditAccount**

1. Add a Web Reference to the Woodgrove Online Banking XML Web service.
2. Rename the **localhost** namespace to **Bank**.
3. Add code to import the **Contoso.Bank** namespace.
4. To invoke the Woodgrove **TransferFunds** method, within the **CreditAccount** method, create a **WoodgroveOnlineBank** proxy class object.
5. Set the **Url** data member of the proxy class to the **endPoint** string that the *EFTBindingInfo* parameter provides.
6. Open the **Class View**.
7. Expand the **Contoso.Bank** namespace.
8. Expand the **WoodgroveOnlineBank** class.
9. Notice that the **TransferFunds** method takes an **EFTRoutingInfo** object. The **EFTRoutingInfo** class provides routing information for the Contoso Micropayment financial institution and the Contoso target account that the Woodgrove bank requires to complete a transfer.
10. Create an **EFTRoutingInfo** object.
11. Set the **EFTRoutingInfo** data members as follows.

Data Member	Value
<b>ABA_RoutingNo</b>	<b>"12345"</b>
<b>AccountName</b>	<b>"Contoso Micropayments"</b>
<b>AccountNumber</b>	<b>"12345678"</b>
<b>SubAccountName</b>	<i>Name of the Contoso account holder</i>
<b>SubAccountNumber</b>	<i>The Contoso account number</i>

To obtain the **SubAccountName** and **SubAccountNumber**, invoke the Contoso **GetAccount** method. Although this method is exposed as an XML Web service method, it can also be invoked locally.

Recall that **GetAccount** returns an **AccountDataSet**. The **\_GetAccount[0].Name** (**\_GetAccount(0).Name** for Visual Basic programmers) data member provides the **SubAccountName**. The **\_GetAccount[0].AccountID** (**\_GetAccount(0).AccountID** for Visual Basic programmers) data member provides the **SubAccountNumber**.

12. Invoke the Woodgrove proxy class **TransferFunds** method. Use the *bindingInfo.token* property as the *token* parameter and use the **EFTRoutingInfo** object created in step 10 as the *ri* parameter.
13. Save the **EFTConfirmation** object that is returned.

Lastly, add a credit transaction to the TransactionLog table of the Contoso database. The **\_CreateTransaction** stored procedure is provided to do this. Note that, unlike the other stored procedures that you have used in the Contoso XML Web service, **\_CreateTransaction** does not return a dataset.

► **Record the Contoso transaction in the database**

1. Create a **SqlConnection** object. Initialize its **ConnectionString** property by using **ConfigurationSettings.AppSettings** to retrieve the value of the **connectStringContoso** application setting.
2. Create a **SqlCommand** object.
  - a. Initialize its **CommandText** property to "**\_CreateTransaction**".
  - b. Initialize its **CommandType** property to **System.Data.CommandType.StoredProcedure**.
  - c. Initialize its **Connection** to the connection created in step 1.
3. For each of the parameters of the **\_CreateTransaction** stored procedure, add a **SqlParameter** object to the **SqlCommand** object.

```
_CreateTransaction(@userID AS nvarchar(16), @password AS
nvarchar(16), @transDate AS datetime, @amount AS money,
@desc AS nvarchar(50), @type AS char(2), @transactionID AS
int OUTPUT)
```

You can assign most of the parameters for the constructor of the **SqlParameter** class default values. The following table specifies the parameters and the default values that **SqlParameter** should use.

Parameter name	Value
<i>ParameterDirection</i>	<b>System.Data.ParameterDirection.Input</b>
<i>IsNullable</i>	<b>True</b>
<i>Precision</i>	<b>0</b>
<i>Scale</i>	<b>0</b>
<i>SourceColumn</i>	<b>""</b>
<i>DataRowVersion</i>	<b>System.Data.DataRowVersion.Current</b>

The following table lists the parameters to be added and values that are specific to each parameter.

Name	Type	Size	Value
<i>@userID</i>	<b>NVarChar</b>	16	<b>authInfo.Username</b>
<i>@password</i>	<b>NVarChar</b>	16	<b>authInfo.Password</b>
<i>@transdate</i>	<b>DateTime</b>	8	<b>DateTime.Now</b>
<i>@amount</i>	<b>Money</b>	8	the <i>amount</i> parameter
<i>@desc</i>	<b>NVarChar</b>	50	a string that contains the <b>transactionID</b> member of the <b>EFTConfirmation</b> object
<i>@type</i>	<b>Char</b>	2	"CR"
<i>@transactionID</i>	<b>Int</b>	4	<b>null/Nothing</b>

For the *@transactionID* parameter, you must set the associated **ParameterDirection** to **System.Data.ParameterDirection.Output**.

4. Open a connection object.
5. Call **SqlCommand.ExecuteNonQuery** to invoke the stored procedure.
6. Close the connection.
7. Build and run the XML Web service. The **CreditAccount** method will be displayed on the Service Help page, but you will not be able to invoke the method since the method uses data types that are incompatible with the HTTP GET protocol.

## Exercise 2

### Creating the Northwind Trader XML Web Service

In this exercise, you will implement the Northwind Traders EFT portal. The Northwind Traders XML Web service will aggregate the Contoso and Woodgrove XML Web services (and any XML Web service that is compatible with these XML Web services). The Northwind Traders XML Web service will allow clients to specify a source and destination financial institution and then initiate an electronic fund transfer between the two financial institutions. To support this functionality, this XML Web service provides operations to retrieve lists of compatible source and destination financial institutions. It also brokers the communication between the client and the two aggregated XML Web services.

#### ► Create the Northwind Trader XML Web service

1. In Visual Studio .NET, on the **File** menu, point to **New**, and then click **Project**.
2. Select the **ASP.NET Web Service** project template for the language of your choice.
3. Name the project **Northwind** and create it in the default location.
4. Click **OK** to begin.
5. Rename **Service1.aspx** to **Traders.aspx**.
6. To open **Traders.aspx**, right-click **Traders.aspx**, and click **View Code** on the shortcut menu.
7. Import the **System.Web.Services.Protocols**, **System.Xml**, **System.Xml.Serialization**, and **System.Net** namespaces.
8. To allow a method in the Northwind Traders XML Web service to use SOAP header authentication, add a **SoapHeader**-derived class to the **Northwind** namespace. The class should look as follows.

C#	Visual Basic .NET
<pre>public class AuthToken : SoapHeader {     public string srcUserName;     public string srcPassword;     public string destUserName;     public string destPassword; }</pre>	<pre>Public Class AuthToken     Inherits SoapHeader     Public srcUserName As         ↪String     Public srcPassword As         ↪String     Public destUserName As         ↪String     Public destPassword As         ↪String End Class 'AuthToken</pre>

This **SoapHeader** class encapsulates the user name and password information that you will use when invoking the methods in both the Woodgrove and Contoso XML Web services.



9. Rename the **Service1** class to **Traders**.
10. For C# programmers, rename the **Service1()** constructor to **Traders()**.
11. Add a public instance of **AuthToken** class, named **authInfo**, to the **Traders** class.

