



Microsoft WS-I Basic Security Profile 1.0 Sample Application

Preview release for the .NET Framework version 1.1



patterns & practices

Microsoft WS-I Basic Security Profile 1.0 Sample Application

Preview release for the .NET Framework version 1.1

patterns & practices

Jason Hogg, Microsoft Corporation
Hernan de Lahitte, Lagash Systems SA
Diego Gonzalez, Lagash Systems SA
Pablo Cibraro, Lagash Systems SA
Pete Coupland, VMC Consulting Corporation
Mrinal Bhao, Infosys Technologies Ltd
Paul Slater, Wadeware LLC

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.

© 2005 Microsoft Corporation. All rights reserved.

Microsoft, MS-DOS, Windows, Windows NT, Windows Server, Visual Studio, Visual C#, and Visual Basic 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

Chapter 1

Introduction	1
Who Should Read This Guide	2
Installation Prerequisites	2
How to Use This Guide	2
Chapter 1 — Introduction	2
Chapter 2 — Installing the Sample Application	3
Chapter 3 — Sample Application Walkthrough	3
Chapter 4 — Sample Application Architecture.	3
Chapter 5 — Policy Usage in the Sample Application	3
Chapter 6 — Designing Web Services for Interoperability and Resilience	3
Chapter 7 — Designing an Interoperable Web Service Using the WS-I BSP 1.0	3
Appendix A — Enterprise Library Integration	4
Appendix B — Sample Application Retailer Service Messages.	4
The Web Services Interoperability Organization (WS-I)	4
WS-I Deliverables	5
WS-I Profiles	5
Existing WS-I Profiles.	7
Web Services Security Specifications	8
The WS-I Basic Security Profile.	9
Secure SOAP Messaging	10
Summary	10
Additional Information	10

Chapter 2

Installing the Sample Application	11
Installing the Sample Application	11
Installing and Running the Application on a Single Computer.	11
Installing the Application in a Two-Computer Environment	13
Post-Installation Configuration	17
Modifying Configuration Files	17
Application Pools.	18
Enabling SSL on the Presentation Tier	18
Verifying the Installation.	19
IIS Virtual Directories	19
SQL Server 2000 SP3a Databases and Tables	20
X.509 Certificates.	20

Uninstalling the Application	21
Uninstalling the Application on a Single Computer	21
Uninstalling the Application in a Two Computer Environment	21
Summary	21

Chapter 3

Sample Application Walkthrough 22

Sample Application Control Flow	22
Walkthrough	25
Demo Configuration Page	26
Shopping Cart Page.	28
The Order Status Page	31
Summary	32

Chapter 4

Sample Application Architecture 33

Supply Chain Management Architecture	33
Requirements View	33
Conceptual View	37
Implementation View.	41
Deployment View.	44
Demo System	44
Retailer and Warehouse Systems.	45
Manufacturer System	45
Sharing Configuration Between Web Services	45
Summary	47

Chapter 5

Policy Usage in the Sample Application 48

Policy Assertions.	49
Policy Mappings	49
Custom Assertions	51
ConfidentialityEx Custom assertion	51
Custom Token Managers	52
X.509 Security Token Manager.	52
UsernameToken Manager	54
WSE Policy Advisor	54
Format of the Report.	55
Part of a Typical Report	56
Summary	57

Chapter 6**Designing Web Services for Interoperability and Resilience 58**

Designing Web Services for Interoperability	59
Contract-First Development	59
WS-I Test Tools	65
Basic Security Profile 1.0 Test Tools	66
Designing a Web Service for Resiliency	66
WSE 2.0, WSE 3.0, and Indigo Interoperability	67
Separation of Concerns	67
Summary	71

Chapter 7**WS-I BSP Interoperability Guidance 72**

Basic Security Profile 1.0 Guidance	72
Specifying Security Requirements	73
Exchanging Security Tokens	74
Interoperability Considerations for the Sample Application	75
X509 Certificates	75
Verification of WS-Addressing Headers	76
Mapping Security Policy to a Web Service Operation	76
UsernameToken	78
Message Age and Clock Skew	78
Basic Security Profile Detailed Review	79
Summary	79

Appendix A**Enterprise Library Integration 80**

Application Blocks	80
Solution Structure and Application Block Projects	81
Configuration Console	81
Configuration Application Block	82
Sample Application Configuration Information	83
Exception Handling Application Block	86
Web Client	87
Retailer, Warehouse, and Manufacturer Data Access Component Layers	88
Manufacturer Service	88
Custom Policy Assertion	88
SOAP Server Exceptions	88
Logging and Instrumentation Application Block	89
Enabling Logging and Tracing	89
Web Client and Services	90
Data Access Application Block	91
Web Client	91
Retailer, Warehouse, and Manufacturer Services	91

Appendix B

Sample Application Retailer Service Messages	92
Defining XML Namespaces	93
Using the Retailer Namespace	94
Using the RetailOrder XML Schema	96
Mapping Simple XML Types	98
Specifying Literal Usage	98
Specifying Formatting Style	99
Specifying Parameter Style	99
 Contributors	 101
 Additional Resources	 103

1

Introduction

Welcome to the Microsoft *Basic Security Profile (BSP) 1.0 Sample Application Guide*. This guide describes the design and implementation of the WS-I Basic Security Profile 1.0 Sample Application: Preview release for the .NET Framework version 1.1 (hereafter referred to as the Sample Application). Sample applications are currently being developed by WS-I members to test interoperability of secure Web services.

The preview release is not an official deliverable from the WS-I. Instead, it is intended to provide Microsoft customers early access to guidance for securing interoperable Web services based on the WS-I Basic Security Profile Working Group Draft dated January 20, 2005. The Sample Application was developed using Visual Studio 2003 and Microsoft Web Services Enhancements (WSE) 2.0. The version of the Sample Application accompanying the final release of the BSP 1.0 is expected to be implemented using Visual Studio 2005 and WSE 3.0.

This guide describes the design and implementation of the Sample Application and includes information on the:

- Design considerations.
- Design of services to provide interoperability and resilience to change.
- Manner in which the application uses WSE 2.0 to provide interoperable secure Web services based on the WS-I BSP 1.0.
- Process of installing and using the Sample Application.
- Factors to consider when developing and implementing your own secure interoperable Web services.

The main focus of this guide and the accompanying application is demonstrating Web service interoperability using the BSP and WS-Security. While every effort has been taken to abide by Microsoft security guidance, there are several security considerations and functional assumptions that you should be aware of when examining the application. For more details, see the Sample Application release notes.

Note: Microsoft will also be releasing regular builds of the WSE 3.0 version of the Sample Application to the GotDotNet workspace when WSE 3.0 is available in beta. Microsoft is also looking for more information from customers about the kind of guidance they would like to see to supporting a sample application based on Web service security interoperability and interoperability in general. To provide feedback, see “Microsoft WS-I Basic Security Profile Sample Application: Workspace Home” at <http://go.microsoft.com/fwlink/?LinkId=47780>.

Who Should Read This Guide

This guide is targeted at developers and architects who need to design and implement interoperable Web services with message layer security. By reading this guide and examining the Sample Application in detail, you can gain an understanding of how conforming to the BSP 1.0 helps you to create secure Web services that are interoperable.

Installation Prerequisites

This guide will provide you with useful information about the Microsoft version of the Sample Application. However, to get the most out of this guide, it is useful to have the application installed. To install the application on a single computer, you must have the following software installed:

- Microsoft Windows 2000, Windows XP, or Windows Server 2003
- Microsoft Visual Studio .NET 2003, Enterprise Architect or Enterprise Developer edition
- Microsoft Web Services Enhancements (WSE) 2.0 SP3
- Microsoft SQL Server 2000a SP3a or Microsoft SQL Server 2000 Desktop Engine (MSDE 2000) Release A

For more information about installing the Sample Application, see Chapter 2, “Installing the Sample Application.”

How to Use This Guide

This book is divided into seven chapters and two appendices. The next sections describe each of these.

Chapter 1 — Introduction

This chapter discusses the WS-I and the deliverables they produce. It examines the requirements for Web services security and the Web services security specifications defined by standards organizations such as OASIS, W3C, and IETF. It then provides an overview of the BSP 1.0.

Chapter 2 — Installing the Sample Application

This chapter discusses how to install the Sample Application. It also examines how the Windows Installer package configures the environment and describes how you can modify this configuration for your own requirements.

Chapter 3 — Sample Application Walkthrough

This chapter examines the flow of the application and guides you through a walkthrough of the application's functionality.

Chapter 4 — Sample Application Architecture

This chapter examines how the WS-I Sample Application Working Group specification for the Supply Chain Management (SCM) application was implemented using Microsoft .NET technologies and WSE 2.0. It discusses the use of various architectural patterns in the design of the application and shows how the security configuration of the application can be specified declaratively by implementing the functionality within Microsoft Web Services Enhancements (WSE) 2.0 toolkit.

Chapter 5 — Policy Usage in the Sample Application

This chapter examines how the Sample Application uses policy. It discusses the WSE Policy Advisor, an unsupported tool from Microsoft Research that examines policy files and generates a report to make security recommendations.

Chapter 6 — Designing Web Services for Interoperability and Resilience

This chapter examines the design considerations for creating interoperable Web services, including general recommendations for using XML Schema (XSD) and Web Services Description Language (WSDL). It also discusses how to ensure that your Web services are resilient to changes from within products.

Chapter 7 — Designing an Interoperable Web Service Using the WS-I BSP 1.0

This chapter provides guidance on designing interoperable Web services according to the Basic Security Profile (BSP). The chapter also provides guidance for designing your Web services so they are more resilient to change from within products. It examines how to expose and consume Web services and discusses known issues that could affect interoperability.

Note: Several interoperability issues currently exist when mapping security policy to an operation on a service. In its current form, the manner in which the Sample Application's security policy is being mapped to an operation is not proving to be interoperable. At this time, the WS-I is discussing the recommended approach to solving these issues. This chapter has more information describing the specific issue and recommended workarounds — one of which will likely be implemented in the final version of the Sample Application. We will also provide updates on the [Community Workspace](#).

Appendix A — Enterprise Library Integration

Appendix A describes how the Microsoft *patterns and practices* Enterprise Library was used to provide functionality within the application.

Appendix B — Sample Application Retailer Service Messages

Appendix B demonstrates how to analyze the Sample Application messages, using the Retailer Web service as an example.

The Web Services Interoperability Organization (WS-I)

Web services provide a platform for application integration in which the applications are adapted to the Web and use Web-based standards. Currently, both software and hardware vendors are rushing Web services products to market. The widespread adoption of core standards, such as XML, SOAP, WSDL, and WS-Security, represents a significant breakthrough in the industry. Applications can now be built using a combination of platform-independent Web services from multiple suppliers. However, for Web services to truly fulfill their promise, they should be interoperable. An interoperable Web service is one that can be called by other platforms.

Interoperability requires consensus, a clear understanding of requirements, and adherence to specifications. In response to these needs, industry members formed the Web Services Interoperability Organization (WS-I), an industry consortium that takes a higher-level view of Web services and determines how to fit together the various specifications and standards to get them to work together in a heterogeneous landscape. The goal is to provide and promote a common vision among Web service software vendors that inspires customer confidence and ensures Web services' continued adoption and evolution.

Note: For more information, see the WS-I Web site at <http://www.ws-i.org>.

The goals of the WS-I include:

- Providing education and guidance that will further the adoption of Web services.
- Promoting consistent and reliable practices that will help developers write Web services that are interoperable across platforms, applications, and programming languages.
- Articulating and promoting a common industry vision for Web services interoperability to ensure that Web services evolve in a systematic, coherent way.

WS-I Deliverables

To achieve its goals, the WS-I provides three types of deliverables:

- **Profiles.** These are documents that contain lists of named and versioned Web services specifications, along with implementation and interoperability guidelines that recommend how the specified components should be used together to develop interoperable Web services.
- **Test tools.** These are responsible for testing Web service artifacts, such as WSDL files and SOAP messages, to ensure conformance with particular profiles.
- **Sample applications.** These are vendor-specific implementations of a sample application, based on a common set of use cases and usage scenarios specific to a particular profile. Implementations are designed to demonstrate interoperability across platforms.

The WS-I also produces supporting documentation in conjunction with each of these deliverables.

WS-I Profiles

Web service profiles collect key Web service specifications into meaningful groups and simplify the implementation of interoperable applications. By demonstrating how each of these specifications relates to the others, profiles also promote the adoption of those standards. The WS-I plans to constantly improve the scope and definition of the profiles to reflect the demands of the market.

This section examines the common characteristics that are found in WS-I profiles, such as the Basic Security Profile.

Note: For the complete list of guiding principles, see the “Basic Security Profile Working Group Draft” document on the WS-I Web site at <http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0-2005-01-20.html>.

Scope of the Profile

The *scope* of the profile is defined by a group of specifications at particular version levels. Specifications often provide ways of extending the components they describe to increase their capabilities. For example, the SOAP 1.1 specification states that typical examples of extensions that can be implemented as header entries are authentication, transaction management, and payment methods. These extensions provide support for additional protocols to be developed, as required.

Level of Granularity

The profile's Web services layer relies on well-understood lower-layer protocols such as TCP/IP and Ethernet. They assume that protocols such as SSL/TLS and HTTP are also well understood and mention them specifically only when there is an issue that affects Web services. WS-I does not intend to address all issues, only those that are related to Web services interoperability.

Strength of Requirements

Requirements are the rules that any Web service must adhere to in order to meet the profile's standards for interoperability. Each requirement is individually identified by the letter "R" and a number, such as the following example:

R3029 A **SECURITY_TOKEN** named *wsse:BinarySecurityToken* **MUST** specify an *EncodingType* attribute

Profiles make strong requirements (for example, **MUST** and **MUST NOT**) wherever appropriate. If there are times when a strong requirement cannot be met, conditional requirements (for example, **SHOULD** and **SHOULD NOT**) are used. If the requirements are going to be amended, they may become more restrictive, but they will not become looser. For example, the profile usually will not change a **MUST** to a **MAY**.

Profile Conformance

Conformance to the profile means adherence to specifications for the profile's scope and requirements.

Where possible, the profile places requirements on artifacts. Artifacts are the primary elements of any Web service, such as WSDL descriptions and SOAP messages. An artifact is conformant when all of the profile's requirements associated with that artifact type are met. Artifacts are concrete, making them easier to verify and, therefore, making conformance easier to understand and less error-prone.

Testability

When possible, profiles only make statements that are testable. Preferably, testing is achieved in a non-intrusive manner, such as examining artifacts "on the wire." However, due to the nature of cryptographic security, non-intrusive testing may not be possible.

Multiple Mechanisms

If one of the profile's specifications allows a variety of mechanisms to be used interchangeably, the profile uses the ones that are best understood and most widely implemented. This is because underspecified or little-known mechanisms and extensions can introduce complexity and, therefore, may reduce interoperability.

Compatibility

Although backward compatibility is not an explicit goal of profiles, they try to avoid introducing changes to the requirements of a specification unless doing so addresses specific interoperability issues. Also, where possible, the profile's requirements are compatible with in-progress revisions to the specifications it references.

Application Semantics

Application semantics lie outside of the profile. In this case, "application semantics" means definitions of what an application does. For example, an airline's reservations Web service will have the format of a message request defined, but whether that message charges a credit card or sends tickets through the mail is not specified.

Existing WS-I Profiles

The Basic Security Profile builds on two existing profiles defined by the WS-I, the Basic Profile 1.1 and the Simple SOAP Binding Profile (SSBP) 1.0. To understand the Basic Security Profile, it is important to also have a good understanding of these profiles.

The WS-I Basic Profile (BP) 1.1

The WS-I Basic Profile consists of a set of constraints and guidelines that, if followed, will help developers write interoperable Web services. The Basic Profile 1.1 scope includes the following specifications:

- [Simple Object Access Protocol \(SOAP\) 1.1](#)
- [Extensible Markup Language \(XML\) 1.0 \(Third Edition\)](#)
- [RFC2616: Hypertext Transfer Protocol — HTTP/1.1](#)
- [RFC2965: HTTP State Management Mechanism](#)
- [Web Services Description Language \(WSDL\) 1.1](#)
- [XML Schema Part 1: Structures](#)
- [XML Schema Part 2: Datatypes](#)
- [UDDI Version 2.04 API Specification, Dated 19 July 2002](#)
- [UDDI Version 2.03 Data Structure Reference, Dated 19 July 2002](#)
- [UDDI Version 2 XML Schema](#)
- [RFC2818: HTTP Over TLS](#)
- [RFC2246: The TLS Protocol Version 1.0](#)
- [The SSL Protocol Version 3.0](#)
- [RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile](#)

The WS-I Basic Profile 1.1 supersedes Basic Profile 1.0. In Basic Profile 1.1, the serialization of envelopes over HTTP is split off into a separate profile, the Simple SOAP Binding Profile (SSBP) 1.0.

The WS-I Simple SOAP Binding Profile (SSBP) 1.0

The SSBP 1.0 consists of a series of constraints and guidelines on how to serialize SOAP messages over HTTP. It includes the following specifications:

- [Simple Object Access Protocol \(SOAP\) 1.1](#)
- [Extensible Markup Language \(XML\) 1.0 \(Third Edition\)](#)
- [RFC2616: Hypertext Transfer Protocol — HTTP/1.1](#)
- [Web Services Description Language \(WSDL\) 1.1, Section 3](#)
- [Namespaces in XML 1.0](#)

This allows different transports to be used in conjunction with the Basic Profile 1.1.

Web Services Security Specifications

Figure 1.1 shows the relationships between various Web service specifications focusing on Web service security.

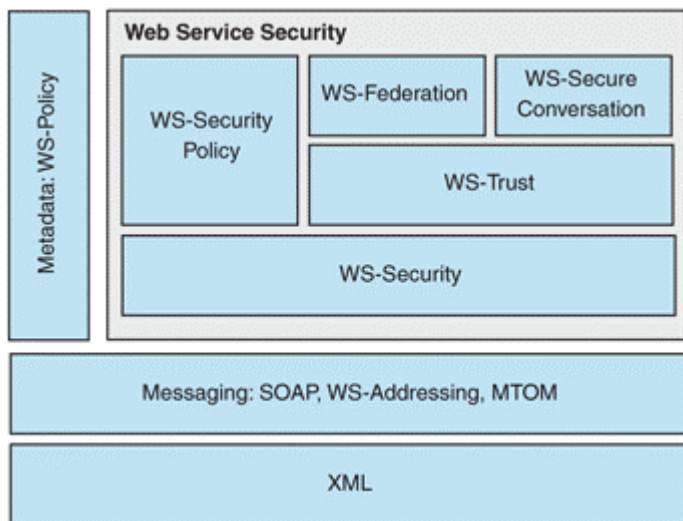


Figure 1.1

The relationship between Web services specifications

These specifications are designed to be used together to provide secure Web services. However, each of the specifications provides its own discrete functionality:

- **WS-Security**. This describes enhancements to SOAP messaging to provide *quality of protection* through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies. It also provides a general-purpose mechanism for associating security tokens with messages. Also, WS-Security describes how to encode X.509 certificates and Kerberos tickets as well as how to include opaque encrypted keys. WS-Security includes extensibility mechanisms that can be used to further describe the characteristics of the credentials that are included with a message.
- **WS-SecurityPolicy**. This describes the capabilities and constraints of the security (and other business) policies on intermediaries and endpoints (for example, required security tokens, supported encryption algorithms, or protected message parts).
- **WS-Trust**. This describes a framework for trust models that enables Web services to securely interoperate.
- **WS-SecureConversation**. This describes how to manage and authenticate message exchanges between parties, including security context exchange and establishing and deriving session keys.
- **WS-Federation**. This describes how to manage and broker the trust relationships in a heterogeneous federated environment, including support for federated identities.

The WS-I Basic Security Profile

The WS-I Basic Security Profile specifies how the OASIS WS-Security specifications should be interpreted to increase the likelihood of use of WS-Security in an interoperable way. It builds on the WS-Security specifications, WS-I Basic Profile 1.1 (BP 1.1), and the Simple SOAP Binding Profile (SSBP) 1.0.

While the Basic Security Profile is consistent with BP 1.1 and SSBP 1.0, it profiles additional functionality, showing how to add conformant security features to the basic profile when needed. Specifically, it builds on the following underlying specifications:

- [HTTP Over TLS](#)
- [Web Services Security: SOAP Message Security](#)
- [Web Services Security: UsernameToken Profile](#)
- [Web Services Security: X.509 Token Profile](#)
- [XML-Signature Syntax and Processing](#)
- [Web Services Security: SOAP Message Security Section 9](#)
- [XML Encryption Syntax and Processing](#)

Secure SOAP Messaging

Web service clients interact with services using SOAP messages. Securing these SOAP messages involves ensuring that the messages are transported in a secure way. While SSL/TLS provides transport-level security from point-to-point, it can be insufficient for Web services application topologies. These topologies often include a complex combination of geographically dispersed intermediaries (such as gateways, proxies, and message queues) that receive, process, and forward the messages. SOAP message security provides support for confidentiality and data integrity at the message layer. This allows the security to persist across intermediaries and results in security from end-to-end between the client and the Web service.

Summary

For Web services to provide application integration across different platforms, they must adhere to common standards. Core standards such as XML, SOAP, and WSDL are being widely adopted, but for true interoperability, there must be a clear understanding of Web service requirements and adherence to specifications. The WS-I was formed to take a high-level view of Web services and determine how they should work in a heterogeneous environment. By creating profiles such as the BSP 1.0 profile, the WS-I can stipulate the use of named and versioned Web services specifications, along with explaining how the specified components should be used together to develop interoperable Web services.

This chapter has explained the role of the WS-I and introduced the Basic Security Profile, which forms the basis of the rest of the chapters in this guide. Future chapters will discuss the Sample Application in detail.

Additional Information

For more information on deliverables from the WS-I Basic Security Profile Working Group, visit <http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicsecurity>.

2

Installing the Sample Application

This chapter demonstrates how to install the Sample Application and describes additional configuration steps you need to perform in your environment after installation. It also examines the installation in more detail and describes the configuration changes that occur during the installation process.

Installing the Sample Application

You can install the Sample Application in two different environments — on a single computer that meets all the software prerequisites, or in a two-computer environment, where the databases run on a separate computer.

It is also possible to run the Sample Application in a multi-tier environment, where all services could be deployed and run on different computers. However, the setup routine is designed to support only the single server and two server environments.

Note: The default installation of the Sample Application has been optimized for the development environment. Installation assumes defaults to debug mode within Visual Studio, uses a common application pool for all Web services and maps multiple warehouses and manufacturers to an individual physical directory to simplify navigation within Visual Studio. This configuration is not recommended for a production environment.

Installing and Running the Application on a Single Computer

To install the Sample Application, your environment must meet a number of prerequisites:

- Microsoft Windows 2000, Windows XP, or Windows Server 2003
- Microsoft Visual Studio .NET 2003, Enterprise Architect or Enterprise Developer edition
- Microsoft Web Services Enhancements (WSE) version 2.0 SP3
- Microsoft SQL Server 2000a SP3a or Microsoft SQL Server 2000 Desktop Engine (MSDE 2000) Release A

► To install the Sample Application

1. Download the application installation file
WSISampleApplicationPreviewRelease.msi from the WS-I BSP v1.0
Sample Application Preview Release MSDN site.
2. In the folder where you downloaded the .msi file, double-click
WSISampleApplicationPreviewRelease.msi.
3. On the Welcome page, click **Next**.
4. View the License Agreement, and if you agree, select **I Agree**. Click **Next**.
5. Select a folder in which the application should be installed, and choose whether
the application should be available only to yourself, or to everyone using the
computer, by selecting either **Just Me** or **Everyone**. Click **Next**.
6. On the confirmation page, click **Next**. The application will now install.
7. When the application finishes installing, click **Close** to close the setup wizard.