► **Add Web references to Contoso and Woodgrove**

1. Add a Web reference to the Woodgrove Online Banking XML Web service.
2. To rename the **localhost** namespace to **Source**, in Solution Explorer, expand **Web References**. Right-click **localhost** and click **Rename** from the shortcut menu, and then type **Source**.
3. Add a Web reference to the Contoso XML Web service.
4. Rename the **localhost** namespace containing the **Contoso** proxy classes to **Sink**.
5. Import the **Northwind.Sink** and **Northwind.Source** namespaces.

► **Add the EncryptionExtension attribute to the methods in the Contoso XML Web service proxy class**

1. Open **Class View**.
2. Expand the **Northwind.Sink** namespace.
3. Double-click the **ContosoMicropaymentService** class.
4. Add the **EncryptionExtension** attribute to the **GetAccount**, **GetTransactionHistory**, and **CreditAccount** methods in the proxy class.

C#

---

```
[EncryptionExtension(Encrypt=EncryptMode.Request,
    ↳SOAPTarget=Target.Header)]
```

Visual Basic .NET

---

```
<EncryptionExtension(Encrypt := EncryptMode.Request,
    ↳SOAPTarget := Target.Header)>
```

► **Add references to the encryption extension and UDDI assemblies**

1. Add a reference to the provided EncryptionExtension.dll assembly that can be found in the folder `<labroot>\WebServicesSolution\Utils`.
2. Add a reference to the Microsoft.Uddi.Sdk.dll assembly.
3. Import the following namespaces into the code behind file for Traders.asmx.

```
Microsoft.Uddi
Microsoft.Uddi.Api
Microsoft.Uddi.Authentication
Microsoft.Uddi.Binding
Microsoft.Uddi.Business
Microsoft.Uddi.Service
Microsoft.Uddi.ServiceType
```



4. Within this method, create an **ArrayList** object.
5. Within a **try/Try** block:
  - a. Set the **Inquire.Url** property to **http://glasgow/uddi/api/inquire.asmx**.
  - b. Set the **Inquire.AuthenticationMode** property to **AuthenticationMode.UddiAuthentication**.
  - c. Create a new instance of the class **NetworkCredential**. Pass the following values to the constructor:

Parameter	Value
Username	MOCUser
Password	MOC\$Pwd
Domain	GLASGOW

- d. Assign the instance of **NetworkCredential** to the **Inquire.HttpClient.Credentials** property.
6. Create a **FindBusiness** object.
7. Add a **tModelKey** to the **FindBusiness.TModelKeys** collection by using the *tModelKey* parameter.
8. Call the **Send** method of the **FindBusiness** object.
9. Store the returned **BusinessList** object.
10. For each **BusinessInfo** object in the **BusinessList.BusinessInfos** collection, loop through the **ServiceInfos** collection. For each **ServiceInfo** object in this collection, do the following:
  - a. Create a **FindBinding** object.
  - b. Set the **ServiceKey** property to the value of the **ServiceInfo.ServiceKey** property.
  - c. Add a **tModelKey** to the **FindBinding.TModelKeys** collection by using the *tModelKey* method parameter.
  - d. Call the **Send** method of the **FindBinding** object.
  - e. Store the returned **BindingDetail** object.
  - f. Loop through the **BindingDetail.BindingTemplates** collection.
  - g. Create an **AccessPoint** object. Set the **AccessPoint.businessName** property to the **BindingTemplate.Name** property. Set the **AccessPoint.endPoint** property to the **BindingTemplate.AccessPoint.Text** property.
  - h. Add the **AccessPoint** object to the **ArrayList** object.

11. Return the **ArrayList** object as an **AccessPoint[]** object, using the **ToArray** method, as shown in the following code:

**C#**

---

```
return  
➔ (AccessPoint[])accessPoints.ToArray(typeof(AccessPoint));
```

**Visual Basic .NET**

---

```
Return accessPoints.ToArray(GetType(AccessPoint))
```

12. Add a **catch/Catch** block.
13. Leave the **catch/Catch** block empty. After the **catch/Catch** block, return **null** for C# programmers and **Nothing** for Visual Basic .NET programmers.

► **Implement methods that retrieve source and sink (destination) endpoints**

1. Add a method with the following signature to the **Traders** class:

**C#**

---

```
[WebMethod]  
[return:XmlArray("Sources")]  
public AccessPoint[] GetTransferSources()
```

**Visual Basic .NET**

---

```
<WebMethod()> _  
Public Function GetTransferSources()  
As<XmlArray("Sources")> AccessPoint()
```

This method will return all fund transfer sources (Woodgrove implementations) registered with the UDDI directory.

2. Within the **GetTransferSources** method, call the **GetTModelEndpoints** method with the **tModelKey** identifier corresponding with Woodgrove. Return the results of the method call.
3. Add a method with the following signature to the **Traders** class.

**C#**

---

```
[WebMethod]  
[return:XmlArray("Sinks")]  
public AccessPoint[] GetTransferSinks()
```

**Visual Basic .NET**

---

```
<WebMethod()> _  
Public Function GetTransferSinks() As<XmlArray("Sinks")>➔  
AccessPoint()
```

This method will return all fund transfer targets (Contoso implementations) registered with the UDDI directory.

4. Within the **GetTransferSinks** method, call **GetTModelEndpoints** with the **tModelKey** identifier corresponding with Contoso. Return the results of the method call.

► **Implement a method to initiate a funds transfer**

1. Add the following class declaration to the **Northwind** namespace.

**C#**

```
public class BankInfo
{
    public string uri;
    public int acctID;
}
```

**Visual Basic .NET**

```
Public Class BankInfo
    Public uri As String
    Public acctID As Integer
End Class 'BankInfo
```

You will use this class to return endpoint information to clients.

2. Add a method with the following signature to the **Traders** class.

**C#**

```
[WebMethod]
[SoapHeader("authInfo",Required=true)]
public string EFTTransfer(BankInfo src, BankInfo snk,
    ↪decimal amt)
```

**Visual Basic .NET**

```
<WebMethod(), SoapHeader("authInfo", Required := True)> _
Public Function EFTTransfer(src As BankInfo, snk As
    ↪BankInfo, amt As Decimal) As String
```

3. Within the **EFTTransfer** method, create a **WoodgroveOnlineBank** proxy class object.
4. Create a **WoodgroveAuthInfo** class object. Set its **Username** and **Password** methods to the **srcUserName** and **srcPassword** data members of the **authInfo** object.
5. Set the **WoodgroveAuthInfoValue** property of the **WoodgroveOnlineBank** proxy to the **WoodgroveAuthInfo** object.
6. Set the **Url** property of the **WoodgroveOnlineBank** proxy to the **uri** data member of the **src** parameter.
7. Create a **ContosoMicropaymentService** proxy class object.
8. Create a **ContosoAuthInfo** class object. Set its **Username** and **Password** methods to the **destUserName** and **destPassword** data members of the **authInfo** object.

9. Set the **ContosoAuthInfoValue** property of the **ContosoMicropaymentService** proxy to the **ContosoAuthInfo** object.
10. Set the **Url** property of the **ContosoMicropaymentService** proxy to the **uri** data member of the *snk* parameter.
11. Invoke the **AuthorizeFundsTransfer** method of the Woodgrove proxy.
  - a. Set the *acctID* parameter to the **acctID** data member of the *src* parameter.
  - b. Set the *amount* parameter to this method's *amt* parameter.
  - c. Save the returned **Northwind.Source.EFTBindingInfo** object.

---

**Note** When you declare the **EFTBindingInfo** instance, you must fully qualify **EFTBindingInfo** with the **Northwind.Source** namespace because the **Northwind.Sink** namespace also defines a structure of this same type.

---

12. Create a **Northwind.Sink.EFTBindingInfo** object.
13. Copy the **endPoint** and **token** data member from the returned **Northwind.Source.EFTBindingInfo** object to the **Northwind.Sink.EFTBindingInfo** object.
14. Invoke the Contoso proxy **CreditAccount** method.
  - a. Set the *bindingInfo* parameter to the **Northwind.Sink.EFTBindingInfo** object.
  - b. Set the *amount* parameter to this method's *amt* parameter.
15. Return the string "**Transaction succeeded**".
16. Enclose the code that you added in steps 3 through 14 within a **try/Try** block.
17. Add a **catch/Catch** block to catch any errors. In the event that an error is thrown, return the string "**Transaction failed**".

► **Test the application**

1. Build the application.
2. Press F5 to invoke the Help page.
3. Using the Help pages, invoke **GetTransferSources**.

You should see the Woodgrove implementation that you registered in Lab 6.1.
4. Using the Help pages, invoke **GetTransferSinks**.

You should see the Contoso implementation that you registered in Lab 6.1.

## Exercise 3

### Using the Northwind Trader XML Web Service

In this exercise, you will extend the Northwind client that is provided to you to use the Northwind Trader XML Web service. The starter application project can be found at <labroot>\Lab08\_2\Starter\NorthwindClient.

#### ► Add a Web reference to the Northwind XML Web service

1. Add a Web reference to the Northwind XML Web service.
2. Rename the **localhost** namespace to **Northwind**.
3. Open the following file.

C#	Visual Basic .NET
NorthwindClient.cs	NorthwindClient.vb

4. Import the **NorthwindClient.Northwind** namespace.

#### ► Populate the To and From list boxes

1. Locate the constructor for the form.
2. Create a **Traders** object after the existing code within this method.
3. Call the **GetTransferSources** method of the **Traders** object.
4. Store the returned **AccessPoint** array object.
5. Loop through each **AccessPoint** object in the array. For each object in the array, do the following:
  - a. Create a new **ListItem** object.

---

**Note** The **ListItem** class is defined within NorthwindClient (.cs or .vb) file. It is used to store information about financial institutions in the list boxes.

---

- b. Pass the **AccessPoint.businessName** and **AccessPoint.endPoint** members as parameters to the constructor.
  - c. Add the **ListItem** object to the **lstFrom** list.
6. Repeat steps 3 through 5, and call **GetTransferSinks** instead of **GetTransferSources**. Add **ListItem** objects to the **lstTo** list.

#### ► Transfer funds

1. Locate the **btnTransfer\_Click** method.
2. Create **BankInfo** objects for the source financial institution.
  - a. Set the **BankInfo.acctID** property to the value stored in the **txtAcctIDFrom** textbox.
  - b. Set the **BankInfo.uri** property to the **ListItem.Url** property of the **ListItem** selected in the **lstFrom** listbox.

3. Create **BankInfo** objects for the destination financial institution.
  - a. Set the **BankInfo.acctID** property to the value stored in the **txtAcctIDTo** textbox.
  - b. Set the **BankInfo.uri** property to the **ListItem.Url** property of the **ListItem** selected in the **lstTo** listbox.
4. Create a Northwind Traders XML Web service proxy class object.
5. Create a **SoapHeader** authentication object.
6. Populate the **SoapHeader** authentication object with the values found in **txtUserFrom**, **txtUserTo**, **txtPasswordFrom**, and **txtPasswordTo** boxes.
7. Set the **Cursor** property of the form to **Cursors.WaitCursor** to indicate that the application is processing.
8. Set the **statusBar.Text** property to "Processing...".
9. Transfer the funds by calling the **EFTTransfer** method on the Northwind Traders XML Web service proxy with the source and destination **BankInfo** objects, and the amount specified by the **Text** property of the **txtAmt** object.
10. Display the results of the **EFTTransfer** call in a messagebox.
11. Set the **Cursor** property of the form to **Cursors.Default** to indicate that the application is processing.
12. Reset the **statusBar.Text** property to "Ready...".

► **Test the client application**

1. Build and run the application.
2. Do not modify the default values in the text boxes.
3. In the **From** list, click a **Woodgrove Online Bank** entry. In the status bar the URL for this entry will be displayed. Systematically click on each **Woodgrove Online Bank** entry until the URL for your computer is displayed. The URL will be `http://computername/woodgrove/bank.asmx`.
4. In the **To** list, click a **Contoso Micropayments** entry. In the status bar the URL for this entry will be displayed. Systematically check each **Contoso Micropayments** entry until the URL for your computer is displayed. The URL will be `http://computername/contoso/Micropayment.asmx`.
5. Click **Transfer**.
6. Open the Woodgrove and Contoso Account Manager application. Log on as the user shown in the NorthwindClient application.
7. Retrieve the transaction history for the Woodgrove bank.  
Verify that there is a new debit transaction.
8. Retrieve the transaction history for the Contoso micropayment service.  
Verify that there is a new credit transaction.



# Review

- Data Type Constraints
- Performance
- Reliability
- Versioning
- HTML Screen Scraping XML Web Services
- Aggregating XML Web Services

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

1. Which protocol that XML Web services use supports the richest set of data types?

**SOAP**

2. Name the two forms of caching that the .NET Framework provides and that you can use in XML Web services.

**Output and data caching**

3. Name two technologies that you can use to instrument an XML Web service that has been deployed in production.

**WMI and Performance Counters**

4. Which element of an XML Web service should not be versioned?

**The WSDL document**

5. Which protocol should you use if you do not want your XML Web service to be tied to the HTTP protocol?

**SOAP**

6. When should you consider implementing asynchronous Web methods?

**Whenever the method implementation will perform I/O that is based on Win32 kernel handles.**

---

## Module 9: Global XML Web Services Architecture

### Contents

Overview	1
Introduction to GXA	2
Routing and Referral	8
Security and License	16
Review	19
Course Evaluation	20



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, places or events 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, trademarks, 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.