You are now ready to run the Sample Application, as shown in Figure 2.1. To run it, click **Start** on the taskbar, point to **All Programs**, point to **Microsoft Patterns & Practices**, and then point to **WSI Sample Application**. You should see icons for the WS-I BSP 1.0 Sample Application solution and the WS-I Supply Chain Management V1.0 Web site. If setup was successful, you can start the application by clicking the icon for the Web site. For a detailed walkthrough of the application, see Chapter 3, “Sample Application Walkthrough.”

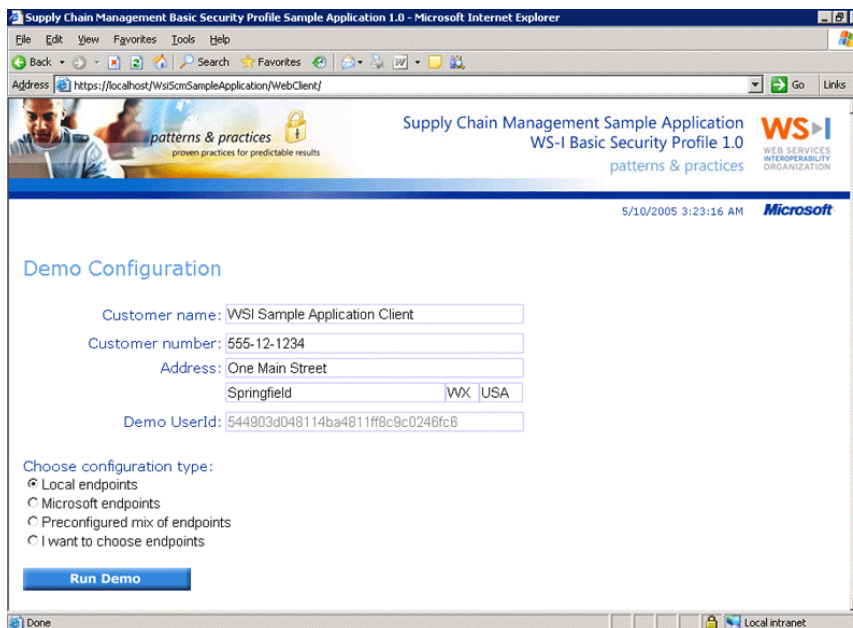


Figure 2.1
The Sample Application

Installing the Application in a Two-Computer Environment

To install the Sample Application into a two-computer system, where the Web server is on one computer, and the database is on the other, both computers must be members of the same Windows server domain or workgroup, and must meet a number of software prerequisites.

Computer A prerequisites:

- Microsoft Windows 2000, Windows XP, or Windows Server 2003
- Microsoft Visual Studio .NET 2003, Enterprise Architect or Enterprise Developer edition
- Microsoft Web Services Enhancements (WSE) version 2.0 SP3
- SQL Server 2000 Client Tools

Computer B prerequisites:

- Must be a member of the same Windows server domain or workgroup as Computer A
- Microsoft Windows 2000, Windows XP, or Windows Server 2003
- Microsoft SQL Server 2000 SP3a or MSDE 2000 Release A

Note: The user running the installation must have administrator privileges on both Computers A and B.

► To install the Sample Application

1. Download the application installation file `WSISampleApplicationPreviewRelease.msi` from the WS-I BSP v1.0 Sample Application Preview Release MSDN site to *Computer A*.
2. In the folder where you downloaded the .msi file, double-click **WSISampleApplicationPreviewRelease.msi**.
3. On the Welcome page, click **Next**.
4. View the License Agreement, and if you agree, select **I Agree**. Click **Next**.
5. Select a folder in which the application should be installed, and choose whether the application should be available only to yourself, or to everyone using the computer, by selecting either **Just Me** or **Everyone**. Click **Next**.
6. On the confirmation page, click **Next**. The application will now begin installing

7. During the installation, the dialog box in Figure 2.2 will appear:



Figure 2.2

Dialog box that appears during installation on Computer A in a two computer environment.

Click **OK** to continue.

8. When the application finishes installing, click **Close** to close the setup wizard.
9. Now install the database for the Sample Application onto remote Computer B:
On *Computer A*, open the file <install dir>\setup\scripts\SCMDatabaseSetup.vbs in Notepad.

Note: The default installation directory is C:\Program Files\Microsoft Patterns & Practices\WSISampleApplication.

10. In the first line of the script, replace the elements **sqlServerHost** and **sqlServerInstance** with the appropriate machine name of the target computer running SQL Server; in this case, *Computer B*.
11. Run the setup\scripts\SCMDatabaseSetup.vbs script.
12. Open the Data.config file in the <install folder>\WSISampleApplication\SCM Sample Application\WebApplication\WebClient\Configuration folder.
13. In the <connectionString name="NameOfService"> section, change the following XML element tag by replacing the period inside the double quotes (".") with the remote server name.

```
<parameter name="server" value="." isSensitive="false" />
```

Change it to the following.

```
<parameter name="server" value="ComputerB" isSensitive="false" />
```

14. Repeat step 13 with the Data.config files in the following paths:
 - <install folder>\WSISampleApplication\SCM Sample Application\Services\ConfiguratorApplication\ConfiguratorService\Configuration
 - <install folder>\WSISampleApplication\SCM Sample Application\Services\LoggingFacilityApplication\LoggingFacilityService\Configuration
 - <install folder>\WSISampleApplication\SCM Sample Application\Services\ManufacturerApplication\ManufacturerService\Configuration
 - <install folder>\WSISampleApplication\SCM Sample Application\Services\RetailerApplication\RetailerService\Configuration
 - <install folder>\WSISampleApplication\SCM Sample Application\Services\WarehouseApplication\WarehouseService\Configuration
15. At this point, there are many options available for configuring how the Web services in the application will interact with the database (for example, using Windows Authentication, SQL Authentication or Integrated Security). Decisions will differ based on a number of variables, such as operating system and the version of SQL Server being used. For more information about setting up secure user accounts in SQL Server 2000, see Chapter 18, "Securing Your Database Server," of *Improving Web Application Security: Threats and Countermeasures* on MSDN at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh18.asp>.

You are now ready to run the Sample Application, as shown in Figure 2.3. To run it, click **Start** on the taskbar, point to **All Programs**, point to **Microsoft Patterns & Practices**, and then point to **WSI Sample Application**. You should see icons for the WS-I BSP 1.0 Sample Application solution and the WS-I Supply Chain Management V1.0 Web site. If setup was successful, you can start the application by clicking the icon for the Web site. For a detailed walkthrough of the application, see Chapter 3, “Sample Application Walkthrough.”

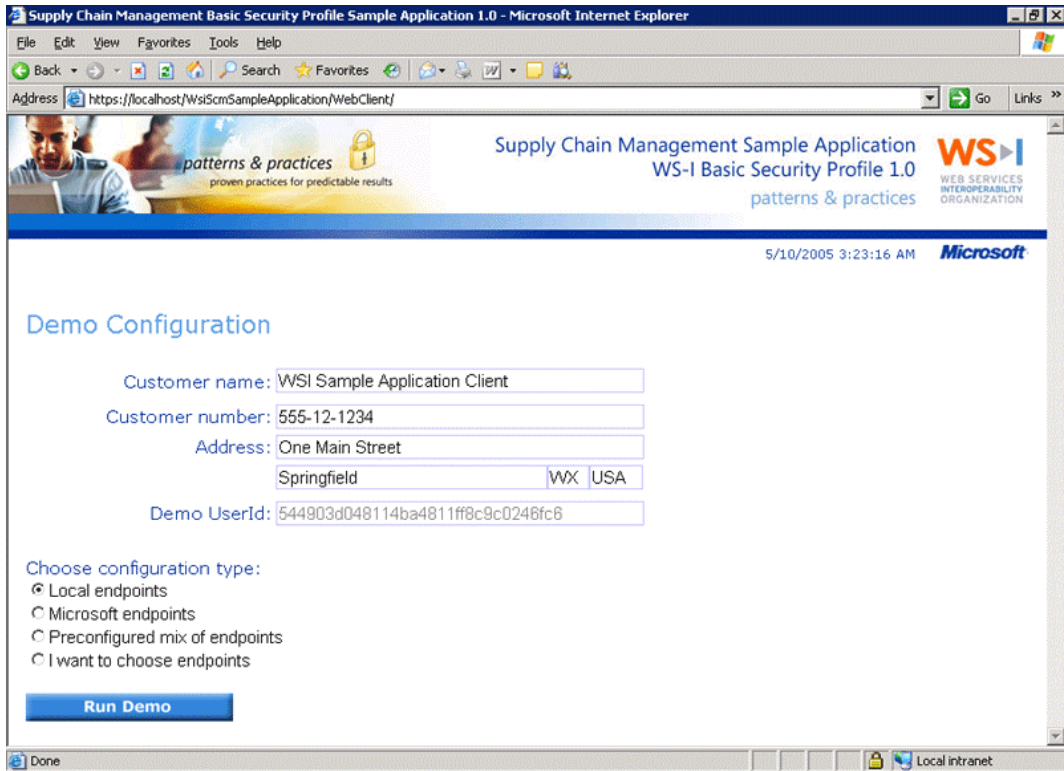


Figure 2.3
The Sample Application

Post-Installation Configuration

After you install the Sample Application, there are some steps that you need to take to ensure the application has been installed properly. This section details those steps.

Modifying Configuration Files

In some cases, you will need to modify the behavior of the application by editing the Web.config files. Because the application is integrated with Enterprise Library, you can use the Enterprise Library Configuration Console to edit these files, as shown in Figure 2.4.

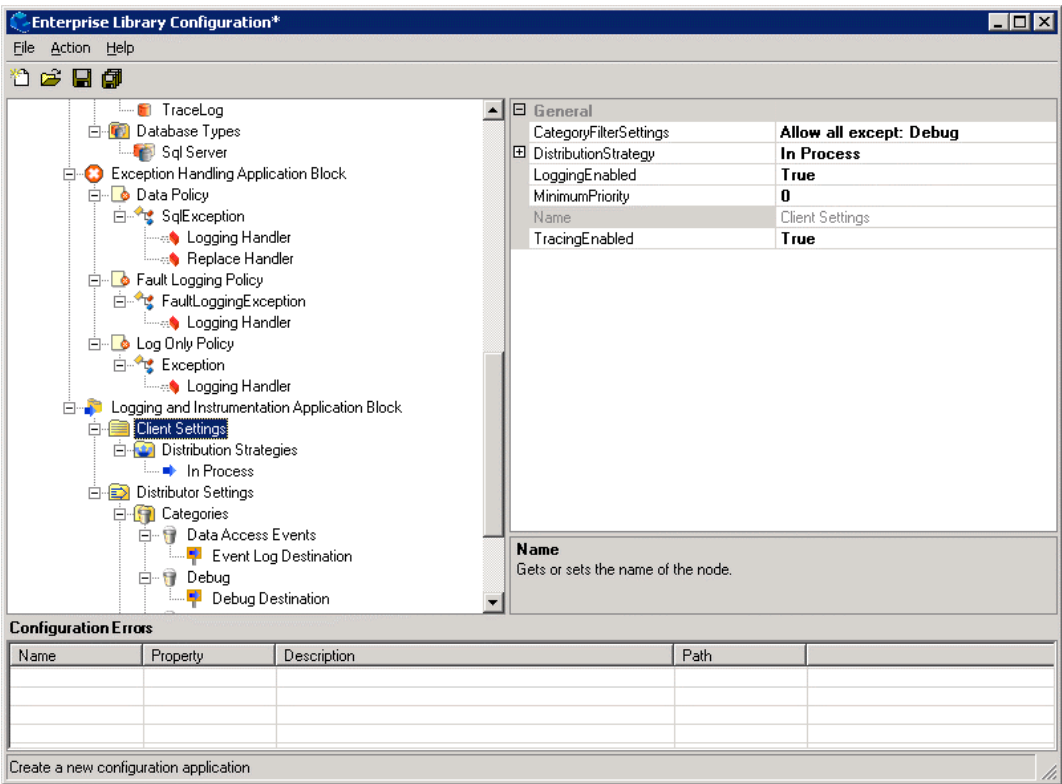


Figure 2.4
The Enterprise Library Configuration Console

For more information, see Appendix A, "Enterprise Library Integration."

Application Pools

Microsoft Windows Server 2003 allows Web services to be deployed into different application pools, optimizing availability and performance. Because the installation is designed for a developer environment, all the virtual directories belong to the default application pool, but in your environment, you should consider creating separate application pools for the following Web services:

- Configurator Web service
- Logging Facility Web service
- Manufacturer Web services
- Retailer Web service
- Warehouse Web services
- Web client Web service

For more information about application pools, see “Configuring Application Isolation on Windows Server 2003 and Internet Information Services (IIS) 6.0” on Microsoft TechNet at <http://www.microsoft.com/technet/prodtechnol/windowsserver2003/technologies/webapp/iis/appisooa.mspx>.

Enabling SSL on the Presentation Tier

The Sample Application does not have any requirements for securing the connection to the Presentation tier. However, it is possible to secure the connection using SSL, as shown in the following procedure.

► To enable SSL on the Presentation tier

1. On the taskbar, click **Start**, point to **Administrative Tools**, point to **Internet Information Services (IIS) Manager**, and then click **Internet Information Services**.
2. In Internet Information Services, expand **Local Computer**, and then expand **Websites**.
3. Right-click **Default Web Site**, and then click **Properties**.
4. Click the **Directory Security** tab, and then click **Server Certificate**.
5. In the Web Server Certificate Wizard, click **Next**.
6. Select **Assign an Existing Certificate**, and then click **Next**.
7. Select **webclient.wsi.org**, and then click **Next**.
8. Verify that the SSL port is 443, and then click **Next**.
9. Review the Certificate Summary, and then click **Next**.
10. Click **Finish**.
11. Click **OK**.

12. Expand **WsiScmSampleApplication**.
13. Right-click **WebClient**, and then click **Properties**.
14. Click the **Directory Security** tab.
15. Under **Secure Communications**, click **Edit**.
16. Select the **Require Secure Channel (SSL) and Require 128-bit encryption** check box, and then click **OK**.
17. Click **OK**.

Note: After SSL is enabled in this way, when you start the Sample Application, you will receive a security warning that the name on the security certificate is invalid or does not match the name of the site. This is because the certificate is not actually issued by or for the WS-I; it is for demonstration purposes only.

Verifying the Installation

Setup installs and configures the following components and services on your computer:

- The IIS virtual directories needed for the application
- The SQL Server 2000 database and tables needed for the application
- The X.509 certificates required by the application
- Additional services required for operation of the application

While setup should be automatic, or in the two computer-case, require minimal manual intervention, it is useful to be aware of the steps that setup performs, so that you can troubleshoot any problems with the installation.

IIS Virtual Directories

The Windows Installer package creates a parent virtual directory named **WsiScmSampleApplication**, with the following subdirectories:

- | | |
|-------------------|--------------|
| • Configurator | • Retailer |
| • LoggingFacility | • WarehouseA |
| • ManufacturerA | • WarehouseB |
| • ManufacturerB | • WarehouseC |
| • ManufacturerC | • WebClient |

SQL Server 2000 SP3a Databases and Tables

The Sample Application uses the following SQL Server databases:

- WSI-ManufacturerA
- WSI-ManufacturerB
- WSI-ManufacturerC
- WSI-Retailer
- WSI-WarehouseA
- WSI-WarehouseB
- WSI-WarehouseC
- WSI-Log
- TraceLog

Each of these databases has a corresponding Web service, except for TraceLog, which is used by all of the Web services for tracing purposes.

Note: The Windows Installer package installs these databases on the local host.

X.509 Certificates

The Sample Application uses the following X.509 certificates:

- Trusted root certificate authority certificates:
 - WS-I Sample App CA.crt
 - WS-I Sample App CA.p7b
- Personal certificates:
 - WSILoggingFacilityEncryption.pfx
 - WSILoggingFacilitySigning.pfx
 - WSILoggingFacilitySSL.pfx
 - WSIManufacturerAEncryption.pfx
 - WSIManufacturerASigning.pfx
 - WSIManufacturerASSL.pfx
 - WSIManufacturerBEncryption.pfx
 - WSIManufacturerBSigning.pfx
 - WSIManufacturerBSSL.pfx
 - WSIManufacturerCEncryption.pfx
 - WSIManufacturerCSigning.pfx
 - WSIManufacturerCSSL.pfx
 - WSIRetailerEncryption.pfx
 - WSIRetailerSigning.pfx
 - WSIRetailerSSL.pfx
 - WSIWarehouseAEncryption.pfx
 - WSIWarehouseASigning.pfx
 - WSIWarehouseASSL.pfx
 - WSIWarehouseBEncryption.pfx
 - WSIWarehouseBSigning.pfx
 - WSIWarehouseBSSL.pfx
 - WSIWarehouseCEncryption.pfx
 - WSIWarehouseCSigning.pfx
 - WSIWarehouseCSSL.pfx
 - WSIWebClientEncryption.pfx
 - WSIWebClientSigning.pfx
 - WSIWebClientSSL.pfx

The Sample Application also requires that the Network Authority account (or ASP.NET account on Windows XP) is given **Read** and **Read and Execute** permissions over all the certificates, except the SSL certificates, which are not used in this version of the Sample Application.

Note: For more information about the various certificate stores available and which certificates should be stored where, see “How to: Make X.509 Certificates Accessible to WSE” on MSDN at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wse/html/ea7d6db0-8d74-4b6b-ba3a-207db29f5757.asp?frame=true>.

Uninstalling the Application

The process for uninstalling the application varies, depending on whether the application was installed on a single computer or on two computers.

Uninstalling the Application on a Single Computer

To uninstall the application on a single computer, simply locate the program in the Add/Remove Programs section of *Computer A*, and then click **Remove**.

Uninstalling the Application in a Two Computer Environment

In a two-computer environment as described earlier in this chapter, you can simply locate the program in Add/Remove Programs on *Computer A*, and then click **Remove**. However, you will also need to manually remove the databases on *Computer B* and remove accounts that were used to access the databases.

Summary

To fully understand the concepts in this guide, it is very useful to have the Sample Application installed. This chapter has explained how to perform the installation and has described some useful post-installation steps.

The chapter has also examined the environment created by the installation, so you can verify that the installation proceeded properly and understand the changes made by the installation program.

3

Sample Application Walkthrough

This chapter demonstrates the usage of the Sample Application, detailing the control flow of the application and guiding you through a detailed walkthrough of the Sample Application.

Note: NUnits are available to verify the functionality of the application. You can download them from the GotDotNet Web site at <http://workspaces.gotdotnet.com/wsibsp>.

Sample Application Control Flow

The Sample Application uses three communication patterns:

- **Typical synchronous exchanges.** The majority of communications are in the form of SOAP requests eliciting SOAP responses.
- **One-way messages.** The messages sent to the logging facility can be sent in the form of one-way messages. This has a lower overhead than SOAP request/response.
- **Asynchronous callbacks.** The interchange between a warehouse and manufacturer requires asynchronous callbacks, because a manufacturer cannot always respond immediately to the purchase order from the warehouse with all the information the warehouse needs to know. The manufacturer may already have the goods, or it may have to schedule a production run.

The flow chart in Figure 3.1 illustrates the core sequence of Web application and Web service calls that take place in an application that is based on the Supply Chain Management (SCM) architecture.

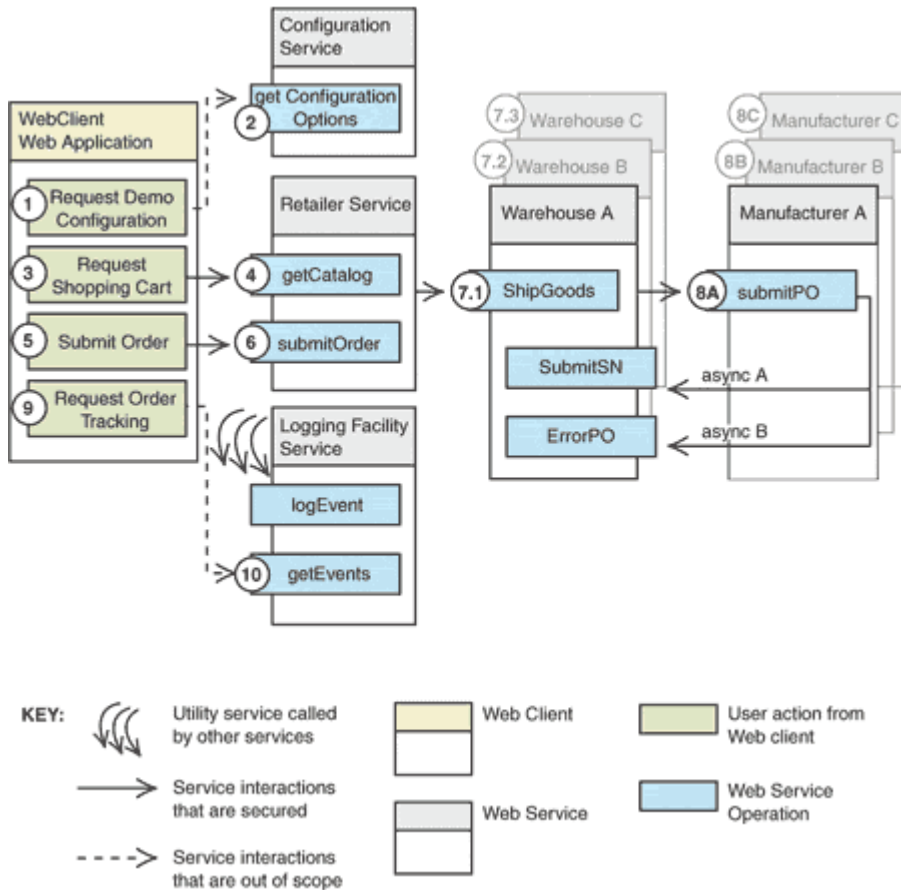


Figure 3.1

Sample Application control flow

The numbered steps in Figure 3.1 represent the steps in the synchronous communication that occurs:

1. The user requests the Web Client Demo Configuration Web page from the Web client Web application.
2. The Web client calls the Configurator Web service's **getConfigurationOptions** method (occurs only when the default **I want to choose endpoints** configuration option is selected on the Web Client Demo Configuration page).
3. The user requests the Shopping Cart Web page from the Web client.

4. The Web client calls the Retailer Web service's **getCatalog** method.
5. The user submits the order by clicking the **Submit Order** button on the Shopping Cart page. After steps 6–7 (and optionally step 8) take place, an Order Status page appears.
6. The Web client calls the Retailer Web service's **submitOrder** method.
7. Retailer code calls the **ShipGoods** method from the WarehouseA instance of the Warehouse Web service. For those part numbers that WarehouseA does not have sufficient stock to fulfill the order, the **Retailer** code calls **ShipGoods** from the WarehouseB instance of the Warehouse Web service. For those part numbers that WarehouseB does not have sufficient stock to fulfill the order, the **Retailer** code calls **ShipGoods** from the WarehouseC instance of the Warehouse Web service.
8. If a warehouse fulfills an order and, in doing so, falls below its repurchase threshold, it calls **submitPO** from the appropriate manufacturer's instance of the Manufacturer Web service. It passes along the necessary information for the manufacturer to later call the **SubmitSN** or **ErrorPO** method of the Warehouse Callback Web service.
9. The user requests the Track Order Web page from the Web client.
10. The Web client calls the **getEvents** method of the logging facility Web service. In the course of the previous steps, calls are made by assorted Web services (or the Web client) to the **logEvent** method of the logging facility Web service. These calls have been left out of the flow chart's numbered sequence to prevent clutter. The results of the **logEvent** invocations are returned in the response to **getEvents**.

The additional asynchronous communication is represented in the diagram by the labels **asynch A** and **asynch B**. After a purchase order is received and acknowledged, one of the following occurs:

- **asynch A**: The manufacturer sends a shipping notice by calling the **SubmitSN** operation of the **Warehouse Callback** service for the same warehouse that originally called the manufacturer's **submitPO** method.
- **asynch B**: Alternatively, if there is an error, the manufacturer calls the callback service's **ErrorPO** method.

Note: Only communications between the Web client, Retailer, Warehouses, and Manufacturers are secured. The communications between the configuration service and logging services are not secured.

Walkthrough

To run the Sample Application, you must first complete the installation steps discussed in Chapter 2 of this guide. Once the installation is complete, you can click **Start**, select **All Programs**, select **Microsoft Patterns & Practices**, then select **WSI Sample Application** and click **WS-I Supply Chain Management v1.0**. The Sample Application consists of the following sequence of Web pages:

1. **Demo Configuration**. This is the start page where you configure the application — that is, decide how Web services are chosen.
2. **Shopping Cart**. This is where you can fill out a sample order.
3. **Order Status**. This is where you can see what you have ordered.
4. **Track Order**. This is where you can view a series of log entries highlighting what happened behind the scenes during the order.

Note: On each page following the Demo Configuration page, you can click the **Start Over** button to go back to the Demo Configuration page.

Demo Configuration Page

To use configure the application, you need to browse to the Sample Application's start page. By default, this is located at localhost/WsiScmV1/WebClient. A Demo Configuration Web form with pre-entered values appears, as shown in Figure 3.2.

Supply Chain Management Basic Security Profile Sample Application 1.0 - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Search Favorites Go Links

Address <https://localhost/WsiScmSampleApplication/WebClient/>

patterns & practices proven practices for predictable results

Supply Chain Management Sample Application
WS-I Basic Security Profile 1.0
patterns & practices

WS-I WEB SERVICES INTEROPERABILITY ORGANIZATION

5/10/2005 3:23:16 AM Microsoft

Demo Configuration

Customer name:

Customer number:

Address:

Demo UserId:

Choose configuration type:

☒ Local endpoints

☐ Microsoft endpoints

☐ Preconfigured mix of endpoints

☐ I want to choose endpoints

Done Local intranet

Figure 3.2

Demo Configuration Web page

You can keep the pre-entered values or you can change the values. The choices for endpoints are as follows:

- **Microsoft endpoints.** Use Microsoft's implementations of the Web services; they are available on the Internet. The Configurator Web service is bypassed.
- **Preconfigured mix of endpoints.** These are implementations from a mix of vendors, including Microsoft, that are available on the Internet. The endpoints are taken from a file, WsiScmV1\WebClient\PreConfigEndPoints.xml. The Configurator Web service is bypassed.
- **I want to choose endpoints.** Use the Configurator Web service to obtain the other Web services. This is the default choice.

Note: The preview version of the sample application does not support running against endpoints from vendors other than Microsoft. As the WS-I BSP becomes finalized, other vendors will have implementations of this application available and pointers will be provided to their implementations so that you can test interoperability for yourself.

If **I want to choose endpoints** is selected, eight drop-down menus appear on the page. These drop-down menus enable you to select implementations for each of the other Web services: Retailer, Logging Facility, three warehouses, and three manufacturers. If the Configurator Web service is configured to find Web services through the local EndPoints.xml file instead of by querying a Universal Discovery Description and Integration (UDDI) registry, each of these eight menus will have one valid choice — the Web service established during setup. For example, the Retailer drop-down menu will have two items, **Please select a Retailer** and **Retailer on localhost**.

After you select the configuration that you want, click **Run Demo**.

Later, you can come back to this page to try other configuration types or to try obtaining endpoints from UDDI.

Note: To use the other configuration type options on the Demo Configuration page, you will need to modify the Web.config file for the Configurator Web service so that it obtains endpoints from both UDDI and EndPoints.xml (set both **UseUDDI** and **UseLocalXML** to **true**). With **I want to choose endpoints** selected, you need to make sure that the eight endpoints you choose are either all UDDI or all local. You cannot have a mix of the two types.

Shopping Cart Page

The next page, Shopping Cart, contains a list of products, each with a box where you can enter a quantity to order, as shown in Figure 3.3.

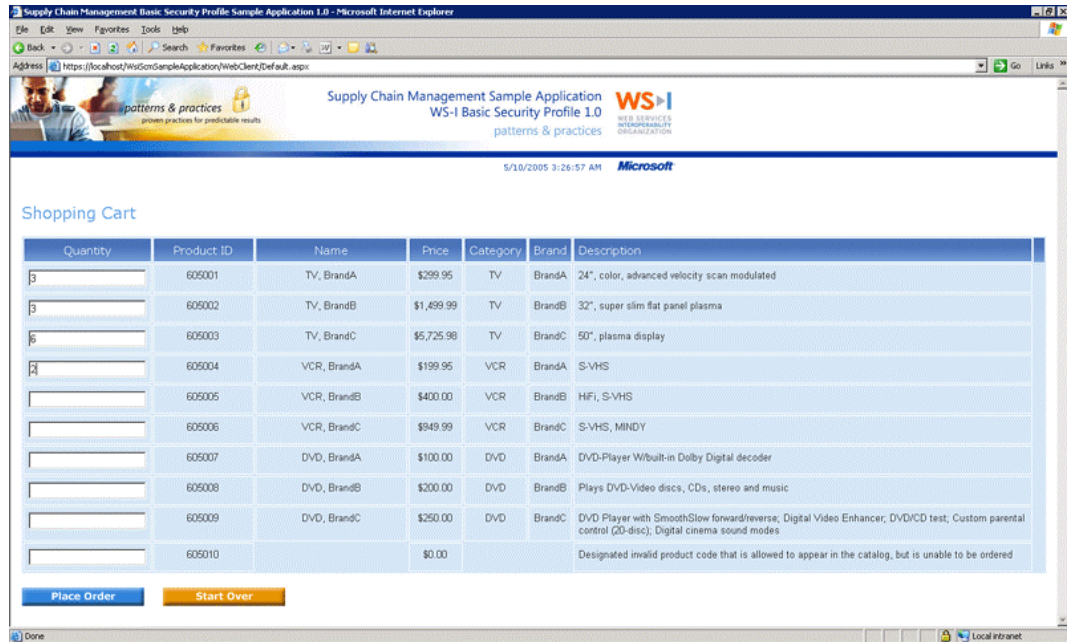


Figure 3.3

The Shopping Cart page of the Sample Application

When the Shopping Cart page displays, the Web client calls the Retailer Web service's **getCatalog** method. Figure 3.4 shows a SOAP message passed between the Web client and the Retailer.

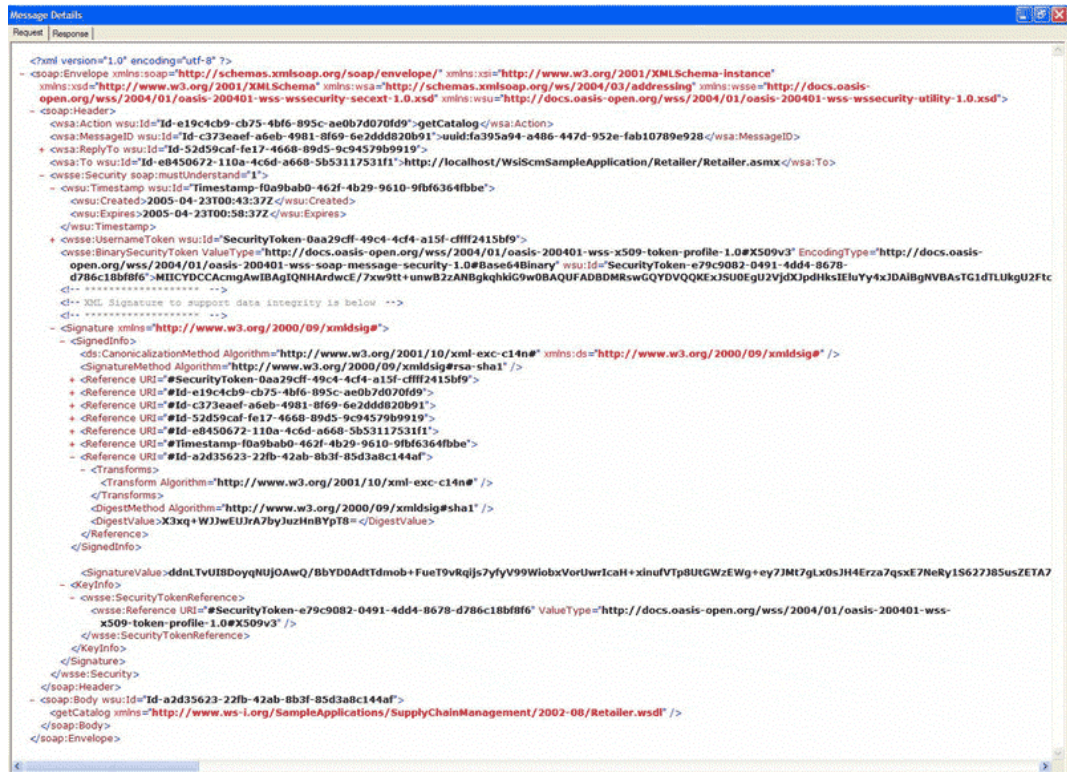


Figure 3.4
SOAP message passed between Web client and Retailer Web service

Figure 3.4 illustrates a SOAP message that includes a signature element within the security header. The signature element includes the following elements:

- **SignedInfo**. This includes references to each element within the message that is signed.
- **SignatureValue**. This contains the XML digital signature for the signed elements.
- **KeyInfo**. This references the binary security token that was used to verify the digital signature.

Note: For more information about how XML digital signatures operate, see “Understanding XML Digital Signature” on MSDN at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/underxmldigsig.asp>.

Figure 3.5 illustrates the response message that is returned from the Retailer to the Web client. The message contains catalog information that is both signed and encrypted.

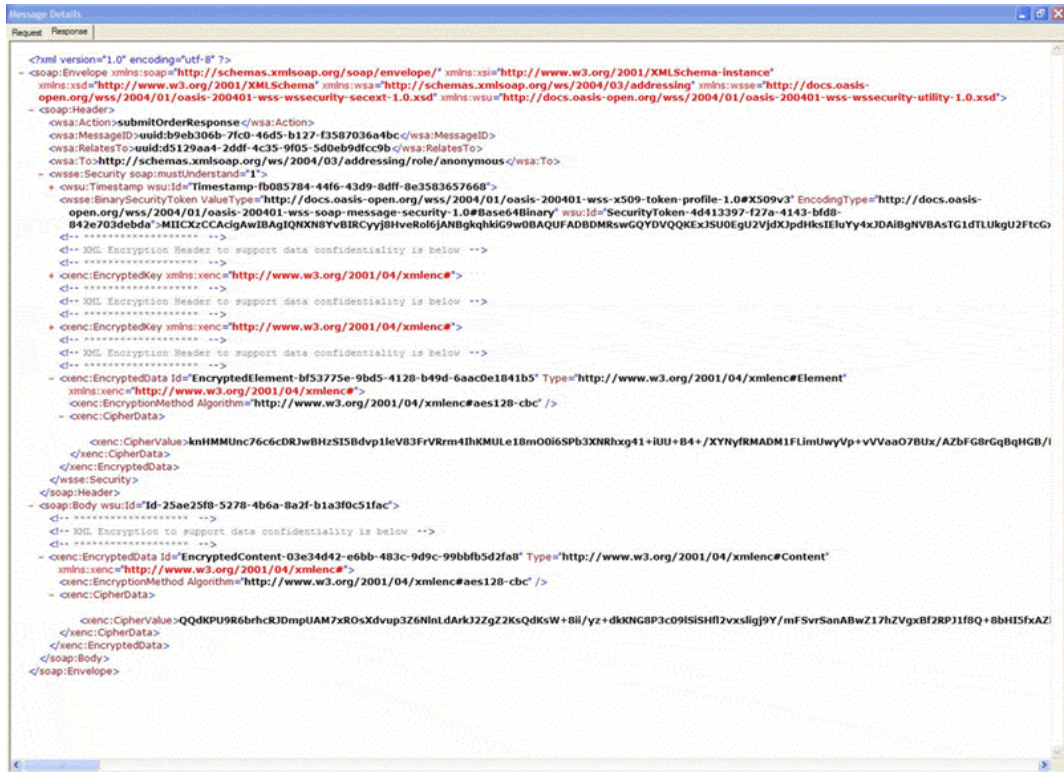


Figure 3.5

Response from the Retailer Web service to the Web client

Note: The SOAP messages captured in this chapter were obtained using a monitor that also automatically adds comments to show where key features such as XML signatures and XML encryption are added to the messages.

To download the monitor, see “Microsoft WS-I Basic Security Profile Sample Application: Workspace Home” on GotDotNet at <http://workspaces.gotdotnet.com/wsibsp>.

The Order Status Page

Figure 3.6 illustrates the Order Status page. This page shows each order, along with its price, and whether the item order can be fulfilled from the warehouse.

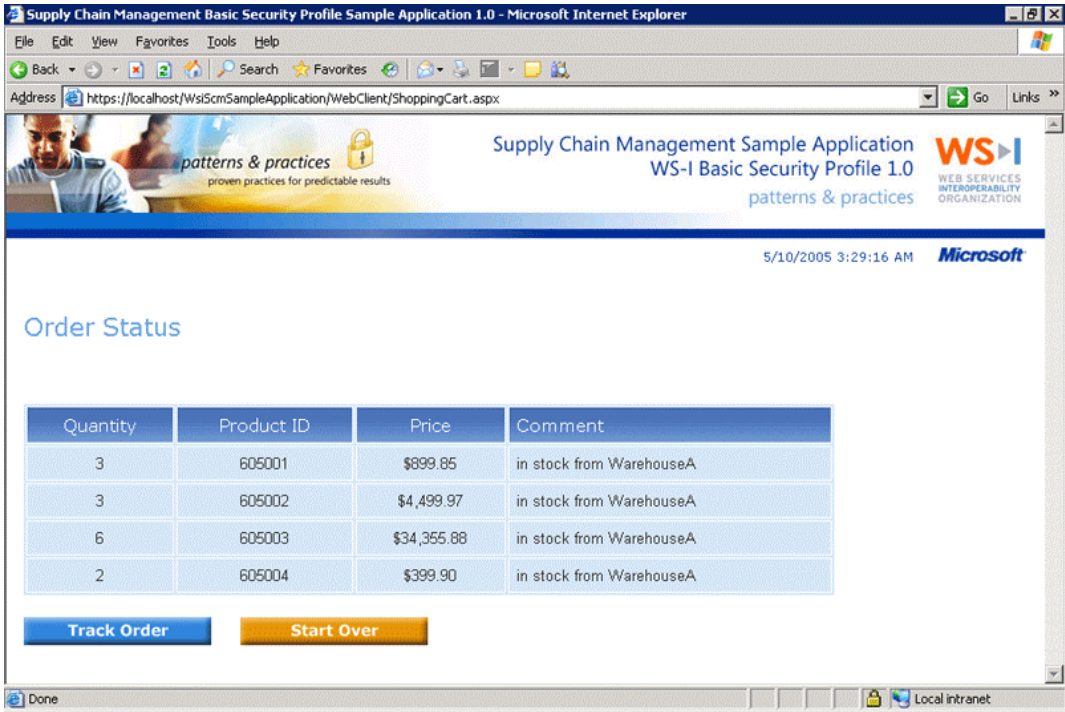


Figure 3.6
The Order Status page of the Sample Application

Finally, you come to the Track Order page.

A list of logged events appears. Each event's row has a timestamp, code, Web service and operation (such as **Retailer.submitOrder**), and description. Figure 3.7 illustrates an example of the Track Order page.

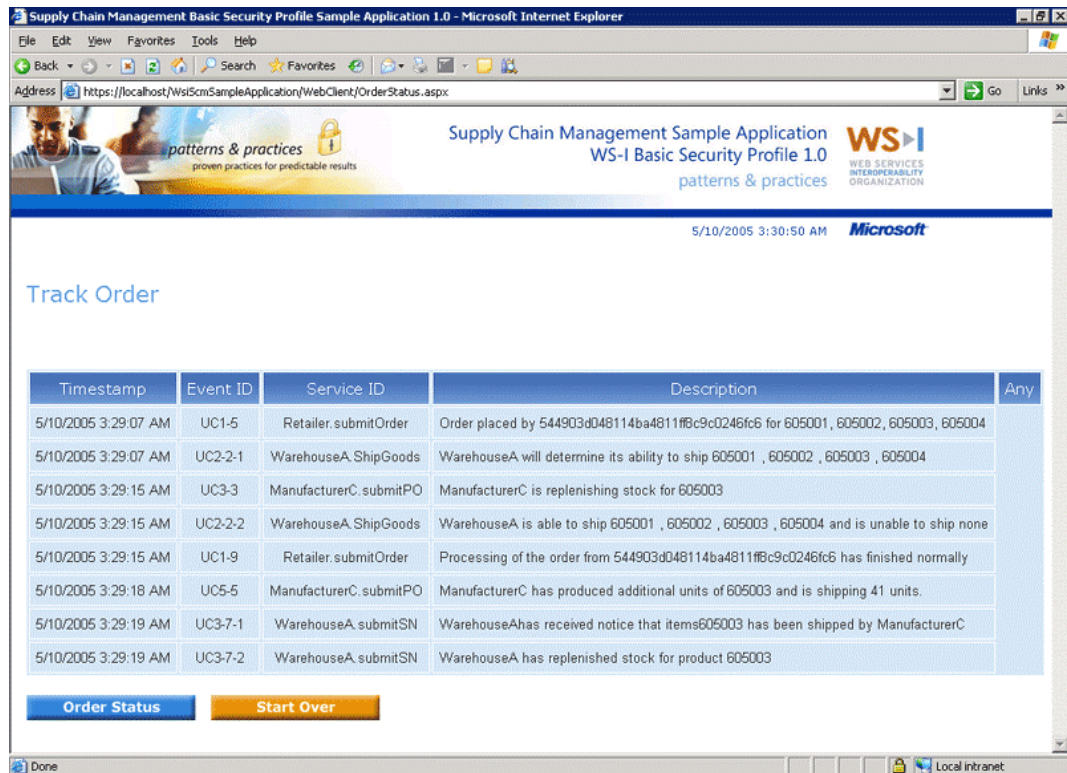


Figure 3.7

Track Order page displaying a log list

Summary

By understanding the control flow of the Sample Application, it is easier to see how the application implements the specifications of the BSP 1.0. This chapter has explained the control flow of the application in detail. It also provided a walkthrough of the application and demonstrated some of the SOAP messages that pass between services during the execution of the application.

4

Sample Application Architecture

This chapter describes the architecture of the Sample Application and demonstrates how the application is implemented using Microsoft technologies.

Supply Chain Management Architecture

Distributed applications often consist of a complex, interdependent set of applications and databases that are not easily described in a single diagram or from a single viewpoint. Examining the architecture of the Supply Chain Management (SCM) application from a number of different perspectives, or *views*, will help you better understand the architecture. This section presents four different views of the architecture:

- Requirements View
- Conceptual View
- Implementation View
- Deployment View

Requirements View

The functional requirements for the original WS-I Sample Application can be found in the “Supply Chain Management Use Case Model” document on the WS-I Web site at <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2003-12/SCMUseCases1.0.doc>. It describes a simplified sample application spanning several fictitious organizations — namely a Retailer, a series of warehouses, and a series of manufacturers.

The use case model document provides baseline functional requirements for the Basic Security Profile (BSP) version of the Sample Application. A new design document is currently in development by the WS-I BSP Sample Application Working Group. The new document describes the security requirements with which this application is implemented. This section contains a brief summary of those requirements as they relate to securing Web services.

The Sample Application uses an e-commerce scenario where a retailer is selling consumer electronics. The retailer has to manage stock in three warehouses, first checking Warehouse A. If Warehouse A cannot fill an order, the retailer then checks Warehouse B, and if Warehouse B cannot fill the order, it checks Warehouse C. When a warehouse's inventory of a particular product falls below a certain threshold, the warehouse orders more units from the appropriate manufacturer. There are three manufacturers.

Communications Between Web Services

Figure 4.1 illustrates how the Web services in the Sample Application communicate with each other and with the client Web application.

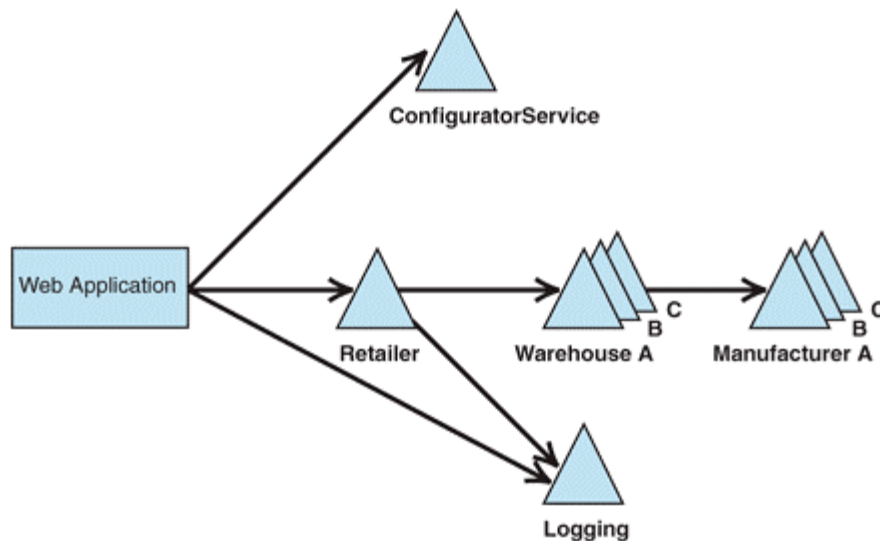


Figure 4.1

Relationship between Web services in the Sample Application

The client Web application directly communicates with three Web services:

- **Configurator.** It communicates with this Web service to determine the URI of the Logging and Retailer Web services.
- **Logging.** It communicates with this Web service to request the transaction log after the operation is completed.
- **Retailer.** It communicates with this Web service to send orders to the retailer.

Note: The Logging Web service is used by all the applications as a common logging repository.