© 2001-2002 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Active Directory, Authenticode, IntelliSense, FrontPage, Jscript, MSDN, PowerPoint, Visual C#, Visual Studio, Visual Basic, Windows NT, and Win32 are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

# Instructor Notes

**Presentation:**  
**30 Minutes**

This module teaches students how to use the security services of the Microsoft® Windows® operating system, Microsoft Internet Information Services (IIS), and the Microsoft .NET Framework and common language runtime to secure XML Web services.

**Lab:**  
**0 Minutes**

After completing this module, students will be able to:

- Describe limitations inherent to the specifications with which today's XML Web services are built.
- Describe the design principles and specifications of Global XML Web services Architecture (GXA).
- Describe XML Web service application scenarios that Web Services Routing Protocol (WS-Routing) and Web Services Referral Protocol (WS-Referral) make possible.
- Explain how to use Web Services Security Language (WS-Security) and Web Services License Language (WS-License) to perform authentication and authorization for XML Web services.
- Design XML Web services that anticipate and can leverage the features that GXA will offer when released.

**Required Materials**

To teach this module, you need the Microsoft PowerPoint® file 2524B\_09.ppt.

**Preparation Tasks**

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

## Module Strategy

Use the following strategy to present this module:

- Introduction to GXA

Ensure that students understand that GXA provides principles, specifications, and guidelines for advancing the protocols of today's XML Web services standards to address more complex and sophisticated tasks in standard ways. Do not imply that it is impossible to implement sophisticated XML Web services by using the current standards.

- Routing and Referral

Explain that as Simple Object Access Protocol (SOAP) has become more widely used as a messaging infrastructure, WS-Referral and WS-Routing have been developed to address some of the issues related to reliable messaging and message routing. Emphasize that these specifications are intended to be building blocks — they are not intended to be complete messaging specifications.

- Security and Licensing

Describe the limitation in the current XML Web services security infrastructure. Explain that XML Signature and XML Encryption make it possible to ensure message integrity and message privacy, but that there is no industry-standard process in place. Explain that WS-Security builds on these standards to specify how to sign and encrypt SOAP messages that are being communicated between an XML Web service and its clients.

Explain that WS-License is an extension to WS-Security that addresses how to represent current license formats and include them in SOAP messages that have been secured according to the WS-Security specification.

# Overview

- Introduction to GXA
- Routing and Referral
- Security and Licensing

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

## Introduction

XML Web services have become the fundamental building blocks for integrating applications as organizations have begun to use distributed computing on the Internet. XML Web services are successful for two reasons: They are based on open standards that make them interoperable, and the technology that is used to implement them is ubiquitous.

The Simple Object Access Protocol (SOAP), the Web Services Description Language (WSDL), and the Universal Description, Discovery, and Integration (UDDI) specifications constitute a set of baseline specifications that provide the foundation for integrating and aggregating applications. But, as organizations develop XML Web services, their solutions have become more complex and their need for standards beyond this baseline has increased. Higher-level functionality such as security, routing, reliable messaging, and transactions in proprietary and often non-interoperable ways becomes increasingly important.

The Microsoft® Global XML Web services Architecture (GXA) provides principles, specifications, and guidelines for advancing the protocols of today's XML Web services standards to address more complex and sophisticated tasks in standard ways.

## Objectives

After completing this module, you will be able to:

- Describe limitations inherent to the specifications with which today's XML Web services are built.
- Describe the design principles and specifications of (GXA).
- Describe XML Web service application scenarios that Web Services Routing Protocol (WS-Routing) and Web Services Referral Protocol (WS-Referral) make possible.
- Explain how to use Web Services Security Language (WS-Security) and Web Services License Language (WS-License) to perform authentication and authorization for XML Web services.
- Design XML Web services that anticipate and can leverage the features that GXA will offer when released.

# Introduction to GXA

- Limitations of XML Web Services
- Overview of Global XML Web Services Architecture
- Global XML Web Services Architecture

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

This section will review some of the limitations of the baseline specifications (SOAP, WSDL, and UDDI) and the limitations of XML Web services that are implemented with them, before presenting an overview of GXA.



## Limitations of XML Web Services

- **XML Web services are used today to**
  - Integrate enterprise applications
  - Interoperate with key partners
- **Tomorrow's XML Web services need to**
  - Interoperate across multiple organizations
- **Problems that baseline standards (SOAP, WSDL, UDDI) do not solve**
  - How do you generically secure an XML Web service?
  - How do you dynamically add or remove a message path?
  - How do you provide reliability over HTTP while supporting long-running XML Web service requests in a scalable way?
  - How do you conduct long-running transactions?

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

Solutions for XML Web services are being developed to support increasingly sophisticated business processes. The development of XML Web services consists of three tiers:

- **Tier 1: Enterprise Application Integration**

Organizations initially use XML Web services to integrate internal applications. XML Web services allow them to expose legacy applications to business applications in heterogeneous environments without having to rewrite significant amounts of code.

- **Tier 2: Interoperability with Key Partners**

The next step for most organizations is to integrate one or two key partners outside of the organization.

- **Tier 3: Interoperability across Multiple Organizations**

As XML Web services solutions become more global in reach and capacity — and therefore more sophisticated — it becomes increasingly important to provide additional capabilities to ensure global availability, reliability, and security.

Tiers 1 and 2 are available today, but tier 3 is dependent on technology and specifications that are still emerging.

## Limitations

Using XML Web services, organizations can extend the benefits of integrating systems within organizations outward to partners and customers. However, the lack of broadly-adopted specifications for security, routing, and other necessary capabilities limits integration to those scenarios where bilateral, out-of-band agreements can be negotiated and maintained. To ensure secure, reliable cross-organization interoperability, developers are forced to implement a series of solutions that are specific to the situation. For example, you may have to configure a custom authentication solution that adds a user name and password to SOAP headers and then uses a SOAP extension to encrypt and decrypt this sensitive information. This solution requires that the encryption extension assembly or, minimally, the encryption algorithm be distributed to all clients of the XML Web service. Although this solution is adequate, clearly it is not optimally secure and requires close cooperation between the XML Web service implementer and client developers.

As the business requirements that drive XML Web services become more complex, developers require additional capabilities that current XML Web services standards do not address. These capabilities include the following:

- *Security*. Developers need a straight-forward, end-to-end security architecture to implement across organizations and trust boundaries.
- *Routing*. Developers need a way of specifying messaging paths and the ability to configure those message paths dynamically to ensure scalability and fault-tolerance.
- *Reliable Messaging*. Developers need an end-to-end guarantee of message delivery across a range of semantics such as, at-least-once, at-most-once, and exactly once.
- *Transactions*. Developers need flexible process and compensation-based transaction schemes to execute transactions across organizations.

If an organization does not use these capabilities, it can be exposed to risks and the value of its XML Web services will degrade. Currently, in the absence of these capabilities in XML Web services, developers must create specific solutions for each situation, which is time consuming and expensive. In addition, specific solutions encroach upon a central value area of XML Web services — cross-organizational interoperability.