The Web services are created with standard .asmx files, but Microsoft Web Services Enhancements (WSE) 2.0 is used because of its WS-Policy implementation.

Table 4.1 summarizes the original requirements for the Basic Profile 1.0 version of the Sample Application.

Table 4.1: Requirements for the Basic Profile 1.0 Version of the Sample Application

Service	Style/Encoding	Header	SOAP Action
Configurator	Document/literal	No	Empty string
Logging Facility	Document/literal	No	WSDL target namespace and operation
Retailer	RPC/literal*	Yes	Empty string
Warehouse	RPC/literal*	Yes	WSDL target namespace
Warehouse Callback	Document/literal	Yes	None
Manufacturer	Document/literal	Yes	None

As you review this table, you should consider a number of important caveats:

- A combination of document/literal and RPC/literal was specified in the original design document to test interoperability. However, Microsoft recommends using document/literal instead of RPC/literal. For more information, see “RPC/Literal and Freedom of Choice” on MSDN at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/rpc_literal.asp.
- Microsoft Visual Studio 2003 does not support importing RPC/literal WSDL, so document/literal versions of the RPC/literal WSDL were used instead. This WSDL is still valid for interoperability because RPC/literal is a subset of document/literal.
- Various SOAP actions were also used to test interoperability within the original Basic Profile 1.0 version of the Sample Application. However, this led to interoperability issues when mapping security policy to specific operations. The BSP version of the Sample Application always includes a SOAP action with a WSDL target namespace and operation. For more information, see “Mapping Security Policy to a Web Service Operation” in Chapter 7.

Each of the Web services, except the Configurator and Logging Facility in the Web client, also use custom SOAP headers. The Manufacturer service and the Warehouse Callback service publish start and callback headers, respectively, to establish and negotiate an asynchronous session. In addition, those two services and the Retailer and Warehouse services publish a configuration header.

A comprehensive description of the security requirements, analysis, and implementation that form the basis for all implementations of the Sample Application is currently being developed by the WS-I Sample Application Working Group. Table 4.2 summarizes the security features implemented for each operation on each service.

Table 4.2: Summary of Security Requirements for Sample Application

Operation	Authentication	Integrity	Confidentiality
Web Client → Retailer getCatalogRequest	Primary: X509 Secondary: UsernameToken (see Note)	Web Client X509: Body, UsernameToken, Timestamp	
Retailer → Web Client getCatalogResponse	X509	Retailer X509: Body, Timestamp	Web Client X509: Body, Signature
Web Client → Retailer submitOrderRequest	Primary: X509 Secondary: UsernameToken (see Note)	Web Client X509: Body, UsernameToken, Timestamp, Config Header	Retailer X509: Body, Signature, UsernameToken
Retailer → Web Client submitOrderResponse	X509	Retailer X509: Body, Timestamp	Web Client X509: Body, Signature
Retailer → Warehouse ShipGoodsRequest	X509	Retailer X509: Body, Timestamp, Config header	
Warehouse → Retailer ShipGoodsResponse	X509	Warehouse X509: Body, Timestamp	
Warehouse → Manufacturer POSubmit	X509	Warehouse X509: Body, Config hdr, Start hdr, Timestamp	Manufacturer X509: Body, Start hdr, Signature
Manufacturer → Warehouse ackPO	X509	Manufacturer X509: Body, Timestamp	

(continued)

(continued)

Operation	Authentication	Integrity	Confidentiality
Manufacturern → Warehousen SNSubmit	X509	Manufacturern X509: Body, Config hdr, Callback hdr, Timestamp	Warehousen X509: body, signature
Warehousen → Manfucaturern ackSN	X509	Warehousen X509: Body, Timestamp	
Manufacturern → Warehousen ProcessPOFault	X509	Manufacturern X509: Body, Config hdr, Callback hdr, Timestamp	
Warehousen → Manfucaturern ackPO	X509	Warehousen X509: Body, Timestamp	
Configurator	Out of scope	Out of scope	Out of scope
Logging	Out of scope	Out of scope	Out of scope

Note: The UsernameToken that is attached to the two messages from the Web client is used as a way to propagate identity across tiers. It does not contain a password. For more information, see “Dominant Patterns for Identity Propagation and Service Authentication” later in this chapter.

Conceptual View

The Sample Application is designed around a relatively small number of architectural patterns. This section examines the most significant patterns that are used in the application.

Dominant Patterns on the Presentation Tier

The Web client is not a required deliverable to support the BSP. However, several vendors are implementing the Web client to demonstrate the Web services. The Web client only has basic workflows and as such, it does not warrant a more complex architecture. For this reason, the Sample Application uses a straightforward Model-View-Controller pattern implementation, separating the Presentation tier into the following three distinct roles, as shown in Figure 4.2:

- **View.** Contains presentation logic to support each Web page.
- **Controller.** Accepts user input and interacts with both the View and the Model.
- **Model.** Responsible for managing the behavior and data necessary to be displayed from in the Presentation tier. Independent of both the View and the Controller.

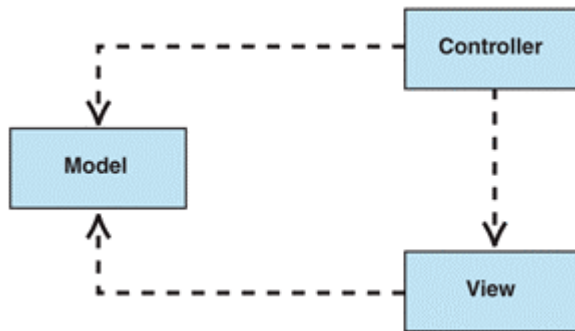


Figure 4.2

The Model-View-Controller pattern

Note: For more information about the Model-View-Controller pattern, see “Model-View-Controller” in the .NET Development Center on MSDN at <http://msdn.microsoft.com/architecture/patterns/default.aspx?pull=/library/en-us/dnpatterns/html/DesMVC.asp>.

Dominant Patterns Within Each Service

The business logic within the Sample Application is designed to be relatively simple in nature, so many of the Web services that form the application do not warrant a complex internal architecture. To reflect this, the Sample Application uses a layered architecture, based around the following three patterns, as shown in Figure 4.3:

- **Service Interface.** Responsible for the implementation of the service's contract and is dependent on a particular transport — in this case, that defined by WSE 2.0. This pattern allows for the separation of a services' interface from the business logic implemented within the service, allowing the business logic and service's interface to evolve independently. It also removes the dependence on a particular transport or product from the business logic.
- **Transaction Script.** Domain logic, organized into procedures. Because this application is meant to demonstrate interoperability instead of the complex kind of business logic that would be included inside a real SCM application, it was not necessary to create a formal domain model.
- **Data Access Layer.** Data access logic, organized into classes.

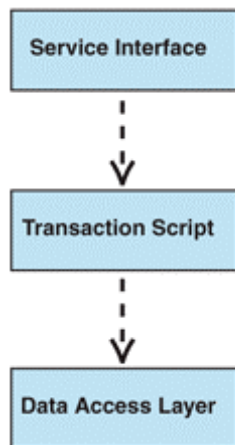


Figure 4.3

Patterns used by the Sample Application

This layered approach to the implementation of each service supports separation of concerns. This reduces dependencies between the product used to support the service's implementation, the business layer, and the location and implementation of the service's data access logic.

Figure 4.4 illustrates the Service Interface (RetailerService), Transaction Script (RetailerBusinessActions), and Data Access Layer (RetailerDataAccess) patterns implemented separately in Visual Studio.

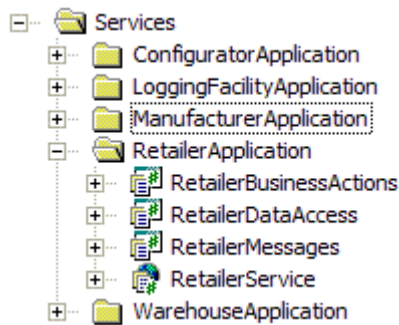


Figure 4.4

Implementing patterns in Visual Studio

For more information about the Service Interface pattern, see “Service Interface” in the .NET Development Center on MSDN at <http://msdn.microsoft.com/architecture/patterns/default.aspx?pull=/library/en-us/dnpatterns/html/DesServiceInterface.asp>.

For more information about the Layered Application pattern, see “Layered Application” in the .NET Developer Center on MSDN at <http://msdn.microsoft.com/architecture/patterns/default.aspx?pull=/library/en-us/dnpatterns/html/ArcLayeredApplication.asp>.

For more information about the Transaction Script pattern, see:

- Fowler, Martin. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional. 2002.

Dominant Patterns for Identity Propagation and Service Authentication

The Sample Application uses a trusted subsystem security model. It uses fixed identities to represent each entity (the retailer, each warehouse, and each manufacturer). Downstream services are responsible for calling the Web client that authenticates the original user.

Authentication between tiers of the application uses X.509 certificates, with a UsernameToken flowing the identity of the originating user from the Web client to the Retailer services. This UsernameToken does not include the originating user’s password, because the Retailer Web service trusts the Web client to authenticate the originating user.

Because this application is based on the BSP and WS-Security specifications, alternative solutions such as Windows impersonation and delegation were not possible as alternatives for propagating the originating user’s identity.

Implementation View

The Sample Application is implemented as a Visual Studio solution, with the Web services defined by a combination of XML Schema and Web Services Description Language (WSDL).

Visual Studio Solution

The Visual Studio solution is divided into the following main sections:

- Application Blocks
- Sample Application
- Setup

Application Blocks

Code from Enterprise Library is distributed as part as the Sample Application source code. For more information about Enterprise Library, see Appendix A, “Enterprise Library Integration.”

Note: Only the Enterprise Library code required to support the Sample Application is installed. The application does allow all configuration files to be edited using the Enterprise Library Configuration Console, but to use the console, you need to install the entire Enterprise Library, available on MSDN at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/entlib.asp>.

Sample Application

The Sample Application section is divided into three internal folders:

- **Resources.** Defines common assemblies for the WS-Policy implementation and some common utilities.
- **Services.** Contains the implementations of all the Web services and their internal logic implementation.
- **Web Application.** Contains the implementation of the Web application for the user interface that uses the Retailer Web service.

Setup

The Setup section defines one assembly named **Installers** that is used to install the local computer Windows Event Log source, using the Enterprise Library Configuration Application Block. It also includes a Scripts folder that contains a set of scripts that installs certificates on the local computer, configures virtual directories pointing to the Web services folders and the application folder, and creates a database for use by the application. The entry point for the setup scripts is Setup.cmd, which detects the version of the operating system and executes the appropriate scripts.

Web Service Definitions

This section details the WSDL and XML Schema files that are used to define each of the Web services. None of the Sample Application architecture's WSDL files define a service or port. In other words, the `<wsdl:definitions>` element does not contain a `<wsdl:service>` element, which normally contains the service address. Instead, the Configurator gets the addresses from a UDDI registry or from its own EndPoints.xml file, depending on the user's choice on the Demo Configuration page.

Retailer

The following WSDL and XML Schema files are used to define the Retailer Web service:

- <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Retailer.wsdl>
- <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/RetailOrder.xsd>
- <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/RetailCatalog.xsd>

The Web service also imports Configuration.wsdl and Configuration.xsd.

In the Sample Application, the default page of the client Web application (reached at a URL such as <http://localhost/wsiscmv1/Retailer/>) displays hyperlinks to two WSDL descriptions of the Retailer Web service:

- Document/literal (recommended)
- RPC/literal

The two documents describe exactly the same Web service, not variations, so they produce identical SOAP structures.

Warehouse

The following WSDL and XML Schema files are used to define the Warehouse Web service:

- <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Warehouse.wsdl>
- <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Warehouse.xsd>

The Web service also imports Configuration.wsdl and Configuration.xsd.

Each warehouse Web application's default page (reached at a URL such as <http://localhost/wsiscmv1/WarehouseA/>) displays hyperlinks for equivalent document-style and RPC-style WSDL documents.

Manufacturer and Warehouse Callback

The following WSDL and XML Schema files are used to define the Manufacturer and Warehouse Callback Web services:

- <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/Manufacturer.wsdl>
- <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/ManufacturerPO.xsd>
- <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-10/ManufacturerSN.xsd>

The Web services also import Configuration.wsdl and Configuration.xsd.

Configurator

The following WSDL and XML Schema files are used to define the Configurator Web service:

- <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Configurator.wsdl>
- <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Configurator.xsd>

The Web service also imports Configuration.xsd.

Logging Facility

The following WSDL and XML Schema files are used to define the Logging Facility Web service:

- <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/LoggingFacility.wsdl>
- <http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/LoggingFacility.xsd>

Deployment View

Figure 4.5 illustrates the deployment configuration used by the WS-I Sample Application Working Group when specifying requirements. In this configuration, the Retailer services and Warehouse services are assumed to be deployed within a single data center and the Web client (Demo System) and Manufacturing System are assumed to each reside outside that boundary.

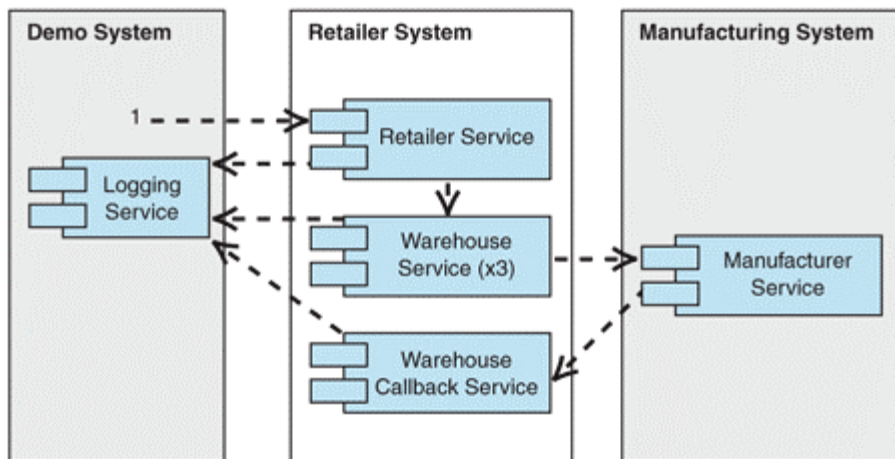


Figure 4.5

Retailer system deployment diagram

Demo System

The demo system is a client Web application that provides an interface for users to order items from a retailer. With it, a user can choose how the Web service providers are obtained. The application includes one or more application servers, hosting the Presentation tier of the retailer application, the Configurator service, and the Logging service, which is an infrastructure service to aid testing.

Note: Securing the connection between the customer and the demo system is out of scope of the BSP 1.0, because it focuses on securing Web services.

Retailer and Warehouse Systems

The Retailer system consists of a series of Web services that allow trusted clients to access the product catalog of the Retailer. Clients can then order products; the products are ordered by additional calls to Web services that support each of the warehouses. The Retailer system consists of one or more application servers, hosting the Retailer service; the Warehouse A, Warehouse B, and Warehouse C services; and the Warehouse Callback service.

Note: Although the Warehouse Callback service is part of the Retailer system, its WSDL port type is defined in the Manufacturer system.

Manufacturer System

The Manufacturer system provides a Web service for warehouses to use when they run low on inventory. The Manufacturer system consists of one or more application servers, hosting the Manufacturer A, Manufacturer B, and Manufacturer C services.

Sharing Configuration Between Web Services

The Manufacturer, Retailer, Warehouse, and Warehouse Callback Web services use a configuration header to share the following:

- A unique **UserId** generated by the demo system.
- A list of dynamically obtained service endpoints mapped to keys. Every Web service, except the Configurator Web service, is listed.

The Web client Web application establishes a user's configuration. The user's choices for endpoints come from the Configurator Web service.

A SOAP header containing an element of type **ConfigurationType** is shown in the following code example.

```
<soap:Header>
  <Configuration xmlns="http://www.ws-i.org/SampleApplications
/SupplyChainManagement/2002-08/Configuration.xsd">
    <UserId>76b4a291-87f6-473a-9923-ff23139abccb</UserId>
    <ServiceUrl Role="LoggingFacility">
      http://localhost/wsiscmv1/LoggingFacility/LoggingFacility.asmx
    </ServiceUrl>
    <ServiceUrl Role="Retailer">
      http://localhost/wsiscmv1/Retailer/Retailer.asmx
    </ServiceUrl>
    <ServiceUrl Role="WarehouseA">
      http://localhost/wsiscmv1/WarehouseA/Warehouse.asmx
    </ServiceUrl>
    <ServiceUrl Role="WarehouseB">
      http://localhost/wsiscmv1/WarehouseB/Warehouse.asmx
    </ServiceUrl>
    <ServiceUrl Role="WarehouseC">
      http://localhost/wsiscmv1/WarehouseC/Warehouse.asmx
    </ServiceUrl>
    <ServiceUrl Role="ManufacturerA">
      http://localhost/wsiscmv1/ManufacturerA/Manufacturer.asmx
    </ServiceUrl>
    <ServiceUrl Role="ManufacturerB">
      http://localhost/wsiscmv1/ManufacturerB/Manufacturer.asmx
    </ServiceUrl>
    <ServiceUrl Role="ManufacturerC">
      http://localhost/wsiscmv1/ManufacturerC/Manufacturer.asmx
    </ServiceUrl>
  </Configuration>
</soap:Header>
```

The **UserId** provides a mechanism for user session tracking during interoperability testing.

To see the usefulness of placing a list of service URLs in a SOAP header (as shown in the preceding code example), consider the example of the Logging Facility Web service. Other Web services need to log events to the Logging Facility. The configuration offers a common place where they can obtain that service's location.

A configuration SOAP header demonstrates one way a group of Web services can coordinate the sharing of metadata while minimizing coupling and reducing the number of needed service calls. By adding this information to the regular flow of Web service calls, the Web client does not need to make special calls to various services just to pass it along. Likewise, the Web client does not need to be turned into a service so that the services can query it for user configuration.

The services and Web client need to agree to share configuration information in a specific way. Accordingly, every Web service that exposes configuration information should reference the following:

- *<http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Configuration.wsdl>*
- *<http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Configuration.xsd>*

Note: Configuration.wsdl does not define a standalone Web service; instead, it defines a fragment to be referenced by other WSDL files.

Summary

The Sample Application is a distributed application that consists of a series of interdependent Web services and databases. To help you understand the nature of the application, this chapter examined the architecture from a number of different views. By examining the application from each of the different views, you can gain a more complete understanding of the nature of the application architecture.

5

Policy Usage in the Sample Application

Microsoft Web Services Enhancements (WSE) version 2.0 allows Web services to be secured either imperatively (using a comprehensive set of APIs), or using declarative security policy. The Sample Application standardizes on policy for implementing its security requirements:

- Policy provides an effective separation of concerns between the business logic of a service and the logic needed to verify and enforce security requirements.
- Policy allows Web services to validate that incoming messages conform to a security policy. It also allows outgoing messages to be secured automatically, without your developers having to learn and write complex code.
- WSE's policy framework can be extended. This means that custom requirements can be defined as extensions to the policy framework and reused by all applications.
- Migration to WSE 3.0 and Indigo will be simplified because policy is completely independent of a services' business logic.

Note: WSE 3.0 will have a different model for declaratively specifying security requirements than is currently implemented in WSE 2.0 SP3.

As broader agreement is reached about the implementation of a standard policy framework, Web services will specify explicit security requirements that can be verified and enforced automatically. The policy framework will supplement the structural contract that WSDL currently provides with a policy that can describe the semantic requirements of services.

Until there is agreement on a standard implementation of policy, Web services need to communicate their security requirements to calling applications out of band. The members of the WS-I Basic Security Profile (BSP) 1.0 Sample Application Working Group chose to use a design document to capture and communicate the security requirements for each service. This design document will be made available through the WS-I later this year (2005).

This chapter examines how the Sample Application uses policy, and discusses the WSE Policy Advisor, an unsupported tool from Microsoft Research that examines policy files and generates a report to make security recommendations.

Policy Assertions

A policy assertion is an individual preference, requirement, capability, or other general characteristic. WS-SecurityPolicy and WS-PolicyAssertions are specifications that define standard sets of policy assertions that can be used within a policy expression. WS-PolicyAssertions and WS-SecurityPolicy define some standard policy assertions that address the common needs of Web services applications.

The WSE for Microsoft .NET has built-in support for the following policy assertions that are part of WS-SecurityPolicy:

- **SecurityToken.** This specifies which kind of security token should be used.
- **Integrity.** This specifies that a message should be signed.
- **Confidentiality.** This specifies that a message should be encrypted.
- **MessageAge.** This specifies the age limit of a message.

WSE also has built-in support for MessagePredicate, which is part of WS-PolicyAssertions. MessagePredicate is used to ensure that messages conform to a policy assertion.

Note: For more detailed information about WS-Policy, see the following resources:

- Skonnard, Aaron. "The XML Files: WS-Policy and WSE 2.0 Assertion Handlers." *MSDN Magazine*, March 2004. <http://msdn.microsoft.com/msdnmag/issues/04/03/XMLFiles/>
 - "Understanding WS-Policy" on MSDN at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/understwsapol.asp>
-

Policy Mappings

Policy mappings declaratively define the relationship between policy assertions and Web service endpoints. Within each endpoint, multiple operations are specified, and a **defaultOperation** element is used to refer to all operations within an endpoint for which no operation element is defined.

Note: Be careful defining a **defaultOperation** element that includes **policy = ""**. By specifying an empty string in this way, you would ensure that no policy is run when an action does not correspond to any of the specified **requestActions**.

The **requestAction** attribute is used to represent the action of an operation. This corresponds to the value in the `wsa:Action` SOAP header defined by WS-Addressing, or if that is missing, the `SOAPAction` HTTP header for SOAP over HTTP.

Note: By default, the ASMX Web services framework uses the `SOAPAction` HTTP header to determine which Web method (within a single Web service endpoint) the request is for. If no `SOAPAction` header is specified (empty string ""), an exception is thrown, unless the Web method specifies a **RoutingStyle** of **RequestElement** in the **SoapDocumentService** attribute.

Both the **operation** and **defaultOperation** elements allow a separate policy to be specified for the operation's request message, response message, and fault message using the `policy` attribute of the request, response, and fault elements respectively.

The following XML example is from the Retailer service endpoint. The **ShipGoods** operation is defined using the **requestAction** attribute as are the **GetCatalog** and **SubmitOrder** operations. In each case request, response, and fault policies are defined.

```
<mappings xmlns:wse="http://schemas.microsoft.com/wse/2003/06/Policy">
  <defaultEndpoint>
    <!-- ShipGoods operation -->
    <operation requestAction="http://www.ws-i.org/SampleApplications
/SupplyChainManagement/2002-08/Warehouse.wsdl">
      <request policy="#Request-ShipGoods-Sign-X.509" />
      <response policy="#Response-ShipGoods-Sign-X.509" />
      <fault policy="#FaultLogging" />
    </operation>
    <operation requestAction="getCatalog">
      <request policy="#Request-getCatalog-Sign-X.509" />
      <response policy="#Response-Sign-X.509-Encrypt-X.509" />
      <fault policy="#FaultLogging" />
    </operation>
    <operation requestAction="submitOrder">
      <request policy="#Request-submitOrder-Sign-X.509-Encrypt-X.509" />
      <response policy="#Response-Sign-X.509-Encrypt-X.509" />
      <fault policy="#FaultLogging" />
    </operation>
  </defaultEndpoint>
</mappings>
```

Custom Assertions

Custom assertions can be used to extend the policy assertions built into WSE 2.0. The Sample Application uses several custom assertions to meet the security requirements defined in the BSP. These assertions extend the **SecurityToken** and **Confidentiality** built-in policy assertions.