## Overview of Global XML Web Services Architecture

### ■ Design tenets of GXA

- General-purpose
- Federated
- Modular
- Standards-based

### ■ Released specifications (October 2001)

- WS-Routing
- WS-Security
- WS-Referral
- WS-License

### ■ Future

- Reliability
- Transactions

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

Microsoft and IBM presented an architectural sketch for the evolution of XML Web services at the World Wide Web Consortium (W3C) Workshop on Web Services in April 2001. This sketch was the forerunner of GXA. GXA provides principles, specifications, and guidelines for advancing the protocols of today's XML Web services standards to address more complex and sophisticated tasks in standard ways.

### Design tenets of GXA

GXA is based on four design tenets:

- *General Purpose.* GXA is designed for a wide range of XML Web services scenarios, ranging from business-to-business and Enterprise Application Integration (EAI) solutions to peer-to-peer applications and business-to-consumer services.
- *Modular.* GXA uses the extensibility of the SOAP specification to deliver a set of composable modules that you can combine as needed to deliver end-to-end capabilities. As your system requires new capabilities, you can create new modular elements.
- *Federated.* GXA is fully distributed and designed to support XML Web services that cross organizational and trust boundaries and requires no centralized servers or administrative functions.
- *Standards-Based.* Similar to previous XML Web services specifications, GXA protocols will be submitted to the appropriate standards bodies.

### Released specifications

This section describes a set of the new Global XML Web services specifications that were made available in October 2001. These specifications represent a significant step toward a comprehensive Global XML Web services Architecture.

- WS-Routing is a simple, stateless SOAP extension for sending SOAP messages in an asynchronous manner over a variety of communication transports such as Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Hypertext Transfer Protocol (HTTP).

- WS-Referral is a simple SOAP extension that enables the routing between SOAP nodes on a message path to be dynamically configured.
- WS-Security provides a security language for XML Web services. WS-Security describes enhancements to SOAP messaging, providing three capabilities: credential exchange, message integrity, and message confidentiality. Message integrity is provided by taking advantage of XML Signature and licenses to ensure that messages are transmitted without modifications. Similarly, message confidentiality takes advantage of XML Encryption and licenses to keep portions of SOAP messages confidential.
- WS-License describes how to use several common license formats, including X.509 certificates and Kerberos tickets, as WS-Security credentials. WS-License includes extensibility mechanisms that enable new license formats to be easily incorporated into the specification.

### Future protocols

Interactions across organizations have many opportunities for failure ranging from transmission errors to incompatible or unavailable business processes. The following future protocols will allow the builders of XML Web services to manage the scope and effect of failures.

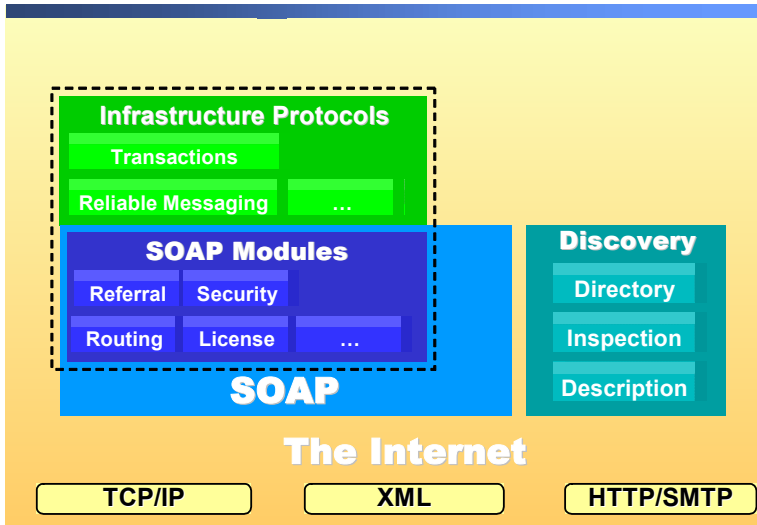
- **Reliable messaging**

XML Web services need to operate reliably over intranets and the public Internet, and over transport protocols that are not completely reliable. For example, HTTP, the most commonly used XML Web service transport protocol, provides no mechanisms to ensure that a request was received or that the client received a response. Although lower network-level protocols can alert a client in case of common catastrophic failures like resource not found, a SOAP-level reliable messaging protocol can provide delivery guarantees. Using a protocol that provides delivery guarantees isolates application processes from the detailed handling of transmission failure and its recovery, allowing a developer to concentrate on automating a process with a much-simplified error handling model. In the exchange of messages, individually or as part of a long-running process, communicating parties will be able to obtain end-to-end delivery guarantees so that messages will not be lost, duplicated, or delivered in the wrong order.

- **Transactions**

Transactions address the possibility of a business-level inability to complete a process. Transactions allow multiple parties that are involved in a process to arrive at a consistent final outcome (or discover that this is not possible). Although existing two-phase commit protocols are appropriate in some circumstances, developers also need more loosely-coupled techniques, such as exceptions and compensation, which enable a broader range of transactions to be automated across trust boundaries. Developers will have powerful process-modeling languages to express the patterns of messages that are exchanged between XML Web services, the interactions of those messages, and the business processes that they reflect, including both normal and exceptional conditions.

## Global XML Web Services Architecture



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

This graphic gives an overview of the Global XML Web services Architecture.

### SOAP and SOAP modules

Primarily, SOAP defines a framework for message structure and a message processing model. SOAP provides the foundation for a wide range of composable modules and protocols running over a variety of underlying protocols such as HTTP. SOAP modules take advantage of this extensibility to provide composable building blocks that are suitable for building the higher-level capabilities. Because the WS-Security, WS-License, WS-Routing, and WS-Referral SOAP modules are modular, you can use them together. For example, WS-Security describes how to digitally sign SOAP messages that use a WS-Routing header. Each of these specifications provides extension and composition mechanisms that enable future specifications for the Global XML Web services Architecture to be incorporated into a complete solution.

The generality, breadth, and uniformity of SOAP modules allow a wide range of services to take advantage of the XML Web services-enabled network infrastructure, which includes routers, switches, proxies, caches, and firewalls.

### Infrastructure protocols

The reliable infrastructure protocols for messaging and transactions build on SOAP modules to provide end-to-end functionality. Protocols at this layer tend to have semantically-rich finite state machines as part of their definition. They maintain state across a sequence of messages and may aggregate the effect of many messages to achieve a higher-level result.

# Routing and Referral

- **WS-Routing and WS-Referral**
- **Routing Scenario**
- **Referral Scenario**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

As SOAP messaging evolves into a general-purpose Global XML Web services Architecture, there must be a means of addressing and transmitting SOAP messages over various types of communications systems. Using several types of communications systems enables a wide range of communication patterns such as peer-to-peer or store-and-forward networking. It also allows messages to be efficiently routed to distributed processing nodes. The WS-Routing and WS-Referral specifications support these features.

This section explains the key features of these specifications and gives two scenarios demonstrating how to use these specifications.

In this section, you will learn about application architectures for XML Web services that WS-Routing and WS-Referral make possible.

## WS-Routing and WS-Referral

- **SOAP message processing model**
  - SOAP provides a distributed processing mechanism via SOAP:actor
  - Does not define a message path
- **WS-Routing**
  - Routes messages across intermediate SOAP nodes to enable asynchronous messaging via a message path
  - Provides transport and route flexibility
    - Supports message forwarding, resolution, and reverse path routing
- **WS-Referral**
  - Enables route configuration to provide configuration of SOAP routers
    - Describe a referral
    - Query a SOAP router for a referral
    - Insert, delete, and exchange referrals

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

Intermediaries are central to the SOAP message model. SOAP provides a distributed processing mechanism in which you can use the SOAP **SOAP:actor** attribute to indicate which part of a message is intended for a given SOAP receiver.

### The SOAP message processing model

Despite the *implied* SOAP message model, SOAP does not define a mechanism for indicating the SOAP senders and receivers along the SOAP message path or the order in which the senders and receivers are composed. In short, SOAP does not define a message path.

To provide the semantics for actually exchanging messages, you can bind SOAP to other application layer protocols, such as HTTP and Simple Mail Transfer Protocol (SMTP). However, these protocols define their own message path models and message exchange patterns, which differ from the general SOAP model. As a result, it is not possible to use these protocol bindings alone to describe an exchange of a SOAP message from point A to point B.

### WS-Routing and WS-Referral and the SOAP message processing model

WS-Routing, on the other hand, defines a message path model that is fully compatible with the SOAP message processing model. In other words, WS-Routing makes it possible to describe the complete exchange of a SOAP message from point A to point B and to describe which parts of a SOAP message are intended for which SOAP receiver in the message path.

WS-Referral builds on a simple model of a SOAP router being able to delegate a Uniform Resource Identifier (URI) space or a part thereof to another SOAP router by manipulating its routing entries. By changing its configuration, a SOAP router can learn about other SOAP routers and as a result affect the message path of any given SOAP message passing through it. By controlling the amount of information that a given SOAP router has, it is possible to build a variety of routing configurations that can support scenarios.

### WS-Routing

With WS-Routing, you can describe the entire message path for a SOAP message (in addition to its return path) directly within the SOAP envelope. WS-Routing supports one-way messaging, two-way messaging, such as request/response and peer-to-peer conversations, and long-running dialogs.

**WS-Routing mechanisms**

The purpose of WS-Routing is to define the mechanisms that are needed to describe messages that are being exchanged along the following two message paths:

- A forward message path where messages travel from the initial sender through zero or more intermediaries to the ultimate receiver.
- An optional reverse message path where messages travel in the direction from the ultimate receiver through zero or more intermediaries to the initial sender.

In addition, being able to identify a message and to correlate that message with other messages is essential to WS-Routing. The correlation can for example be between multiple messages flowing in the same direction on either the forward or the reverse message path or it can be between messages on different message paths. An example of a correlation is between a WS-Routing fault message and the faulty message. WS-Routing defines a correlation mechanism.

**WS-Routing elements**

WS-Routing defines a single new SOAP header and associated processing model. To illustrate this, consider a SOAP processor A that wishes to send a SOAP message to an ultimate receiver D via B and then via C.

To express such routes, WS-Routing defines a new SOAP header named path, and, within that header, defines:

- A **<from>** element for the message originator (A).
- A **<to>** element for the ultimate receiver (D).
- A **<fwd>** element to contain the forward message path.
- A **<rev>** element to contain the reverse message path.

WS-Routing defines the **<rev>** element to enable two-way messaging exchange. Both **<fwd>** and **<rev>** contain **<via>** elements to describe each intermediary (B and C). Other elements are defined for message identification, correlation, and intent.

Note that there is no requirement that A knows the complete path in advance; the path may be discovered dynamically.

As a message moves along a path, each hop along the way moves its corresponding **<via>** element from the fwd path to the rev path, dynamically constructing a path back to the sender. Other processing details cover gateways and routing-specific SOAP faults.

**WS-Referral**

WS-Referral is complementary to WS-Routing in that WS-Referral provides a way to configure how SOAP routers will build a message path, whereas WS-Routing provides a mechanism for describing the actual path of a message.

WS-Referral aids in the configuration of message paths, which in turn enables a variety of services. In addition to relay services such as high performance overlay message delivery or corporate firewall services, SOAP routers can provide XML Web services like load-balancing, mirroring, caching, and client authentication services. As an example, an XML Web service may delegate responsibility for some aspects of its service to third parties in a manner that is transparent to users of that service. The goal of WS-Referral is to provide the mechanisms that are needed to enable SOAP applications to insert, delete, and query routing entries in a SOAP router through the exchange of referral information.



**Referral mechanisms**

The basic unit in WS-Referral is the referral statement. Referral statements are exchanged via three mechanisms:

- *Register messages*. This SOAP message instructs the receiver to utilize the enclosed referral statement. The recipient explicitly accepts or rejects the registration.
- *Query messages*. A SOAP router can be queried for referrals with query messages. You may extend this message to allow complex queries.
- *Referrals header*. You can augment any SOAP message with a referrals SOAP header containing referral statements. A referrals header allows a mechanism to attach referrals on existing message exchange streams.

An administrator typically uses register and query messages to configure a set of SOAP routers on a path. Register provides a push method of updating the SOAP level routing tables in a router. Query methods provide a pull method, which a router may use to learn about message paths.

You typically use the referrals SOAP header in dynamic environments, where updates to message paths are cached. As part of an existing message exchange, a SOAP router would include a referrals SOAP header to indicate to a sender a better path to reach a desired SOAP actor.

**WS-Referral elements**

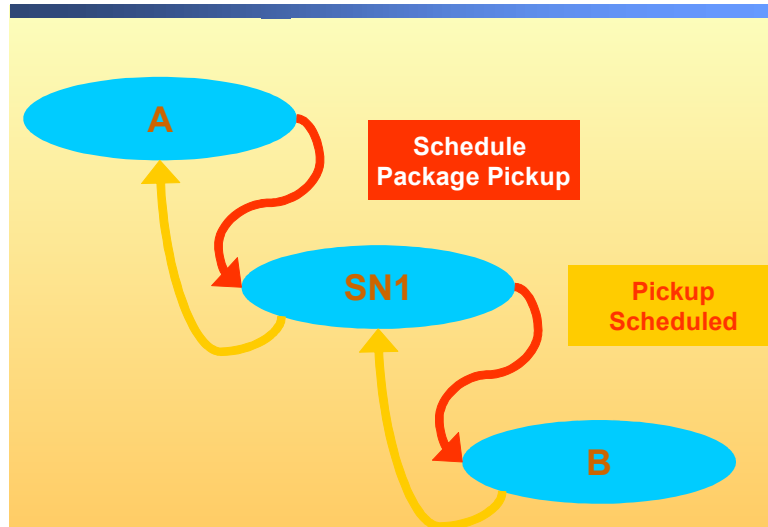
The referral statement has five elements:

- **<for>**  
Indicates the URIs (also called SOAP actors) for which the referral is intended.
- **<if>**  
This element is a set of conditions that the recipient of the referral must understand to use the referral.
- **<go>**  
If a SOAP message is headed for a SOAP actor (for which this referral is intended) and the set of conditions are fulfilled, then you send the message via one of the SOAP routers listed in the **<go>** element.
- **<desc>**  
Additional information that the recipient does not need to understand to use the referral, but may find useful.
- **<refId>**  
A unique identifier for a referral so that it is possible to identify a specific representation of a referral.