To use define a custom assertion, it must be added to the policy section of the Web.config file for the Web service, as shown in the following code example from within the **Microsoft.web.services2** element.

```
<policy>
  <cache name="Configuration\retailerPolicyCache.config" />
  <assertion name="wsisa:ConfidentialityEx"
type="Microsoft.Practices.WSI.SCM.BSP10.CustomPolicyAssertions.ConfidentialityExAs
sertion, Microsoft.Practices.WSI.SCM.BSP10.CustomPolicyAssertions,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" xmlns:wsisa="http://www.ws-
i.org/SampleApplications/SupplyChainManagement/2004-12/wsisa"/>
  <assertion name="wsisa:FaultLogging"
type="Microsoft.Practices.WSI.SCM.BSP10.CustomPolicyAssertions.FaultLoggingAsserti
on, Microsoft.Practices.WSI.SCM.BSP10.CustomPolicyAssertions, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=null" xmlns:wsisa="http://www.ws-
i.org/SampleApplications/SupplyChainManagement/2004-12/wsisa" />
</policy>
```

ConfidentialityEx Custom assertion

The WSE built-in **Confidentiality** assertion supports the encryption of only the **<body>** element and UsernameTokens. The **ConfidentialityEx** custom assertion extends this functionality and allows the encryption of the **Signature** element and any other custom headers. It uses the same **SecurityToken** specified in the **Confidentiality** policy assertion, so it requires a **Confidentiality** policy assertion located in the same policy definition to work properly.

This custom assertion is useful to prevent against plain text guessing attacks against plain text digital signatures. For more information, see Section 13.2.2, “Combining Security Mechanisms” of the Oasis document, *Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)*. You can download this document at <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>.

The services that use these assertions are:

- Retailer
- Warehouses
- Manufacturers

The following XML code example shows the **ConfidentialityEx** custom assertion being used to encrypt the signature element and a custom header called **StartHeader**.

```
<wsisa:ConfidentialityEx wsp:Usage="wsp:Required"
    xmlns:wsisa="...wsisa ns">
  <wssp:MessageParts Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
    wsp:Signature() wsp:Header(s:StartHeader)
  </wssp:MessageParts>
</wsisa:ConfidentialityEx>
```

The **ConfidentialityEx** custom assertion works for both senders and receivers. On the sender side, it encrypts the specified parts with the same **SecurityToken** used by **Confidentiality** policy assertion. On the receiver side, it validates the incoming message parts to ensure that the specified message parts are encrypted.

The **SecurityToken** used for encryption is looked for under the **SecurityElements** collection from the current **SoapContext**. If a **SecurityToken** is not found, **SoapContext** searches the *PolicyEnforcementSecurityTokenCache* for a match. If again, a **SecurityToken** is not found, an exception is thrown.

Custom Token Managers

The Sample Application also includes two custom token managers, the X.509 Security Token Manager, and the Username Security Token Manager. This section examines both of these managers.

X.509 Security Token Manager

The Sample Application includes a custom X.509 Security Token Manager. This manager is implemented using an **X509SecurityTokenManagerBroker** class that derives from the X509SecurityTokenManager WSE built-in class. The manager allows you to select a specified X.509 security token based on configured rules for integrity and confidentiality assertions.

The manager usually operates on outgoing messages, where you need to choose between several security tokens that can be used to sign or encrypt the outgoing message. The manager is used by the warehouse and manufacturer Web services, because they need to use the corresponding X.509 certificate that belongs to each instantiated endpoint. Because one single policy configuration file is used for the three endpoint instances (A, B, and C), the manager is used to choose from three different certificates to sign and encrypt the messages.

The following code example demonstrates how to configure the built-in integrity assertion that is added to a sender-side request policy or to a receiver-side response policy definition. The example shows three possible certificates to choose from based on the configured criteria for integrity assertions. In this example, a certificate is selected if the certificate CN (**FriendlyName** property) matches the application name (**AppDomain FriendlyName** property). If it does, the security token is used for the corresponding assertion.

```
<wssp:Integrity wsp:Usage="wsp:Required">
  <wssp:TokenInfo>
    <wsp:OneOrMore wsp:Usage="wsp:Required" >
      <wssp:SecurityToken wsp:Preference="100">
        <wssp:TokenType>
          http://docs.oasis-open.org...-1.0#X509v3
        </wssp:TokenType>
        <wssp:TokenIssuer>0="...token issuer..."</wssp:TokenIssuer>
        <wssp:Claims>
          <wssp:SubjectName MatchType="wssp:Exact">...WarehouseA
          </wssp:SubjectName>
          <wssp:X509Extension OID="2.5..." MatchType="wssp:Exact">
            7vm8ogrbcXnEWXzQiQLPrT17sRg=
          </wssp:X509Extension>
        </wssp:Claims>
      </wssp:SecurityToken>
    <wssp:SecurityToken wsp:Preference="100">
      <wssp:TokenType>
        http://docs.oasis-open.org...-1.0#X509v3
      </wssp:TokenType>
      <wssp:TokenIssuer>0="...token issuer..."</wssp:TokenIssuer>
      <wssp:Claims>
        <wssp:SubjectName MatchType="wssp:Exact">...WarehouseB
        </wssp:SubjectName>
        <wssp:X509Extension OID="2.5..." MatchType="wssp:Exact">
          dsDG8vIsKXY76YBz07DSi2iNUWQ=
        </wssp:X509Extension>
      </wssp:Claims>
    </wssp:SecurityToken>
  <wssp:SecurityToken wsp:Preference="100">
    <wssp:TokenType>
      http://docs.oasis-open.org...-1.0#X509v3
    </wssp:TokenType>
    <wssp:TokenIssuer>0="...token issuer..."</wssp:TokenIssuer>
    <wssp:Claims>
      <wssp:SubjectName MatchType="wssp:Exact">...WarehouseC
      </wssp:SubjectName>
      <wssp:X509Extension OID="2.5..." MatchType="wssp:Exact">
        AvGcfj7C0p3+kV7oZideeyFjQSY=
      </wssp:X509Extension>
    </wssp:Claims>
  </wssp:SecurityToken>
</wssp:Integrity>
```

UsernameToken Manager

WSE provides a default **UsernameTokenManager** that authenticates all UsernameToken security tokens in a received SOAP message against a Windows account. WSE calls the Win32 LogonUser function for this authentication. If it succeeds, a Windows principal is assigned to the **Principal** property of the UsernameToken.

The Sample Application does not authenticate the users specified in the UsernameTokens — instead, it is simply using this as a mechanism to flow identity across application tiers. To facilitate this functionality, a custom

UsernameTokenManager has been created in the Resources/CustomPolicyAssertions section of the Visual Studio project. For more information, see the “Dominant Patterns for Identity Propagation and Service Authentication” section of Chapter 4, “Sample Application Architecture.”

WSE Policy Advisor

In Web services and clients implemented with WSE 2.0, many aspects of SOAP message processing can be determined by declarative XML configuration and policy files, separate from imperative code.

This separation allows you to customize the configuration of your environment without recompiling code. It also allows you to separate security-critical processing from code. However, the flexibility of the configuration and policy formats also allows for a range of subtle errors, which can lead to a number of security vulnerabilities. Specifically, the environment is vulnerable to XML rewriting attacks, including man-in-the-middle and dictionary attacks.

WSE Policy Advisor is an unsupported plug-in for WSE 2.0 available from Microsoft Research at <http://research.microsoft.com/projects/Samoa>. It is a static analyzer to help with security reviews of WSE-based Web services. You can invoke the plug-in either from the WSE Configuration Editor or as a stand-alone tool.

The WSE Policy Advisor examines the policy files that configure WSE to generate a report that summarizes their security properties, highlights typical security risks, and provides some remedial advice. Like most security tools, it can generate false alarms. Conversely, the absence of warnings does not guarantee the absence of security vulnerabilities. Even so, WSE Policy Advisor can prove very useful in finding a range of vulnerabilities to XML rewriting attacks that otherwise may go undetected.

Note: The Microsoft Research WSE Policy Advisor is available for WSE 2.0 only.

Format of the Report

The advisor produces an HTML report, either on a single policy file, or on the combination of a configuration file and its associated policy file. The report includes the results of running approximately 35 different security queries on the advisor's input files.

First, the report includes advice about the configuration file. Advice may include:

- Test root certificates are allowed.
- Replay detection is not enabled for a **SecurityTokenManager** of type **wsse:UsernameToken**.

Second, the report describes every mapping from SOAP endpoint and operations to request, response, and fault policies, with details and advice for each mapping. These mappings determine which policy applies to each incoming or outgoing SOAP message. Warnings about mappings include:

- This mapping does not authenticate requests.
- This mapping has no policy for responses.
- This mapping does not authenticate responses.
- This mapping has no policy for faults (although it has a policy for responses).

Finally, the report describes every policy defined in the policy file. For each policy, the report first includes a table that indicates, for typical elements found in SOAP messages, whether those elements are mandatory, signed-if-present, and encrypted-if-present according to that policy. The report also includes specific advice on the policy. Warnings about policies include:

- This policy accepts certificates from any issuer.
- This policy uses an unencrypted **<wsse:UsernameToken>** for signing messages.

Part of a Typical Report

The WSE Policy Advisor window is a custom browser for the report. The screenshot in Figure 5.1 shows part of the report produced from a typical policy file. (This file, named *HeaderWarnings-policyCache.xml*, is one of the samples distributed with the advisor.) The window divides into three panes. The pane at the lower-left can be used to navigate around the report: clicking a branch in the tree updates the two other panes. The scrollable pane at the top displays details from the report. The scrollable pane at the lower-right displays relevant XML code examples extracted from the configuration and policy files.

The screenshot shows the WSE Policy Advisor window with the title bar 'WSE Policy Advisor'. The main content area displays a report titled 'Policy body-and-headers-optionally-signed'. The report includes a table with the following data:

SOAP Element	Mandatory	Signed If Present	Encrypted If Present
wsp:Header(wsa:To)	no	yes	N/A
wsp:Header(wsa:Action)	no	yes	N/A
wsp:Header(wsa:ReplyTo)	no	no	N/A
wsp:Header(wsa:FaultTo)	no	no	N/A
wsp:Header(wsa:MessageID)	no	yes	N/A
wsp:Header(wsa:From)	no	no	N/A
wsp:Header(wsa:RelatesTo)	no	no	N/A
wse:Timestamp()	no	yes	N/A
wse:UsernameToken()	N/A	no	no
wsp:Body()	no	yes	no

Below the table, the 'Report Contents' pane shows a tree view with the following structure:

- Report Contents
 - policy file C:\Program Files\Microsoft WSE\w2.0\WsePolicyAdvisor\Sample
 - endpoint http://stockservice.contoso.com/test2/service.asmx
 - operation http://stockservice.contoso.com/action1
 - operation http://stockservice.contoso.com/action2
 - operation defaultOperation
 - policy only-body-signed
 - policy body-and-headers-optionally-signed
 - This policy has no <Confidentiality> assertion.
 - This policy accepts messages without a <wsa:To> or <wsa:Action> header.
 - This policy accepts messages without a <wse:Timestamp> or <MessageID> header.
 - policy body-and-headers-mandatory-and-signed

The right pane displays the XML code examples extracted from the configuration and policy files:

```
<wsp:Policy>
  <wsp:SecurityToken>
  </wsp:SecurityToken>
  <wsp:TokenInfo>
  </wsp:TokenInfo>
  <wssp:MessageParts
    Dialect="http://schemas.xmlsoap.org/2003/11/transport/ws-policy/"
    <wsp:Header(wsa:To)
    <wsp:Header(wsa:Action)
    <wsp:Header(wsa:MessageID)
    <wse:Timestamp()
  </wssp:MessageParts>
  <wssp:Integrity>
  </wssp:Integrity>
</wsp:Policy>
```

Figure 5.1

A report from the WSE Policy Advisor

This screenshot is the result of clicking the policy *body-and-headers-optionally-signed* in the lower-left pane. This policy has an `<Integrity>` assertion with a `<MessageParts>` element to require signing of certain SOAP elements, though only if they are present. The policy has no `<Confidentiality>` assertion (which would require encryption of certain elements of the message), and no `<MessagePredicate>` assertion (which would require the presence of certain optional header elements). The policy triggers three warnings, shown in blue.

The first warning indicates there is no confidentiality protection on messages conforming to the policy. Depending on the application, confidentiality may or may not be required, so this may or may not be a false alarm. Still, the need for confidentiality should be considered in a security review.

The second warning more likely indicates a vulnerability. Clicking the warning yields an explanation and advice concerning a redirection attack.

The third warning, similar to the second, concerns the risk of a replay attack.

Summary

The Sample Application uses policy to implement its security requirements. This has a number of advantages, including the ability to customize the configuration of your environment without recompiling code. However, until there is agreement on a standard implementation of policy, Web services need to communicate their security requirements to calling applications out of band.

This chapter examined how the Sample Application uses policy, and also discussed the WSE Policy Advisor, an unsupported tool from Microsoft Research that can be very useful when examining policy. The tool generates a report that makes a number of security recommendations about the policy used.

6

Designing Web Services for Interoperability and Resilience

Web service specifications are still maturing. In some cases, such as SOAP and WSDL, the specifications have been ratified through organizations such as Oasis or the W3C, and even profiled by the WS-I, but in other cases, the specification is currently in review by the WS-I, or is yet to finish review by Oasis or the W3C.

In addition, different vendors' products are currently offering different combinations of the WS* specifications. This makes it harder to interoperate with the broad set of Web service specifications. Web Services Enhancements (WSE) version 2.0 Service Pack (SP) 3 provides access to many WS* capabilities now, with further functionality provided in WSE 3.0 (released shortly after Visual Studio 2005), and more functionality in the release of Indigo.

Despite the changing landscape, there are some guidelines that you can follow now that will assist you in designing Web services that are both interoperable across different vendors' platforms and more resilient to change as products evolve.

This chapter examines how you can design interoperable Web services and demonstrates how the Sample Application supports interoperability. It also demonstrates how to implement Web services so that they are resilient to changes in the underlying platform.

Designing Web Services for Interoperability

To understand how to design interoperable Web services that conform to the Basic Security Profile 1.0, you should have knowledge of some general guidelines for Web service interoperability. This section provides a high-level introduction to some of these matters, but is not intended to be fully comprehensive, because the focus is on the Basic Security Profile 1.0.

Many of the interoperability issues you may encounter were examined during the development of the Basic Profile 1.0. For information, see “Building Interoperable Web Services: WS-I Basic Profile 1.0” on MSDN at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsvcenter/html/WSI-BP_MSDN_LandingPage.asp. For more information about Web service interoperability, see the Web Services Interoperability page in the Web Services Developer Center on MSDN at <http://msdn.microsoft.com/webservices/building/interop/>.

Contract-First Development

A common challenge you will face when integrating applications that run on different platforms is how to translate data types from one platform to another. Different programming languages and operating systems support different data types, each with different names and characteristics. For example, an integer in Visual Basic 6.0 is 16 bits; while an integer in Visual Basic .NET and Visual C# is 32 bits. Similarly, if you expose a **float** data type from Java, it is not clear what the corresponding data type should be in the .NET Framework.

As you combine different data types into complex structures, you are increasingly likely to encounter translation problems between platforms. Examples of complex structures include collection classes, arrays, and platform-specific data types such as datasets. Even the treatment of NULL values can vary across platforms.

To help reduce the dependence on platform-specific data types, you can adopt the approach of designing the messages, first using an XML Schema (XSD) and then generating the representative platform-specific data types based on the message schema and WSDL information.

This approach was used to create the WS-I Basic Security Profile, allowing each vendor to generate platform-specific classes based on the platform-agnostic XSD representations of messages. This approach to designing messages and related contracts for services is commonly referred to as contract-first development.

This section provides a brief overview of two approaches to contract-based development. For more comprehensive information, see the following resources:

- Skonnard, Aaron. "Service Station: Contract-First Service Development." *MSDN Magazine*. May 2005.
<http://msdn.microsoft.com/msdnmag/issues/05/05/ServiceStation/default.aspx>
- Shohoud, Yasser. "Place XML Message Design Ahead of Schema Planning to Improve Web Service Interoperability." *MSDN Magazine*. December 2002.
<http://msdn.microsoft.com/msdnmag/issues/02/12/WebServicesDesign/default.aspx>

Message Oriented Design Using XSD

XSD provides a uniform platform-agnostic way of describing data types. In the Sample Application, XSD is used to define all the messages. For example, the message sent from the Web client to the retailer to place an order for a service contains a complex type that includes a product number, quantity, and price, as shown in the following code example.

```
<xsd:complexType name="PartsOrderItem">
  <xsd:sequence>
    <xsd:element name="productNumber" type="tns:productNumber" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="quantity" type="xsd:nonNegativeInteger" minOccurs="1"
maxOccurs="1"/>
    <xsd:element name="price" type="xsd:decimal" minOccurs="1"
maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

You can create an XSD file visually in Visual Studio. Figure 6.1 shows the file being created graphically in Visual Studio 2005.

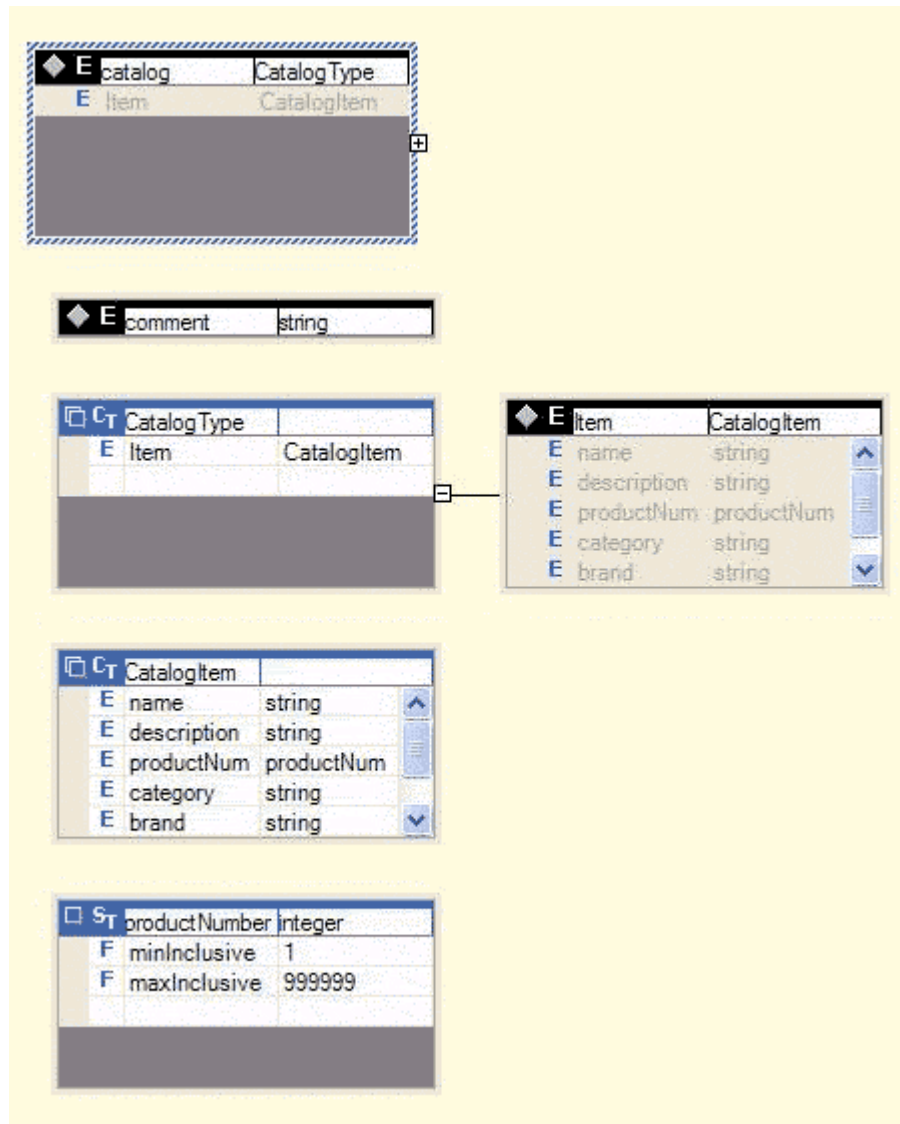


Figure 6.1
Creating an XSD file graphically in Visual Studio 2005

After you create the XSD file, you need to turn it into usable classes. To help you with this task, the Microsoft .NET Framework ships with an XSD tool named Xsd.exe. If you run the Xsd.exe utility against an XSD file with the /c switch, you can generate the appropriate classes automatically. The following code example shows the .NET type created from the previous XSD example.

```
namespace Microsoft.Practices.WSI.SCM.BSP10.Retailer.Messages
{
    /// <remarks/>
    [XmlType(Namespace=Constants.RetailerOrderNamespace)]
    public class PartsOrderItem : IListSerializer
    {
        #region Public Fields
        /// <remarks/>
        [XmlElement( "productNumber", DataType="int")]
        public int ProductNumber;

        /// <remarks/>
        [XmlElement( "quantity", DataType="unsignedShort")]
        public UInt16 Quantity;

        /// <remarks/>
        [XmlElement( "price" )]
        public System.Decimal Price;
        #endregion
    }
}
```

Note: The **PartsOrderItem** class implements the **IListSerializer** interface, which is used to convert messages into strings. The interface is used for logging purposes.

By using Xsd.exe, you automatically ensure that the method is prefixed with **XmlTypeAttribute** and **XmlRootAttribute**. These attributes specify how the data type appears on the wire after it is serialized and forwarded using Web services.

Note: There are also tools that are similar to Xsd.exe in Java-based toolkits. For IBM WebSphere, the Eclipse IDE includes a “Java Bean for XML Schema” wizard. For BEA, placing the XSD file into a schemas folder within a WebLogic Workshop project will cause objects (referred to as XML Beans) to be generated.

After the .NET classes are generated from the XSDs, you can then add them to your Web service. This is shown in the following code example of a Web service definition.