While occasionally grouped for convenience, you can evaluate each referral statement independently of any other referral.

When interpreting a referral statement, the **<for>** element gives the list of SOAP actors for which this referral applies. Consider the **<if>** clause as the set of conditions to satisfy when applicable. If satisfied, then go (re-route) to one of the SOAP routers listed in the **<go>** element.

## Routing Scenario



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

This topic illustrates using WS-Routing to send a SOAP message from A to B via the intermediary SOAP node SN1. The `<rev>` element is present in the message, so the reverse message path is built dynamically when the message travels along the forward message path.

### STEP 1: From A to SN1

The following example shows the SOAP message leaving initial WS-Routing sender A in forward direction towards B with a reverse path. In the following code sample, notice the bolded code where the `<path>` header gives the destination B in the `<to>` element and the source A in the `<from>` element, indicating with the `<fwd>` element that SN1 must be used as an intermediary.

```
<S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  <S:Header>
    <p:path xmlns:p="http://schemas.xmlsoap.org/rp/">
      <p:action>http://example.org/alert</p:action>
      <p:to>soap://serverB.com/B</p:to>
      <p:fwd>
        <p:via>soap://SN1.com</p:via>
      </p:fwd>
      <rev>
        <p:via>soap://serverA.com/A</p:from>
      </rev>
      <p:from>soap://serverA.com/A</p:from>
      <p:id>uuid:12dfs83476-4asd-s234-s3df-d25656adssf4</p:id>
    </p:path>
  </S:Header>
  <S:Body>...
</S:Body>
</S:Envelope>
```

**STEP 2: From SN1 to B**

The following code example shows the SOAP message leaving intermediary SN1 in a forward direction towards B with a reverse path. Notice in the bolded code that SN1 has removed itself (in the **<via>** element) from the **<fwd>** element, and added itself to the reverse path that is given by the **<rev>** element to specify the reverse path. The **<to>** and **<from>** elements are not modified.

```
<S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  <S:Header>
    <p:path xmlns:p="http://schemas.xmlsoap.org/rp/"
      <p:action>http://example.org/alert</p:action>
      <p:to>soap://serverB.com/B</p:to>
      <p:fwd>
        </p:fwd>
      <rev>
        <p:via>soap://SN1.com</p:via>
        <p:via>soap://serverA.com/A</p:via>
      </rev>
      <p:from>soap://serverA.com/A</p:from>
      <p:id>uuid:12dfs83476-4asd-s234-s3df-d25656adssf4</p:id>
    </p:path>
  </S:Header>
  <S:Body>...
</S:Body>
</S:Envelope>
```

**STEP 3: From B to SN1**

The following code example shows the SOAP message leaving B towards intermediary SN1. Notice in the bolded code that the **<fwd>** element gives the reverse path.

```
<S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  <S:Header>
    <p:path xmlns:p="http://schemas.xmlsoap.org/rp/"
      <p:action>http://example.org/alert</p:action>
      <p:fwd>
        <p:via>soap://SN1.com</p:via>
        <p:via>soap://serverA.com/A</p:from>
      </p:fwd>
      <p:from>soap://serverB.com/B</p:from>
      <p:id>uuid:74583476-45gd-sg6g-sf54-dfgsgfgdssf4</p:id>
    </p:path>
  </S:Header>
  <S:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pickup scheduled</m:msg>
    </m:alert>
  </S:Body>
</S:Envelope>
```

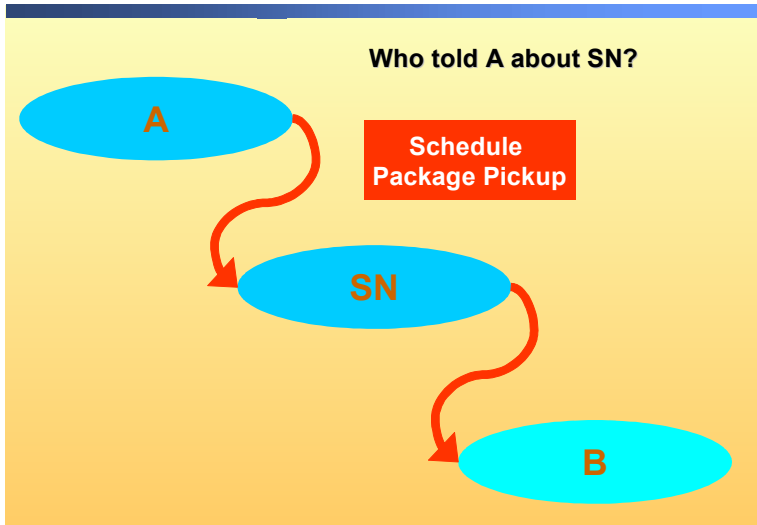
**STEP 4: From SN1 to A**

The following code example shows the SOAP message leaving intermediary SN1 towards A. Notice in the bolded code that the **<fwd>** element gives the remainder of the reverse path.

```
<S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <p:path xmlns:p="http://schemas.xmlsoap.org/rp/">
      <p:action>http://example.org/alert</p:action>
      <p:fwd>
        <p:via>soap://serverA.com/A</p:from>
      </p:fwd>
      <p:from>soap://serverB.com/B</p:from>
      <p:id>uuid:74583476-45gd-sg6g-sf54-dfgsgfgdssf4</p:id>
    </p:path>
  </S:Header>
  <S:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Pickup scheduled</m:msg>
    </m:alert>
  </S:Body>
</S:Envelope>
```

You are not required to specify a reverse path or every intermediary in the reverse path if the underlying protocol is bidirectional. In this case it is assumed that the underlying protocol provides a bidirectional communication channel.

## Referral Scenario



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

The interaction between WS-Referral and WS-Routing provides a straightforward example of how to use WS-Referral. Notice in the following code example for a referral registration that the bolded code would cause the sender of a WS-Routing message bound for the `soap://ServerB.com` service to add an additional forward path `<via>` element to `soap://SN1.com`.

```
<S:Envelope xmlns:S="http://www.w3.org/2001/09/soap-envelope">
  <S:Header>
    <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
      ...
    </m:path>
  </S:Header>
  <S:Body>
    <r:register
      xmlns:r="http://schemas.xmlsoap.org/ws/2001/10/referral">
      <r:ref>
        <r:for>
          <r:exact>soap://serverB.com/B</r:exact>
        </r:for>
        <r:if>
          <r:ttl>43200000</r:ttl>
        </r:if>
        <r:go>
          <r:via>soap://SN1.com</r:via>
        </r:go>
        <r:refId>uuid:09233523-345b-4351-b623-
          5dsf35sgs5d6</r:refId>
      </r:ref>
    </r:register>
  </S:Body>
</S:Envelope>
```

Note that the `<ttl>` element is a required if condition — it gives a time-to-live or lifespan of the referral request.