```
public Catalog GetCatalog([XmlElementAttribute("getCatalog",
Namespace=Constants.RetailerNamespace)] GetCatalog getCatIn)

    [WebMethod(MessageName="GetCatalog")]
    [SoapDocumentMethod(Constants.RetailerGetCatalogAction,
        RequestElementName="getCatalog",
        RequestNamespace=Constants.RetailerNamespace,
        Binding="RetailerSoapBinding",
        Use=SoapBindingUse.Literal,
        ParameterStyle=SoapParameterStyle.Bare)]
    [return: XmlElement("getCatalogResponse",
Namespace=Constants.RetailerNamespace)]
    public Catalog GetCatalog([XmlElementAttribute("getCatalog",
Namespace=Constants.RetailerNamespace)] GetCatalog getCatIn)
```

The Retailer service uses a single input parameter (**GetCatalog**) and uses the **SoapDocumentMethod** to specify that the message should be formatted as **Document**. It also uses the **SoapBindingUse** parameter to specify an encoding type of **Literal** and the **SoapParameterStyle** parameter to specify that parameters do not need to be encapsulated within one XML element within the Body element. Specifying **Document / Literal / Bare** ensures that the schema elements created are as expected by the schema author.

The WSDL service contract can be viewed by the .asmx file in a browser by appending “?wsdl” (without the quotation marks) to the URL.

While this approach to contract-based development is simpler than manually creating WSDL, it does still require developers to understand additional .NET attributes such as **XmlRoot**, **XmlType**, **XmlElement**, and the **SoapDocumentMethod**.

Contract Design Using XSD and WSDL

As an alternative to the message-based contract design approach, you can define the complete WSDL contract. This involves defining the service’s operations at an abstract level (referred to as a WSDL PortType) using XSD messages and then defining the binding to a particular transport.

After you create the appropriate WSDL information, you can use a tool to implement the corresponding Web service interfaces. The Microsoft .NET Framework SDK includes a tool named Wsd.exe for this purpose. The tool generates a Web service stub-class that implements the interface and a Web service proxy class that is used by clients.

Note: This approach to developing service contracts was used by the WS-I Sample Application Working Group. Each vendor is able to take the WSDL and generate platform-specific service interfaces and client proxies.

The current version of Visual Studio .NET does not include a WSDL designer, but any XML or text editor can be used for this purpose. To create your own WSDL, you will need to have a comprehensive understanding of the language and knowledge of how the WSDL is generated into .NET code. For more information about manually creating your Web service contracts, see the following resources:

- Skonnard, Aaron. "Service Station: Contract-First Service Development." *MSDN Magazine*. May 2005.
<http://msdn.microsoft.com/msdnmag/issues/05/05/ServiceStation/default.aspx>
- Shohoud, Yasser. "Place XML Message Design Ahead of Schema Planning to Improve Web Service Interoperability." *MSDN Magazine*. December 2002.
<http://msdn.microsoft.com/msdnmag/issues/02/12/WebServicesDesign/default.aspx>

Using Visual Studio Team Edition for Software Architects

In Visual Studio 2005, Visual Studio Team Edition for Software Architects includes tools for defining Web services as part of the Distributed System Designers. Using Application Designer, you can design and implement (generate skeleton code for) ASP.NET Web services using either a contract-first or code-first approach.

If you are developing a service based on a previously existing WSDL contract, you first add an ASP.NET application to the design surface and then select it and use the Create Web Service Endpoints from the WSDL feature. You specify a WSDL contract, and then the designer creates all the appropriate implementation code. This uses the same underlying frameworks as Wsd.exe, although it not only creates the Web service and message handling classes, it also puts them into your project and adds the required .asmx file. All you need to do is add the method body code that implements each operation. An add-in is also being developed that will add the ability to update a Web service to conform to a WSDL document that supports iterative contract development. This can be used when you own the contract under development and you want to create an implementation and keep it updated as you make changes to the contract.

Alternatively, if you are creating a new service, the tools let you define the Web service operations and arguments using common language runtime (CLR) types. These types can be created from XSD using Xsd.exe. After the design is created and implemented, ASP.NET will automatically generate the WSDL and message schemas from your code. The tools provide a great deal of control over the WSDL created, thereby simplifying the design process.

WS-I Test Tools

The WS-I test tools are used to test Web service artifacts to ensure conformance with particular profiles. Conformance is important for making sure that Web services are interoperable.

Basic Profile 1.1 Test Tools

The WS-I developed two test tools for use with the Basic Profile 1.1, a monitor and an analyzer, that are used in conjunction with each other for evaluating whether a Web service follows the Basic Profile guidelines. Both C# and Java versions of the tools are available. The tools do the following:

- The monitor captures and logs the messages between a Web service and a consumer.
- The analyzer examines the messages logged by the monitor and analyzes them to determine whether a Web service follows the Basic Profile guidelines.

Figure 6.2 illustrates the relationship between the monitor and the analyzer.

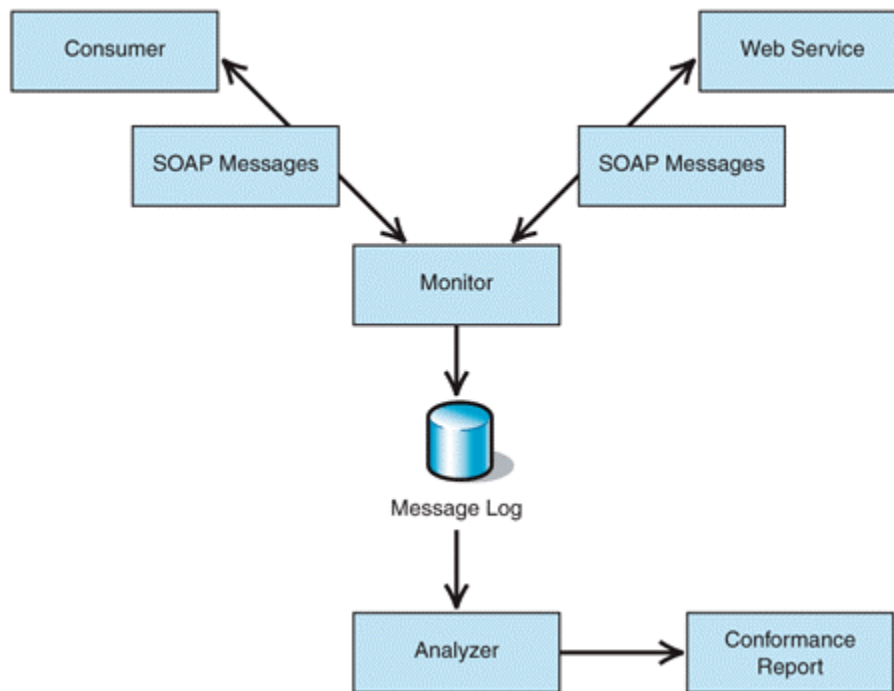


Figure 6.2

The WS-I test tools

Basic Security Profile 1.0 Test Tools

A simple “man-in-the-middle” interceptor, as used in the Basic Profile 1.1 test tools is not useful if encryption has been applied to portions of the SOAP envelope. Encryption changes the nature of the SOAP message so that unintended parties cannot read the content, meaning that the secured SOAP message is no longer obviously related to the original WSDL description, and is not intelligible without decryption. The WS-I Testing Tools team is currently developing an alternative approach for testing messages secured using WS-Security for conformance with the Basic Security Profile 1.0.

Designing a Web Service for Resiliency

Web service specifications continue to evolve and mature. Some specifications, such as WS-Security, have been adopted by multiple vendors and are in the process of being profiled by the WS-I, while other specifications are still going through the standards process pending broader adoption.

The Web Services Enhancements (WSE) for Microsoft .NET builds on the Microsoft .NET Web services foundation. As these advancements gain broad adoption and mature, they will be absorbed into the .NET Framework and Visual Studio. Code written for this version of the WSE is not guaranteed to be compatible with the future versions of the product. In other words, backward compatibility cannot be assured from version to version.

Note: For a complete description of the WSE support strategy, see “Getting Started (WSE for Microsoft .NET)” on MSDN at <http://msdn.microsoft.com/library/?url=/library/en-us/wse/html/f991ad3d-f574-4085-8a61-98326ff206ed.asp>.

As a result of the potential for change in WSE, developers can consider writing their services in such a way that they are more resilient to change — in particular, focusing on minimizing the impact of changes to their business logic. This section provides some examples of how such techniques were applied in the Sample Application.

There are a number of measures you can take, using current technologies, to help make your Web services interoperable and continue to be interoperable in the future. By designing your Web services according to a set of guidelines, you can increase platform interoperability now, and help ensure that the services business logic is more resilient to change as the underlying transports change from WSE 2.0 to WSE 3.0 and eventually, to Indigo.

WSE 2.0, WSE 3.0, and Indigo Interoperability

The Sample Application is currently written using WSE 2.0 SP 3, although it is intended that the final version of the application will be written using WSE 3.0, which will be released alongside Visual Studio 2005.

The following guidelines should be noted for interoperability between current and future Microsoft products:

- Microsoft intends to provide interoperability between WSE 3.0 and Indigo.
- Microsoft will not guarantee interoperability between WSE 2.0 and WSE 3.0.
- Microsoft will not guarantee interoperability between WSE 2.0 and Indigo.
- Code written in WSE 3.0 will be interoperable within Indigo; however, there will not be a way to automatically migrate source code from WSE 3.0 to Indigo.

It may be possible to develop Web services using WSE 2.0 in such a way that they are interoperable with WSE 3.0 and Indigo by using only a reduced set of specifications — specifically, SOAP 1.1, WSDL 1.0, and WS-Security 1.0 — but interoperability in this situation is not guaranteed.

Changes within WS-Addressing, WS-Trust, and WS-Secure Conversation may lead to interoperability issues between WSE 2.0 and later releases of products that incorporate this functionality.

Separation of Concerns

Separation of concerns is a core principle of software engineering, based on the concept that software should be developed so that different or unrelated responsibilities are separated from each other and can, as a result, vary independently. Design patterns often enforce the concept of separation of concerns.

This section includes some principals that have been applied to the Sample Application to allow the services to be robust and adapt to changes in the underlying products.

Separate Business Logic from Underlying Transport Using the Service Interface Pattern

You can separate the business logic from the underlying transport by using the Service Interface pattern. This pattern allows you to protect your business logic from a dependence on a transport that may change over time. Examples of how this pattern could be used include:

- Creating new service interfaces using WSE 3.0, replacing your WSE 2.0 services with minimal impact on your business logic.
- Creating multiple service interfaces that each support different transport capabilities and are interoperable on a subset of your applications.

The following code example shows how the Retailer Web service reduces the dependence between the service's interface and the business logic, by factoring the business logic into a separate class, **CatalogBusinessAction**.

```
[WebMethod(MessageName="GetCatalog")]
[SoapDocumentMethod(Constants.RetailerGetCatalogAction,
    RequestElementName="getCatalog",
    RequestNamespace=Constants.RetailerNamespace,
    Binding="RetailerSoapBinding",
    Use=SoapBindingUse.Literal,
    ParameterStyle=SoapParameterStyle.Bare)]
[return: XmlElement("getCatalogResponse",
    Namespace=Constants.RetailerNamespace)]
public Catalog GetCatalog([XmlElementAttribute("getCatalog",
    Namespace=Constants.RetailerNamespace)] GetCatalog getCatIn)
{
    try
    {
        return CatalogBusinessAction.GetCatalog( getCatIn );
    }
    catch( RetailerException clientException )
    {
        throw SoapUtility.GenerateSoapClientException( clientException );
    }
    catch( Exception serverException )
    {
        throw SoapUtility.GenerateSoapServerException( serverException );
    }
}
```

Note: To maximize separation, you can also have your business logic deployed in separate DLLs from other code. The Sample Application has separate DLLs for both the business logic and the service's interface.

For more information about the Service Interface pattern, see "Service Interface" on MSDN at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpatterns/html/DesServiceInterface.asp>.

Separate Cross-Cutting Concerns

Although many of the requirements of any complex system will be independent of one another, there are also likely to be other requirements that cut across multiple aspects of the system. Examples of such cross-cutting concerns include security requirements, logging, performance, and storage management. By centralizing these concerns, you can maximize their reuse and centrally update components when they change.

Implementing Security Requirements Using Policy

WSE 2.0 allows security requirements to be verified and applied in one of the following ways:

- **Imperatively.** A comprehensive set of APIs can be used by developers to secure a message and to verify that a received message is secured.
- **Declaratively.** Security requirements can be generated using tooling, and they can be edited independently of your business logic or your service's interface.

Migration to WSE 3.0 and Indigo will be simplified because policy is completely independent of a services' business logic. However, WSE 3.0 will have a different model for declaratively specifying security requirements than is currently implemented in WSE 2.0 SP3.

Note: No backward compatibility is guaranteed between major releases of WSE.

There is also tooling such as the Microsoft Research Policy Advisor. This tool acts like FxCop for Web services, meaning that it checks policy for known bad practices. This will help simplify your security reviews — because reviewing policy is much simpler than reviewing dozens of lines of code. For more information about the Policy Advisor, see Chapter 5, “Policy Usage in the Sample Application.”

Centralizing Web Service Fault Handling

In addition to trapping faults that occur within business logic, you can also use a custom fault handler to declaratively handle the management of SOAP faults, including logging, in a standard way.

If you centralize logging of SOAP faults, you reduce the dependency of other code on this functionality. By doing this, you can improve the functionality more easily as the platform evolves. The following code example shows the use of a policy SOAP fault handler.

```
<mappings xmlns:wse="http://schemas.microsoft.com/wse/2003/06/Policy">
  <defaultEndpoint>
    <!-- ShipGoods operation -->
    <operation requestAction="getCatalog">
      <request policy="#Request-getCatalog-Sign-X.509" />
      <response policy="#Response-Sign-X.509-Encrypt-X.509" />
      <fault policy="#FaultLogging" />
    </operation>
  </defaultEndpoint>
</mappings>
```

For more information, see the following resource:

- Skonnard, Aaron. “The XML Files: WS-Policy and WSE 2.0 Assertion Handlers.” *MSDN Magazine*. March 2004
<http://msdn.microsoft.com/msdnmag/issues/04/03/XMLFiles/>

Centralizing Logging Functionality

Logging is another common cross cutting-concern — supporting an organization's requirement for auditing. The Sample Application uses a **SoapExtension** for logging SOAP messages. The following code example shows the **SoapExtension**.

```
Microsoft.Practices.WSI.SCM.BSP10.Utilities.InterceptorTrace
```

The logging functionality is disabled by default, but it can be enabled by simply removing the XML comment. It is used for two reasons:

- During interoperability testing between different vendors, this allows a record of requests and responses to be captured so that you can troubleshoot interoperability issues.
- The graphical application monitor (see the GotDotNet Workspace for the Sample Application at <http://go.microsoft.com/fwlink/?LinkId=47780>) also uses this capability to allow people to observe the secured messages on the wire.

The following code example shows SOAP messages logged to the TraceLog database.

```
<system.web>
  <webServices>
    <soapExtensionTypes>
      <add type="Microsoft.Web.Services2.WebServicesExtension,
Microsoft.Web.Services2, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" priority="1" group="0" />
      <!-- Logs all SOAP messages to the TraceLog DB (close to the wire
format) -->
      <add
type="Microsoft.Practices.WSI.SCM.BSP10.Utilities.InterceptorTrace,
Microsoft.Practices.WSI.SCM.BSP10.Utilities, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null" priority="1" group="0" />
    </soapExtensionTypes>
  </webServices>
```

Centralize Exception Handling

Distributed applications can have different requirements for returning fault information to users. This frequently depends on whether the fault was expected. In many cases, a data validation fault is something that a service may expect to have to process and return a detailed error to the user, but an unexpected internal exception is something that the service may want to log and then return only a standard cleansed fault to the user.

To handle these different requirements, you can either repeat your exception handling logic within each Web service or you can create different custom classes for exception shielding. These classes can be updated independently of the business logic if the underlying transports requirements change, or even if your requirements for shielding exceptions change. The following code shows how client and server exception managers are used, depending on whether the exception is planned.

```
try
{
    return CatalogBusinessAction.GetCatalog( getCatIn );
}
catch( RetailerException clientException )
{
    throw SoapUtility.GenerateSoapClientException( clientException );
}
catch( Exception serverException )
{
    throw SoapUtility.GenerateSoapServerException( serverException );
}
}
```

Summary

With Web services specifications in varying states of maturity, and different vendors using different combinations of the WS* specifications, it can be a significant challenge to provide interoperability between Web services provided by different vendors. There is also the challenge of providing interoperability between the current implementation of WS* specifications provided in WSE 2.0 SP 3 and future implementations that will be provided in the future with Visual Studio 2005 and later, Indigo.

This chapter has provided guidance to help you design Web services that are interoperable across different vendors' platforms now and that are more likely to be interoperable as products change.

7

WS-I BSP Interoperability Guidance

This chapter discusses how you can develop interoperable Web services, focusing on how the Sample Application supports interoperability. It looks at the requirements of the WS-I Basic Security Profile (BSP) 1.0 and how they can affect interoperability. It also examines some specific interoperability issues that have emerged during the development of the Sample Application.

Basic Security Profile 1.0 Guidance

The BSP 1.0 attempts to increase interoperability by addressing the most common problems that implementation experience has revealed to date; it does not provide a guarantee of interoperability between secured Web services.

It is important to note that the BSP 1.0 covers only the specifications contained in WS-Security, focusing on SOAP message security, in particular data integrity and confidentiality. Other specifications, such as WS-Addressing, WS-Trust, and WS-SecureConversation have not been fully and formally reviewed by standards organizations such as OASIS or W3C. As a result, it is possible that other vendors have not implemented all of these specifications or implemented them in the same manner as Microsoft, making interoperability unlikely with those platforms at this time. There are also interoperability issues between Microsoft Web Services Enhancements (WSE) 2.0 Service Pack (SP) 3 and WSE 3.0.

Specifying Security Requirements

In many cases, the BSP 1.0 has very precise requirements that reduce the need for a service to expose its requirements to client applications. However, in other areas, the BSP 1.0 allows some flexibility and extensibility in the application of security to messages. This means that some agreement has to be reached over which mechanisms should be used for message exchange, including for the following areas:

- Encrypted elements
- Signed elements
- Data encryption algorithms
- Key encryption algorithms
- Security tokens
- Certificate extensions
- Certificate issuers

Note: WSE 2.0 supports all the encryption algorithms mentioned in the BSP 1.0.

Before a client application can communicate with a secure Web service, it must know the service's security requirements. These requirements differ for request and response messages.

For request messages sent from a client to a service, the client needs to know the answers to the following questions:

- Do request messages need to be signed to provide data integrity? If they do, what part of the message needs to be signed, and with what kind of token?
- Should request messages be encrypted to provide confidentiality? If they should, what parts of the message should be encrypted, and with which token should it be encrypted?
- What requirements are mandatory and which are optional?

For response messages sent from a service to a client, the service needs to know the answers to the following questions:

- Do response messages need to be signed to provide data integrity? If they do, what part of the message needs to be signed and with what kind of token?
- Should response messages be encrypted to provide confidentiality? If they should, what parts of the message should be encrypted, and with which token should it be encrypted?
- What requirements are mandatory and which are optional?
- How should SOAP Faults be secured?

The most logical way of specifying these requirements would be as an extension to the contract for the Web service, allowing client applications to understand the security requirements of a service using machine-readable policy. However, the Web service security stack incorporates a number of protocols, each of which are in various states of implementation by different vendors. No security policy description language or negotiation mechanism is in scope for the BSP, so much of security policy must be transmitted using out of band agreements.

The WS-I Sample Application Working Group uses a design document to communicate security requirements out of band. The design document specifies the exact requirements that each service must implement for request and response messages. These requirements specify data integrity and confidentiality requirements so that, for example, vendors know which parts of the message to sign and encrypt using which tokens.

Microsoft's implementation of the Sample Application is currently implemented using the WSE 2.0 SP3 implementation of WS-SecurityPolicy. WSE 2.0 SP3 supports all the options specified in the BSP document. However, while WSE 2.0's policy is loosely based on WS-Policy, it is important to note that it is not complete and has not been reviewed by a standards body such as OASIS or W3C. Nor has it been tested for interoperability by other vendors.

Even though WSE 2.0's implementation of policy is not sufficient for exposing a service's interface, it does have other major advantages, such as specifying your security requirements declaratively instead of imperatively (using WSE 2.0's comprehensive APIs). This allows you to specify security settings at deployment time, independent of your application development.

Note: The implementation of WS-Policy will change significantly in WSE 3.0.

Exchanging Security Tokens

In some cases, before an interaction, a client and a service may need to exchange security tokens with one another. For example, if a client needs to be authenticated using an X.509 certificate, both the client and the service must establish a common certificate authority (CA) that both trust. Or, if the client needs to encrypt a request, it will need a reference to the certificate containing the public key for the service.

As with specifying security requirements, the BSP 1.0 does not specify any mechanism for exchanging security tokens. The WS-Trust specification does provide a mechanism for exchanging security tokens dynamically, but because this is out of scope for the BSP 1.0, there is no guarantee of interoperability.

Interoperability Considerations for the Sample Application

During the development and testing of the Sample Application, the development team encountered a number of areas where specific actions needed to be taken to ensure interoperability. This section details those areas, and as such, it is not intended to be a complete description of interoperability issues covered by the BSP. The only security tokens currently used by the Sample Application are X.509 and user name tokens.

Note: The Sample Application is not designed to cover every aspect of the BSP. The test tools are designed to provide broader coverage of the BSP.

X509 Certificates

Oasis has released errata for the X509 token profile that, among other changes, modifies the way in which X509 tokens are referenced. However, the document was not normative, so it should not be implemented by vendors.

In some cases, vendors did implement these errata, which have caused some interoperability issues. The following references to X.509 tokens should be included in secure messages.

- BinarySecurityToken/@ValueType="....#X509v3"
- KeyIdentifier/@ValueType="....@X509SubjectKeyIdentifier"

However, if the errata are incorrectly implemented, the following references are included:

- BinarySecurityToken/@ValueType="....#X5093"
- KeyIdentifier/@ValueType="....@X509v3SubjectKeyIdentifier"

If the service you are consuming uses the Oasis errata, you should be able to contact product support for that vendor and request a service pack to address this issue.

Note: WSE 2.0 has not implemented the errata.

Verification of WS-Addressing Headers

WSE policy that is generated using the policy wizards automatically requires addressing headers to be present and signed. This is not an issue for clients that implement WS-Addressing, but for clients that do not, the generated policy will reject the messages when they call a service running on WSE 2.0 because the WS-Addressing headers are in the **MessagePredicate** policy assertions.

If your Web service will be called by clients other than .NET clients and those clients have not implemented WS-Addressing, you should consider removing the headers from the **MessagePredicate** assertion. However, you should also ensure that the headers remain in the **MessageParts** policy assertion. The **MessageParts** assertion does not require that the headers *are* provided, but if they are, the assertion requires that they must be signed.

Note: The WS-Addressing specification is currently being reviewed by the W3C.

Mapping Security Policy to a Web Service Operation

Most vendors have implemented some declarative mechanism for describing security requirements. This reduces the need for developers to have to understand a complex set of APIs that, if used incorrectly, may result in an insecure application. WSE 2.0 allows security policy to be specified declaratively for all the services within an application, or for a particular service, or for a particular operation on a service.

The ability to map security policy to an operation on a service currently results in several interoperability issues. WSE supports this capability through the **requestAction** policy statement which specifies the security policy to be applied based on a specific value provided by the client in the Web service addressing (WS-Addressing) **wsa:Action** header.

Not all vendors currently implement WS-Addressing. As a result, you might want to apply the same security policy across all operations on a service using the **defaultOperation** policy statement which does not require the use of WS-Addressing. If there are two operations that have different security requirements, you should consider splitting the operations into separate Web services. It is also possible to implement services and clients that use different security mechanisms for different operations exposed on the same endpoint by using WSE 2.0 APIs instead of policy.

If you are specifying security for your Web services using policy — which is a good practice — and you have distinct security requirements for operations within a service, you might want to test the operation of your Web service without the Web service addressing headers to ensure the policy functions correctly without the headers. The following code example shows how to remove the Web service addressing headers.