## Security and License

- **WS-Security and WS-License**
- **Security Factoring and Authorization Scenario**

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

WS-Security describes how to use the existing W3C security specifications, XML Signature and XML Encryption, to ensure the integrity and confidentiality of SOAP messages. And together with WS-License, it describes how to securely associate existing digital credentials and their associated trust semantics with SOAP messages. Together, these specifications form the bottom layer of a comprehensive, modular, security architecture for XML Web services. Future security specifications will build on these basic capabilities to provide mechanisms for credential exchange, trust management, revocation, and other higher-level capabilities.

In this section, you will learn how to use WS-Security and WS-License to perform authentication and authorization for XML Web services.

## WS-Security and WS-License

### ■ WS-Security

- Credential exchange, message integrity, and message confidentiality to enable secure XML Web services that authenticate and authorize incoming requests
  - Support existing industry-standard identity and rights management systems
  - Work across trust domains

### ■ WS-License

- Defines encoding for common license formats with modular and extensible structures
  - Examples include: X.509, Kerberos, SAML, XrML
  - Extension to WS-Security

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

WS-Security and WS-License are two specifications that be used together to specify how to transmit data transmitted between clients in a secure manner. They also specify how to ensure that data remains confidential and that sensitive information is not compromised during transfer.

### WS-Security

WS-Security provides a language to secure XML Web services. WS-Security describes enhancements to SOAP messaging consisting of three capabilities: credential transfer, message integrity, and message confidentiality. These capabilities by themselves do not provide a complete security solution; WS-Security is a building block that you can use in conjunction with other XML Web service protocols to address a wide variety of application security requirements.

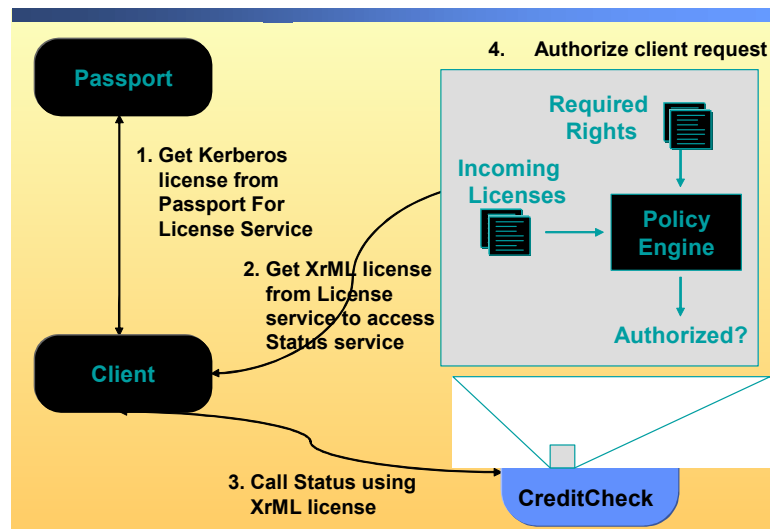
WS-Security provides a general-purpose mechanism for associating licenses (credentials that are signed assertions, for example, X.509 certificates or Kerberos tickets) with messages, although no specific format is required.

Message integrity is provided by using XML Signature and licenses to ensure that messages have originated from the appropriate sender and were not modified in transit. Similarly, message confidentiality takes advantage of XML Encryption and licenses to keep portions of a SOAP message confidential.

### WS-License

WS-License describes a set of commonly used license types (credentials that are signed assertions) and describes how they can be placed within the WS-Security <credentials> tag. Specifically, the WS-License specification describes how to encode X.509 certificates and Kerberos tickets. The WS-License specification also supports the Security Assertion Markup Language (SAML), and the eXtensible rights Markup Language (XrML). In addition to how to include opaque encrypted keys. WS-License includes extensibility mechanisms that you can use to further describe the characteristics of the licenses that are included with a message.

## Security Factoring and Authorization Scenario



Subsequent calls to Status service do not require presentation of Passport licenses

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

### Introduction

This graphic provides a scenario that uses the WS-Security and WS-License specifications to perform authentication and authorization.

### Detailed steps for performing authentication

1. The client obtains a Kerberos license from Microsoft Passport for the License XML Web service.
2. The client uses the Kerberos license to obtain an XrML (Extensible Rights Markup Language) license from the License XML Web service to access the Status service.
3. The client calls Status XML Web service by using the XrML license.
4. The Status XML Web service uses the XrML license embedded in the SOAP header to authorize the client CreditCheck request against its own policy engine. If the request is authorized, the CreditCheck resource is accessed.



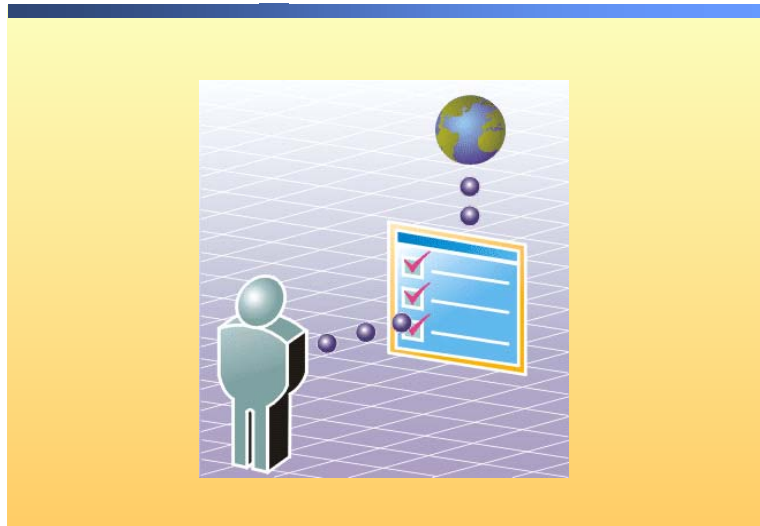
# Review

- Introduction to GXA
- Routing and Referral
- Security and Licensing

\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

1. What are some of the design characteristics of GXA?  
**General-purpose, modular, federated, and standards-based**
2. What does WS-Referral enable?  
**A way to configure how SOAP routers will build a message path.**
3. What does WS-Routing add to the existing XML Web service infrastructure?  
**A mechanism for describing an actual path of a message.**
4. What aspects of security does the WS-Security specification address?  
**Credential exchange, message integrity, and message confidentiality**

## Course Evaluation



\*\*\*\*\*ILLEGAL FOR NON-TRAINER USE\*\*\*\*\*

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.