```
<!-- This goes into the Web.config file of the Web service or client with which
you want to remove WS-Addressing headers to help simulate a non .NET client -->
<microsoft.web.services2>
  <filters>
    <output>
      <add
type="CustomFilterLibrary.RemoveWSAHeadersFilter,Microsoft.Practices.WSI.SCM.BSP10.
WebClient" />
    </output>
  </filters>
```

```
// This is a WSE filter that removes WSA headers from output messages
using System;
using System.Xml;
using Microsoft.Web.Services2;
using System.Collections;
namespace CustomFilterLibrary
{
    // Class derives from the SoapOutputFilter class.
    public class RemoveWSAHeadersFilter: SoapOutputFilter
    {
        public RemoveWSAHeadersFilter()
        {
            // This is an empty constructor.
        }
        // Override the ProcessMessage method.
        public override void ProcessMessage(SoapEnvelope envelope)
        {
            if (envelope == null)
                throw new ArgumentNullException("No envelope!");
            for (int loop=envelope.Header.ChildNodes.Count - 1; loop>=0; loop--)
            {
                if (envelope.Header.ChildNodes[loop].NamespaceURI ==
Microsoft.Web.Services2.Addressing.WSAddressing.NamespaceURI)
                    envelope.Header.RemoveChild(envelope.Header.ChildNodes[loop]);
            }
        }
    }
}
```

UsernameToken

When the UsernameToken is used without a password, the nonce and created date fields are optional. However, WSE assumes the fields will always be there because WSE always provides the values regardless of whether the password is provided.

If the incoming UsernameTokens do not contain a nonce and created date fields, the replay cache within WSE will not function, so you may need to disable replay detection, as shown in the following code example. If you do this, you **MUST** ensure that your messages do not require replay detection, or you should develop one yourself using the signature digest as an alternative to the nonce.

```
<microsoft.web.services2>
  <security>
  .
  .
  .
    <!-- Removes the need for UserNameTokens to have the nonce and timestamp
    included. Other vendors are not
           including this data in their Tokens. I believe the Oasis spec only
    mandates this where a password is provided -->
    <replayDetection enabled="false" />
  </securityTokenManager>
```

Message Age and Clock Skew

The timestamp within a secure message is often critical to an effective replay detection strategy because it can be used to limit the number of items that need to be cached when checking for a replay attack.

To reject messages reliably, you must ensure that clocks on the sending and receiving servers are synchronized. Clocks of client and server computers must be synchronized to prevent messages from being rejected unduly. However, because not all vendors have the ability to ensure that their clocks are synchronized with Internet time services, WSE 2.0 has a clock skew function that allows you to specify a tolerance factor for time synchronization. The **MessageAge** predicate in WSE 2.0 policy specifies the maximum age in seconds of a message, based on the expiration field within the **Timestamp** header within the message, as shown in the following example.

```
<wssp:MessageAge wsp:Usage="wsp:Required" Age="300" />
```

While the clock skew function can help with interoperability, reducing the need for clock synchronization effectively reduces the security of your environment. Therefore, wherever possible, you should use Internet time services to ensure that your clocks are synchronized and remove the need for significant clock skew.

Basic Security Profile Detailed Review

For a detailed analysis of the Basic Profile (BP) 1.0, including the compliance of ASMX to the BP 1.0, see “Building Interoperable Web Services: WS-I Basic Profile 1.0” on MSDN at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsvcinter/html/wsi-bp_msdn_landingpage.asp. Note that this study will not be completed for BSP 1.0 until the BSP is finalized, and it will be performed only if there is sufficient customer demand.

If such an analysis would be interesting for you, please make sure you complete the questionnaire that accompanies this release.

Summary

To ensure that your Web services interoperate with the Sample Application, you must understand how the Sample Application supports interoperability. This chapter has examined how the requirements of the BSP 1.0 can affect interoperability with the Sample Application. It also examined some of the specific interoperability issues that emerged during the development and testing of the Sample Application.

Appendix A

Enterprise Library Integration

The Sample Application uses the Enterprise Library application blocks. Application blocks are reusable software components designed to assist developers with common enterprise development challenges. The Enterprise Library packages together the most widely used application blocks into a single integrated download.

Application Blocks

The Enterprise Library is designed so that its application blocks can function independently of one another. You need to add only the application blocks that your application will use; you do not need to add the entire library. The Sample Application uses the following application blocks from Enterprise Library:

- [Configuration Application Block](#). This application block allows applications to read and write configuration information. The Sample Application uses configuration information stored in an XML file. This information is read using the Configuration Application Block. In addition, each application block also uses the Configuration Application Block to read their configuration settings.
- [Data Access Application Block](#). This application block simplifies development tasks that require implementing common data access functionality. The Sample Application uses the Data Access Application Block to read and write information to a SQL database.
- [Exception Handling Application Block](#). This application block allows developers and policy makers to create a consistent strategy for processing exceptions that occur throughout the architectural layers of enterprise applications. The Sample Application uses the Exception Handling Application Block to respond to exceptions in a consistent fashion.

- **Logging and Instrumentation Application Block.** This application block allows developers to incorporate standard logging and instrumentation functionality in their applications. The Sample Application does not directly use the Logging and Instrumentation Application Block. This application block is used by the Logging Handler supplied by the Exception Handling Application Block when logging exception information.

Solution Structure and Application Block Projects

The Enterprise Library includes the source code for the application blocks. This means you can modify the application blocks to merge into your existing library or you can use parts of the Enterprise Library source code in other application blocks or applications that you build. The Sample Application solution uses unmodified application blocks.

Installing the Sample Application places the source code for the application blocks into the installation directory. The Visual Studio solution file for the Sample Application contains the projects and source code for the application blocks. Each of the Sample Application projects that depend on an application block contains a project reference to that application block.

Each application block also contains a project reference to a common class library (named **Common**). The Common project contains functionality that is shared by multiple application blocks, such as instrumentation support.

Configuration Console

The Enterprise Library also includes a graphical configuration tool named the Enterprise Library Configuration Console. The Configuration Console provides a way to change and validate application block settings without manually editing the XML configuration files where they are stored. The Configuration Console is included in the Enterprise Library download.

Note: Although the Enterprise Library code used by the Sample Application is included with the application, you must download Enterprise Library separately to use the configuration console. Enterprise Library is available for download at <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/entlib.asp>.

After you download and install Enterprise Library, you can launch the Configuration Console. To configure application block usage for the Sample Application, open the Web.config file for the Web client or Web service that you want to modify. To open the Web.config file, click the **Open Existing Application** button on the toolbar. In the **Open** dialog box, navigate to the folder that contains the Web.config file, click the Web.config file, and then click **Open**.

Configuration Application Block

The Configuration Application Block decouples the ability to read and write configuration data from the specifics of the underlying data store. It does this by using storage providers and transformers to transfer data between the application and the physical store. Storage providers are objects that can read or write to a particular physical store, such as an XML file or a SQL database. Transformers read the application's configuration data and then structure the data so that it conforms to the format that the storage provider expects. Figure A.1 shows the relationship between the application object that holds the configuration settings, the Configuration Application Block providers, and the storage location.

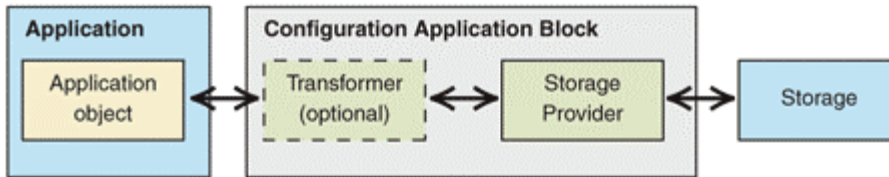


Figure A.1

Relationship between the Application object, the Configuration Application Block providers, and the storage location

You define the storage provider and transformer in a file that contains configuration metadata. For the Sample Application, this is the Web.config file. The metadata includes information such as the name of the configuration section, the storage location, and the type names of the objects to be used when reading or writing configuration settings. This means that you can change from one type of store to another by changing the information in the file. There is no need to rewrite your application.

Similarly, you can change the attributes of your store, such as its location, by changing the same file. Again, there is no need to modify your application code. Deciding where to store configuration data can be done during deployment and operation.

The application block is packaged with an XML file storage provider and an XML transformer. The Sample Application is configured to use these providers for reading configuration data (the Sample Application does not write configuration data.) The Web.config file contains the configuration metadata for each configuration section. The configuration settings for each configuration section are stored in a separate XML file. The Data Access Application Block, Exception Handling Application Block, and Logging and Instrumentation Application Block use the Configuration Application Block to read configuration settings. This means that there is an XML configuration file for each application block.

Sample Application Configuration Information

The Sample Application requires you to select the specific Web services to be used for the retailer, warehouses (A, B, and C), manufacturers (A, B and C) and logging facility entities. For each entity, you can choose the specific Web service, or you can choose a predefined set of Web services for all entities.

The configuration information for each Web service is referred to as an endpoint. Each endpoint contains the following configuration information:

- **Role.** This describes the specific entity type. It can be any of the following values: LoggingFacility, Retailer, WarehouseA, WarehouseB, WarehouseC, ManufacturerA, ManufacturerB, or ManufacturerC.
- **Description.** This is a text description of the Web service. For example, “Retailer on localhost” is the description of the retailer Web service located on the local computer.
- **Url.** This is the URL of the Web service.

The following code shows the configuration information for an endpoint as it appears in the XML file.

```
<Endpoint role="LoggingFacility"
description="LoggingFacility on localhost"
url="http://localhost/WsiScmSampleApplication/LoggingFacility/LoggingFacility.asmx" />
```

A collection of endpoints is called an endpoint set. The configuration for the Sample Application allows endpoint sets to be identified by name. The following code shows the configuration information for the endpoint set named “Local” as it appears in the XML file.

```
<EndpointSet name="Local">
<Endpoint ... />
<Endpoint ... />
...
<Endpoint ... />
</EndpointSet>
```

Web Client

The Web client allows you to select one of the following preconfigured endpoint sets:

- **Local endpoints.** This uses services from the local computer.
- **Microsoft endpoints.** This uses services available on a public Microsoft server.
- **Preconfigured mix of endpoints.** This uses services from various vendors.

The Web client uses the Configuration Application Block to read the configuration information for the preconfigured endpoint set. The Web client requests configuration information by specifying the name of a configuration section ("localEndpointConfiguration"). The Configuration Application Block reads the Web.config file to determine the storage location that contains the settings for that configuration section. The Sample Application is configured to use XML files, and the settings for configuration section **localEndpointConfiguration** are stored in the file **localEndpointConfiguration.config**.

Figure A.2 illustrates the Web client usage of the Configuration Application Block.

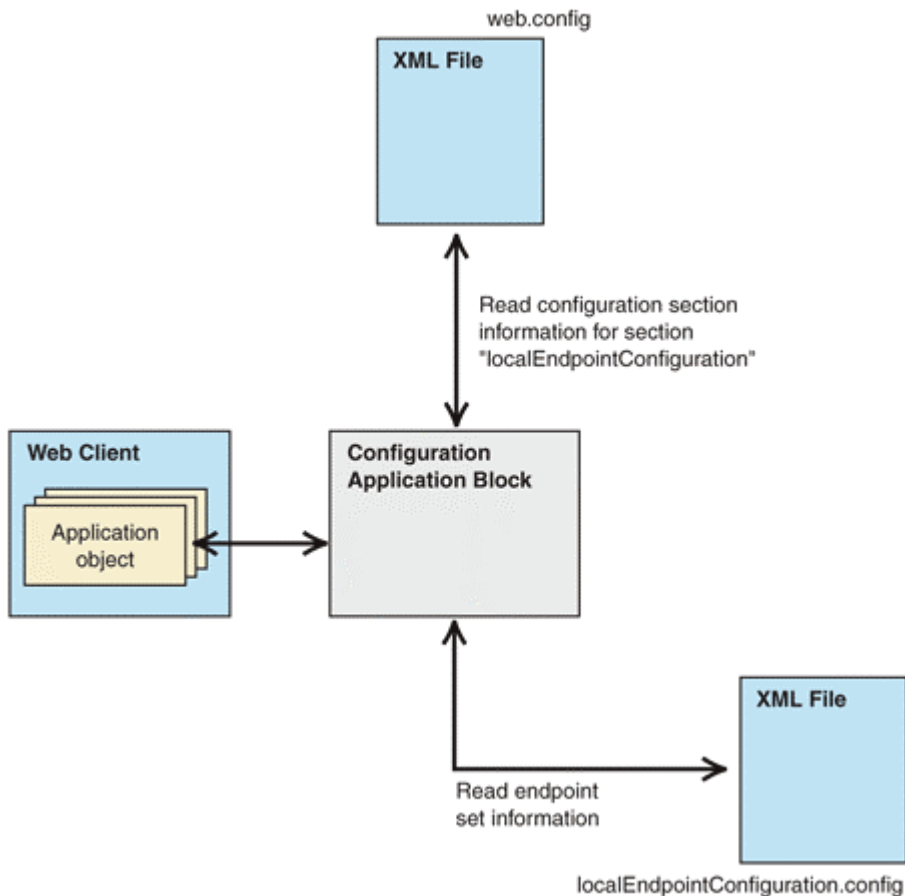


Figure A.2

How the Web client uses the Configuration Application Block

When the user clicks the **Run Demo** button with one of the preconfigured options selected, the Web client calls the Configuration Application Block to read the endpoint set information for that option. The Configuration Application Block reads the XML configuration file and returns an object of type **EndpointConfiguration** to the Web client. The following code examples show the Web client calling the Configuration Application Block to obtain the endpoint sets.

```
private const string EndpointConfigurationSection = "localEndpointConfiguration";
EndpointConfiguration configData = ConfigurationManager.GetConfiguration(
    EndpointConfigurationSection ) as EndpointConfiguration;
```

The class **EndpointConfiguration** is an XML serializable class that represents the configuration settings in object form. In this way, the developer of the Web client does not need to know the form or location of the configuration data. The Web client code contains only references to the objects that hold the configuration data returned by the Configuration Application Block.

Configuration Service

The Configuration Service provides a method to allow clients to retrieve the endpoint configuration data for the available Web services. The Configuration Service can return endpoint configuration data contained in a local XML file. It can also query UDDI to retrieve endpoint configuration settings. The use of a local XML file and UDDI is determined through the application settings in the Web.config file of the Configuration Service.

```
<appSettings>
  <add key="UDDIUrl" value="http://uddi.microsoft.com/inquire"/>
  <add key="UseLocalXML" value="true"/>
  <add key="UseUDDI" value="false"/>
</appSettings>
```

The Configuration Service uses the Configuration Application Block to read configuration settings from the local XML file.

The Configuration Service calls the Configuration Application Block, specifying the name of the configuration section **endpointConfiguration**. The Configuration Application Block reads the Web.config file to determine the storage location that contains the settings for that configuration section. The Configuration Service is configured to use XML files, and the settings for configuration section **endpointConfiguration** are stored in the file endpointConfiguration.config.

Figure A.3 illustrates the Configuration Service usage of the Configuration Application Block.

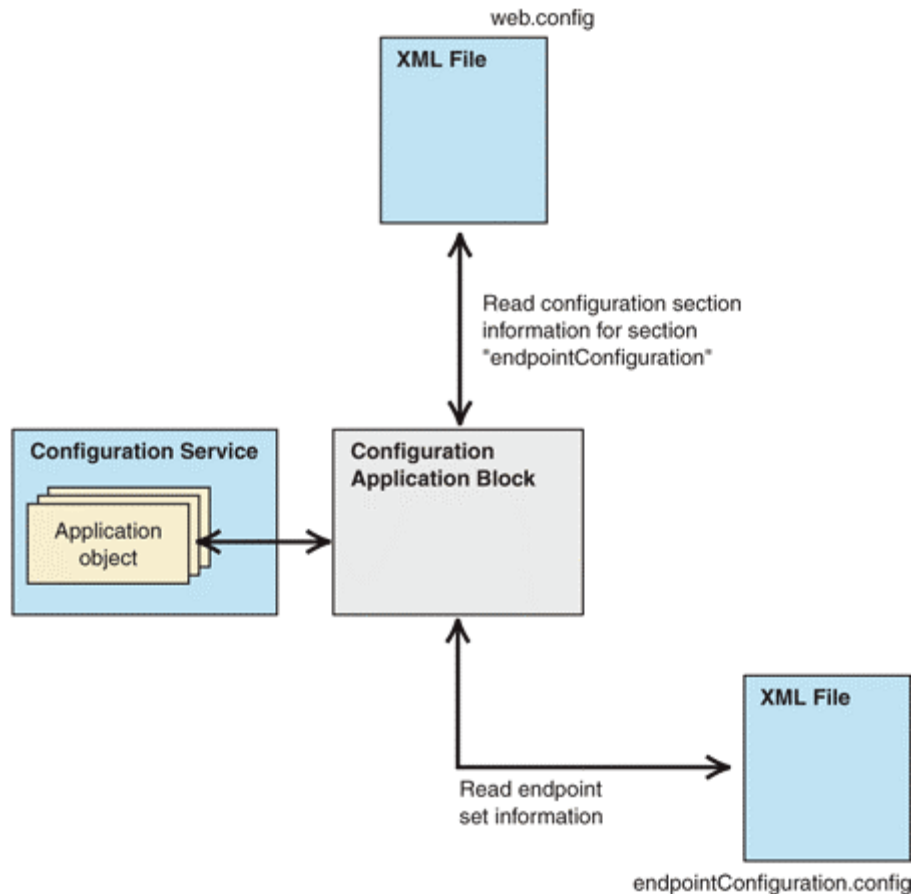


Figure A.3

How the Configuration Service uses the Configuration Application Block

Exception Handling Application Block

The Exception Handling Blocks allows you to define policies that govern how exceptions are handled. By specifying the same policy name, exceptions processed by the Exception Handling Application Block are processed in a consistent manner. Because the actions of a policy are controlled through configuration settings, the application behavior in response to an exception can be changed without required application source code changes.

Web Client

The Web client Web pages use **try/catch** constructs around their operations. In the **catch** clause, control is passed to the method **PublishError** of the class **UIExceptionHandler**. This method calls the Exception Handling Application Block specifying that the policy named **Log Only Policy** to be used.

The **Log Only Policy** configuration settings dictate that all exceptions are delivered to a single exception handler, a **Logging Handler**. That handler sends the exception information to the Logging and Instrumentation Application Block. The configuration settings for the handler determine the format of the information and additional attributes used by the Logging and Instrumentation Application Block. It also contains the following additional information, which is used by the Logging and Instrumentation Application Block to determine if, and where, the exception information will be logged:

- Category: General
- Priority: 0
- Severity: Error

Figure A.4 illustrates the Web client usage of the Exception Handling Application Block.

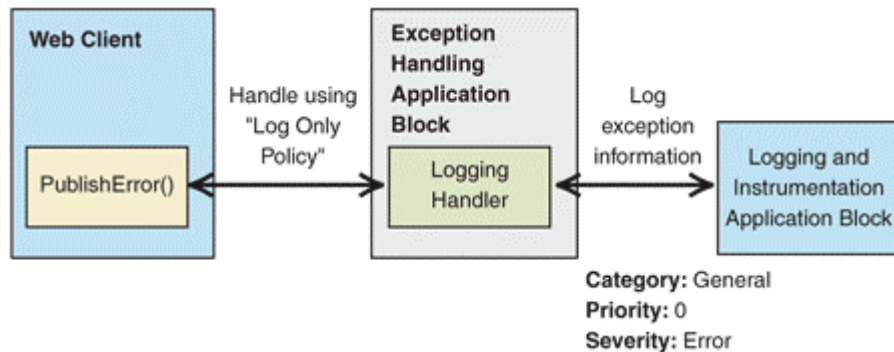


Figure A.4

How the Web client uses the Exception Handling Application Block

Retailer, Warehouse, and Manufacturer Data Access Component Layers

The Web services available for the retailer, warehouses, and manufactures each contain a data access layer component. The Exception Handling Application Block is used to process exceptions occurring during data access. The following code appears in the data access layer components, specifying that the exception handling policy **Data Policy** is to be used to process exceptions of type **SqlException**.

```
catch( SqlException sqlException )
{
    ExceptionPolicy.HandleException( sqlException, "Data Policy" );
    throw;
}
```

The configuration for **Data Policy** is identical in each of the Web services. According to this configuration, the following will occur when the policy is invoked for exceptions of type **SqlException**:

1. A logging handler sends the original exception information to the Logging and Instrumentation Application Block. The logged information contains a unique identifier (“handling instance identifier”) for the execution of the policy.
2. The exception is replaced with a new exception of type **System.ApplicationException**. The new exception message contains the handling instance identifier, providing a means to correlate the new exception with the original exception.

Manufacturer Service

The Manufacturer Service uses the Exception Handling Block to process exceptions that occur during processing. The exceptions are processed according to the policy **Log Only Policy**; this policy is configured with a single logging handler.

Custom Policy Assertion

The custom policy assertion **FaultLoggingAssertion** processes exceptions using the policy **Fault Logging Policy**. This policy is configured with a single logging handler.

SOAP Server Exceptions

The Sample Application Web services generate SOAP exceptions for the exceptions that occur during processing. The application uses the class **SoapUtility** to create SOAP server exceptions. This class calls the Exception Handling Application Block to process the original exception according to the policy **Log Only Policy**.

Logging and Instrumentation Application Block

The Logging and Instrumentation Application Block can filter out log messages and prevent them from being logged. Log messages are filtered by their log categories and their priorities. Through configuration settings, the Logging and Instrumentation Application Block allows you to filter out all messages for specified categories or log only messages for specified categories. In addition, you can set a minimum priority for log messages. Any log message with a value lower than the minimum priority will not be logged. The Logging and Instrumentation Application Block can process log messages either synchronously or asynchronously. The default Web client configuration for the Logging and Instrumentation Application Block specifies that messages will be logged synchronously.

Enabling Logging and Tracing

The Sample Application uses a SOAP extension to intercept and log all SOAP messages to a SQL database. The Utilities project in the sample solution uses the Data Access Application Block to read and write the SOAP message data. This project specifies the database instance named **TraceLog** when creating the Data Access Application Block database object. The name determines the connection string and database system object information that is used when connecting to the underlying database system.

If you are going to use the logging and tracing functionality of the Logging and Instrumentation Application Block, you need to ensure that they are enabled. Figure A.5 shows logging and tracing enabled for the Retailer Service.

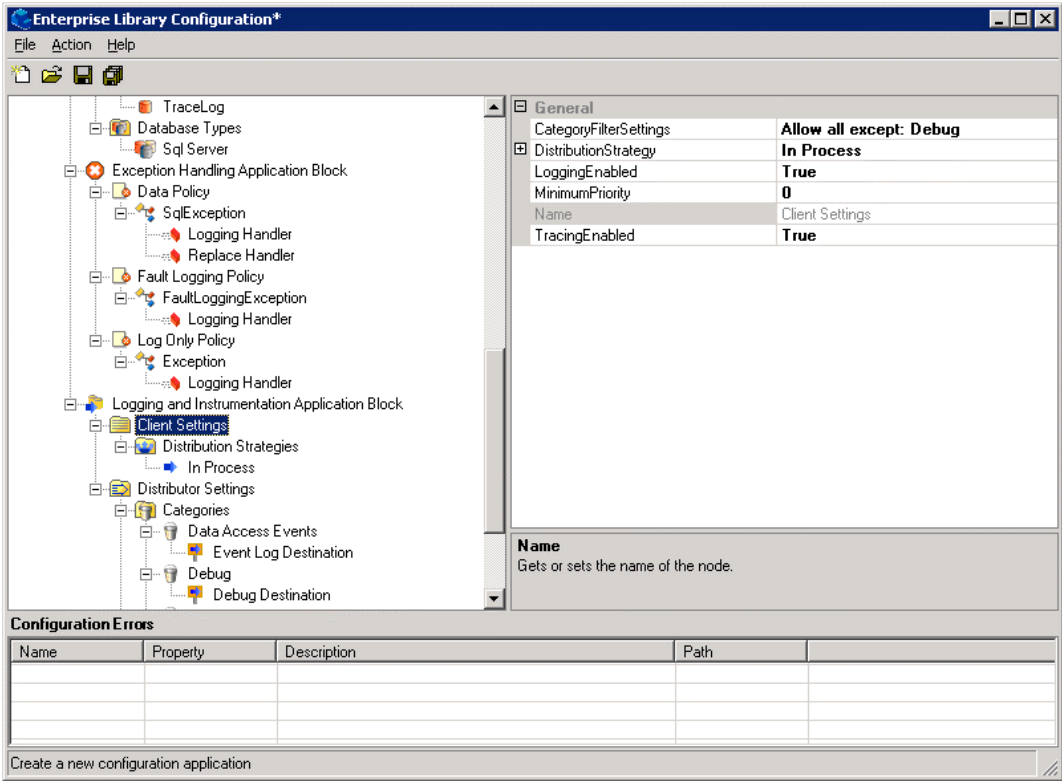


Figure A.5
Configuring logging and tracing for a Web service

Web Client and Services

The Logging and Instrumentation Application Block is not used directly by the Sample Application. Instead, the Sample Application uses the Exception Handling Application Block logging handler, which delivers exception information to the Logging and Instrumentation Application Block for processing.

The default Web client and Web service configurations for the Logging and Instrumentation Application Block does not filter out any log messages. This means that all exception messages delivered from the Exception Handling Application Block will be logged.

The category of the log message determines where the log message will be delivered for the actual logging operation. The categories **General** and **Data Access Events** both specify that log messages are delivered to the event log. This means that all exception messages delivered to the Logging and Instrumentation Application will appear in the event log.

Data Access Application Block

The Data Access Application Block simplifies development tasks that implement common data access functionality. Applications can use the application block in a variety of situations, such as reading data for display, passing data through application layers, and submitting changed data back to the database system. The application block includes support for both stored procedures and in-line SQL, and common housekeeping tasks, such as managing connections and creating and caching parameters, are encapsulated in the application block's methods. In other words, the Data Access Application Block provides access to the most often used features of ADO.NET while hiding its complexity.

Web Client

The Web client allows users to view the trace information stored in the SQL database. This means that the Web client configuration must include the configuration information for the **TraceLog** database instance. The Web client configuration information specifies that the **TraceLog** database instance refers to the SQL database named **TraceLog** on the local system.

Retailer, Warehouse, and Manufacturer Services

The Retailer, Warehouse, and Manufacturer services each contain a data access layer component. These components wrap the calls to the database, shielding the business components from database interaction. These data access components use the Data Access Application Block to read and write SQL information. The Retailer, Warehouse, and Manufacturer services are configured to use different SQL databases. The databases are defined by the Data Access Application Block configuration settings for each service.

To allow the SOAP extension to write the trace information to a common database target, each of them also includes the identical configuration information for the **TraceLog** database instance.

Appendix B

Sample Application Retailer Service Messages

This appendix provides a high-level examination of the source code in Microsoft's implementation of the Sample Application, concentrating on how the Web service and client source implements WSDL and XML Schema files that make up the Supply Chain Management (SCM) architecture's interface. This appendix focuses on the Retailer Web service as an example.

Note: This appendix is an extract from the original WS-I Basic Profile 1.0 application documentation. There are instances where references to code and namespaces might be incorrect; however, the content is only intended for informational purposes and focuses on how the services schema and messages were defined. This appendix will be updated in the final version.

The Configurator Web service uses Endpoints.xml to locate the Retailer instance. The following entry assumes the Retailer instance has been installed on the same computer as the Configurator service.

```
<EndPoint role="retailer" description="Retailer on localhost"
url="http://localhost/wsiscmv1/Retailer/Retailer.asmx" />
Retailer.asmx, an ASP.NET Web service deployment file, consists of a single
WebService directive, as follows:
<%@ WebService Language="c#" Codebehind="Retailer.asmx.cs"
Class="Wsi.SampleApps.Scm.Retailer.RetailerSoapBinding" %>
```

The class **RetailerSoapBinding**, located in Retailer.asmx.cs, extends **System.Web.Services.WebService**. The **getCatalog** and **submitOrder** operations defined in Retailer.wsdl are represented by corresponding public methods in **RetailerSoapBinding**. The .NET Framework attribute, **WebMethodAttribute**, is applied to each method.

RetailerSoapBinding creates instances of various other classes in the **Wsi.SampleApps.Scm.Retailer** namespace. These classes, also located in **Retailer.asmx.cs**, include **CustomerDetailsType**, **PartsOrderItem**, **Catalog**, and so on.

Another file, **RetailerProxy.cs**, contains client proxy classes in a different namespace, **Wsi.SampleApps.Scm.Retailer.Proxies.Retailer**. **RetailerProxy.cs** is auto-generated by Wsdl.exe. A **RetailerProxy** class and other client analogs are then referenced by the Shopping Cart **codebehind** page, **ShoppingCart.aspx.cs**.

In these respects, Retailer is a standard, simplistic ASP.NET Web service. However, when generating WSDL and SOAP, even a standard, simplistic Web service relies on the default behavior of the Web services framework or toolkit it was built with. To make sure that Retailer complies with the WSDL (and XML Schema definitions), more information needs to be provided.

With ASP.NET, additional information is provided mainly through the use of additional .NET Framework attributes. Developers have at their disposal a variety of attributes they can use to precisely control the XML generated by a Web service. These attributes can be defined manually or by using Wsdl.exe and other tools.

Defining XML Namespaces

Retailer.wsdl defines its own namespace URI with its location: *http://www.ws-i.org/SampleApplications/SupplyChainManagement/2002-08/Retailer.wsdl*.

It also imports three XML Schema files together with their XML namespaces and locations in a single `wsdl:types/xs:schema/` section (where `xs` is a shortcut for the XML Schema namespace), as shown in the following example.

```
<wsdl:types>
  <xs:schema>
    <xs:import namespace="http://www.ws-
i.org/SampleApplications/SupplyChainManagement/2002-08/RetailCatalog.xsd"
schemaLocation="RetailCatalog.xsd" />
    <xs:import namespace="http://www.ws-
i.org/SampleApplications/SupplyChainManagement/2002-08/RetailOrder.xsd"
schemaLocation="RetailOrder.xsd" />
    <xs:import namespace="http://www.ws-
i.org/SampleApplications/SupplyChainManagement/2002-08/Configuration.xsd"
schemaLocation="Configuration.xsd" />
  </xs:schema>
</wsdl:types>
```

The Configuration XML Schema file is used by various Web services after being set by the Configurator. The Configuration SOAP header is used to keep track of a user session ID and Web service endpoints.

The Implementation

Classes commonly used by various Web applications in the Sample Application are consolidated in the **Wsi.SampleApps.Scm.Common** namespace and compiled into the **Common.dll** assembly. This assembly is placed in the bin folder of all 10 Web applications.

One of these classes, **Ns**, consists entirely of string constants defining XML namespaces. The Retailer Web service uses four of these constants, as shown in the following code example.

```
public const string RET_NS="http://www.ws-  
i.org/SampleApplications/SupplyChainManagement/2002-08/Retailer.wsdl";  
public const string RETO_TYPE_NS="http://www.ws-  
i.org/SampleApplications/SupplyChainManagement/2002-08/RetailOrder.xsd";  
public const string RETC_TYPE_NS="http://www.ws-  
i.org/SampleApplications/SupplyChainManagement/2002-08/RetailCatalog.xsd";  
public const string CF_TYPE_NS="http://www.ws-  
i.org/SampleApplications/SupplyChainManagement/2002-08/Configuration.xsd";
```

Using the Retailer Namespace

The XML namespace of Retailer.wsdl is literally its target namespace, as shown in the following XML attribute of the root element, **<wsdl:definitions>**.

```
targetNamespace="http://www.ws-  
i.org/SampleApplications/SupplyChainManagement/2002-08/Retailer.wsdl"
```

Elements defined in Retailer.wsdl will belong to that XML namespace. When they are referenced in another file, they can be qualified with a shortcut to the namespace URI of Retailer.wsdl.

The Implementation

Multiple .NET Framework attributes use the **Retailer** target namespace directly or indirectly.

WebServiceAttribute is applied to the **WebService** subclass. Use the **Namespace** property of **WebServiceAttribute** to set the Web service namespace. Otherwise, the default is "http://tempuri.org/." Certain properties of the method-level **SoapDocumentMethod** attribute (also in this list) use this namespace.

Note: Do not confuse **WebServiceAttribute** with the **WebService** directive in an .asmx file or with the **WebService** class.

WebServiceBinding is also applied to a **WebService** subclass to establish a namespace. **WebServiceBinding** has **Name** and **Namespace** properties, allowing a namespace to be mapped to a key. Then the method-level **SoapDocumentMethod** can specify a binding by name (**WebServiceBinding.Name**) so that it can use a namespace (**WebServiceBinding.Namespace**). If **SoapDocumentMethod** does not specify the binding, the binding does not get used.

SoapDocumentMethod is applied to a Web method. Use this attribute's **Binding** property to specify the name of a **WebServiceBinding**. The default value for **Binding** is "**WebServiceNameSoap**," which maps to "http://tempuri.org/."

SoapDocumentMethod also has a **RequestNamespace** property. The default value is the namespace set with the **WebService** attribute or, if that is not set, "http://tempuri.org/." If you set **WebService.Namespace**, you do not have to set **SoapDocumentMethod.RequestNamespace**.

XmlElement is applied to an object/instance. This attribute refers to XML in general, not specifically to SOAP/WSDL.

In the Sample Application implementation, every return value and parameter of a Web method has an **XmlElement** attribute applied to it. Typically, that attribute is applied in a form such as **[XmlElementAttribute("getCatalog", Namespace=Ns.RET_NS)]**, where "getCatalog" is the element name.

With a document/literal Web service, the **XmlElement** attribute provides a readily apparent means of specifying the single child element of the **<Body>** in a SOAP request or response. (The WS-I Basic Profile Version 1.0 specification recommends that a document/literal message have a single part.)

XmlType is applied to a class/type. For each type that gets translated into an XML Schema **<complexType>** for sending over the network, the **XmlType** attribute is applied and a namespace is specified as a property. The Sample Application implementation normally uses the **XmlType** attribute to specify an XML Schema namespace, not a WSDL namespace. However, **XmlType** is additionally used once in **Retailer.asmx.cs** to specify the **Retailer.wSDL** namespace. For example, see the following empty class.

```
[XmlTypeAttribute(Namespace=Ns.RET_NS)]
public class getCatalog
{
}
```

This class takes the place of an otherwise empty SOAP request body for a parameter-less get method.

The `Retailer.wsdl` namespace is specified at the Web service level as follows:

```
...
[WebService(Namespace=Ns.RET_NS)]
[WebServiceBinding(Name="RetailerSoapBinding", Namespace=Ns.RET_NS)]
public class RetailerSoapBinding : System.Web.Services.WebService
```

Then at the method level, the **RetailerSoapBinding** binding is cited by key. At the same time, the namespace is applied to individual elements, as shown in the following code example.

```
[WebMethodAttribute()]
[SoapDocumentMethod("",
    RequestElementName="getCatalog",
    RequestNamespace=Ns.RET_NS,
    Binding="RetailerSoapBinding",
    ...
    [return: XmlElementAttribute("getCatalogResponse", Namespace=Ns.RET_NS)]
    public Catalog GetCatalog([XmlElementAttribute("getCatalog", Namespace=Ns.RET_NS)]
        getCatalog getCatIn)
```

Using the RetailOrder XML Schema

`Retailer.wsdl` imports `RetailOrder.xsd` for use by the **submitOrder** operation.

In `Retailer.wsdl`, the message **SubmitOrderRequest** consists of three parts, one of which is a Configuration SOAP header. (Retailer is defined as an RPC-style service, so it can have more than one body part.) The two non-header parts are as follows:

- **PartsOrderType PartsOrder**. According to `RetailOrder.xsd`, a **PartsOrderType** consists of an array of **PartsOrderItem** types, with each child element named `item`. A **PartsOrderItem** consists of three simple types.
- **CustomerDetailsType CustomerDetails**. In `RetailOrder.xsd`, a **CustomerDetailsType** consists of numerous simple types.

The message **SubmitOrderResponse** has one part, as follows:

- **PartsOrderResponseType return**. According to `RetailOrder.xsd`, a **PartsOrderResponseType** consists of an array of **PartsOrderResponseItem** types. A **PartsOrderResponseItem** consists of four simple types.

The Implementation

The **SubmitOrder** method signature (leaving out some attributes) is as follows.

```
[return: XmlElementAttribute("submitOrderResponse", Namespace=Ns.RET_NS)]
public SubmitOrderResponse
submitOrder([XmlElementAttribute("submitOrder", Namespace=Ns.RET_NS)]
            SubmitOrderRequestType submitOrderIn)
```

The class **SubmitOrderRequestType** has the following two public members:

- **PartsOrderItem[] PartsOrder**. The array is assigned a **System.Xml.Serialization.XmlArrayItem** attribute specifying “**Item**” as the name of each array entry. Its simple members are public.
- **CustomerDetailsType CustomerDetails**. This type also consists of simple, public members.

The following attribute has been applied to all three of these classes’ declarations.

```
[XmlTypeAttribute(Namespace=Ns.RETO_TYPE_NS)]
```

The **submitOrder** method declaration uses the **Retailer.wsdl** namespace for the name of the **submitOrderIn** element. Meanwhile, **SubmitOrderRequestType** uses the **RetailOrder.xsd** namespace for the element’s type. This will produce SOAP XML similar to the following.

```
...
<soap:Body>
  <retail:submitOrder xsi:type="order:SubmitOrderRequestType"
    xmlns:retail="...Retailer.wsdl"
    xmlns:order="...RetailOrder.xsd">
  ...
```

Through the use of attributes, both member variables of **SubmitOrderRequest** are given an unqualified form, where no XML namespace is specified. Instead, the XML elements inherit their parent’s namespace. For the **PartsOrderItem** array, the **XmlArray** attribute is used; for the **CustomerDetailsType** instance, the **XmlElement** attribute is used. For both attributes, the **Form** property is used.

By default, the member **PartsOrderItem[] PartsOrder** is serialized as a nested array, with a single parent element. This corresponds to the **<complexType>** by which “**PartsOrderType**” is defined in **RetailOrder.xsd**.

The response’s SOAP body contains an element, **<submitOrderResponse>**, represented by the C# class **submitOrderResponse**. Unlike with the request, both the name and type of the response belong to the **Retailer.wsdl** namespace. The **RetailOrder.xsd** namespace comes into effect only with the single member of **submitOrderResponse**—**PartsOrderResponseItem[] @return**.

This approach is actually more consistent with the WSDL and XML Schema files from the Sample Application SCM architecture specification. However, either way, you are making a somewhat fussy compromise when the contents of the SOAP body negotiate between two namespaces. While implementing the classes, you have to ask, “Where does the one namespace end and the other namespace begin?”

Contrast this confusion with the cleanness of the document-style Manufacturer Web service, which uses a single XML namespace for the entire contents of **<soap:Body>**. The following is the relevant passage from Manufacturer.wsdl, where “po” is short for ManufacturerPO.xsd.

```
<wsdl:message name="POSubmit">
    ...
    <wsdl:part name="PurchaseOrder" element="po:PurchaseOrder" />
    ...
</wsdl:message>
<wsdl:message name="ackPO">
    ...
    <wsdl:part name="Response" element="po:ackPO" />
</wsdl:message>
```

Mapping Simple XML Types

Any **<complexType>** contains and is ultimately reducible to one or more **<simpleType>** entities. A **<simpleType>** is either predefined in the XML Schema or custom-defined by restriction.

RetailOrder.xsd uses various simple types, both predefined and custom.

The Implementation

The online documentation for the **XmlElementAttribute.DataType** property contains a table of automatic mappings between XML Schema data types and .NET Framework data types. Where the automatic mapping does not suffice, apply an **XmlElement** attribute with a **DataType** property to a member.

For example, the **CatalogItem** class contains the following declaration.

```
[XmlElementAttribute(DataType="integer")]
public string productNumber;
```

Specifying Literal Usage

Drill down from the **<wsdl:binding>** element in every .wsdl file for the Sample Application, and every SOAP binding element will have a **use="literal"** XML attribute. (The default value for use must also be “literal”, according to the Basic Profile.) This attribute indicates literal XML Schema formatting of XML, as opposed to SOAP encoding.

The Implementation

Every Web method is given a **SoapDocumentMethod** attribute with a **Use** property set to **SoapBindingUse.Literal**. **SoapBindingUse** is an enumeration in the .NET Framework namespace **System.Web.Services.Description**.

Literal is also the default value for the **Use** property. Alternatively, **use="literal"** can be specified at the Web service level through the **SoapDocumentService** attribute, which also has a **Use** property.

Specifying Formatting Style

The Basic Profile Version 1.0 specification allows two formatting styles: document and RPC. The Retailer Web service specifies RPC style. In Retailer.wsdl, the first child of **<wsdl:binding>** is shown in the following code example.

```
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
```

Likewise, every **wsdl:message/wsdl:part/** has a **type** attribute instead of an **element** attribute. The intention is for the child and grandchildren of the **<soap:body>** to implicitly follow the RPC pattern.

The Implementation

Simply through the use of built-in attributes, the .NET Framework does not allow the combination of RPC style and literal usage — the combination specified by the Retailer description. The .NET Framework does provide **SoapRpcMethod** and **SoapRpcService** attributes as alternatives to **SoapDocumentMethod** and **SoapDocumentService**. However, these RPC-style attributes do not have a **Use** property. The use is assumed to be encoded.

There is also the option of creating a do-it-yourself RPC/literal solution. This Retailer implementation tries to approximate RPC/literal while still explicitly declaring document/literal. However, Microsoft discourages RPC/literal.

Specifying Parameter Style

One further action needs to be taken to avoid RPC-specific behavior. The .NET Framework offers a formatting construct referred to as wrapping, whereby message parts get bumped from being children of the **<soap:Body>** to being grandchildren. In between, a single element is always added. For a SOAP request, this element is named after the Web method; for a SOAP response, its name is **WebMethodResponse**.

The **SoapDocumentMethod** and **SoapDocumentService** attributes have a **ParameterStyle** property, whose possible values come from the **SoapParameterStyle** enumeration in the **System.Web.Services.Protocols** namespace. Primary values are **Wrapped** and **Bare**. The latter leaves the message parts as they are.

Unfortunately, as a vestige of SOAP's RPC-centric roots, the default value is **Wrapped**. Therefore, you will find that every Web method in the Sample Application implementation has a **SoapDocumentMethod** attribute with **ParameterStyle=SoapParameterStyle.Bare**.

(Like the **Use** property, the **ParameterStyle** property is not available for the **SoapRpcMethod** and **SoapRpcStyle** attributes. The assumed parameter style is **wrapped**.)

Contributors

The following individuals made a substantial contribution to the developing, writing, testing, and reviewing of this content.

Program Management

Jason Hogg, Microsoft Corporation

Development

Hernan de Lahitte, Lagash Systems SA

Diego Gonzalez, Lagash Systems SA

Pablo Cibraro, Lagash Systems SA

Test

Pete Coupland, VMC Consulting Corporation

Carlos Farre, Microsoft Corporation

Ken Perilman, Microsoft Corporation

Mrinal Bhao, Infosys Technologies Ltd

Sajjad Imran, Infosys Technologies Ltd

Sidambara Raja Krishnaraj, Infosys Technologies Ltd

GanapathiRam Natarajan, Infosys Technologies Ltd

Sachin Wagh, Infosys Technologies Ltd

Jude Yuvaraj, Infosys Technologies Ltd

Documentation

Paul Slater, Wadeware LLC

Tim Osborn, Ascentium Corporation

Tina Burden McGrayne, Linda Werner & Associates Inc

Claudette Siroky, CI Design Studio

Nelly Delgado, Microsoft Corporation

Sanjeev Garg, Satyam Computer Services

Review

Edward Jezierski, Microsoft Corporation
Peter Provost, Microsoft Corporation
Scott Densmore, Microsoft Corporation
Tom Hollander, Microsoft Corporation
Mark Fussell, Microsoft Corporation
Tomasz Janczuk, Microsoft Corporation
Bill Shihara, Microsoft Corporation
Andy Gordon, Microsoft Corporation
Karthik Bhargavan, Microsoft Corporation
Cédric Fournet, Microsoft Corporation
Guido Hinderberger, DaimlerChrysler TSS
Brian LeBlanc, DaimlerChrysler TSS
Aaron Skonnrd, Pluralsight LLC
Keith Brown, Pluralsight LLC
Martin Granell, Readify
Edward Bakker, LogicaCMG

patterns & practices

proven practices for predictable results

About Microsoft *patterns & practices*

Microsoft *patterns & practices* guides contain specific recommendations illustrating how to design, build, deploy, and operate architecturally sound solutions to challenging business and technical scenarios. They offer deep technical guidance based on real-world experience that goes far beyond white papers to help enterprise IT professionals, information workers, and developers quickly deliver sound solutions.

IT Professionals, information workers, and developers can choose from four types of *patterns & practices*:

- **Patterns**—Patterns are a consistent way of documenting solutions to commonly occurring problems. Patterns are available that address specific architecture, design, and implementation problems. Each pattern also has an associated GotDotNet Community.
- **Reference Architectures**—Reference Architectures are IT system-level architectures that address the business requirements, LifeCycle requirements, and technical constraints for commonly occurring scenarios. Reference Architectures focus on planning the architecture of IT systems.
- **Reference Building Blocks and IT Services**—References Building Blocks and IT Services are re-usable sub-system designs that address common technical challenges across a wide range of scenarios. Many include tested reference implementations to accelerate development. Reference Building Blocks and IT Services focus on the design and implementation of sub-systems.
- **Lifecycle Practices**—Lifecycle Practices provide guidance for tasks outside the scope of architecture and design such as deployment and operations in a production environment.

Patterns & practices guides are reviewed and approved by Microsoft engineering teams, consultants, Product Support Services, and by partners and customers. *Patterns & practices* guides are:

- **Proven**—They are based on field experience.
- **Authoritative**—They offer the best advice available.
- **Accurate**—They are technically validated and tested.
- **Actionable**—They provide the steps to success.
- **Relevant**—They address real-world problems based on customer scenarios.

Patterns & practices guides are designed to help IT professionals, information workers, and developers:

Reduce project cost

- Exploit the Microsoft engineering efforts to save time and money on your projects.
- Follow the Microsoft recommendations to lower your project risk and achieve predictable outcomes.

Increase confidence in solutions

- Build your solutions on proven Microsoft recommendations so you can have total confidence in your results.
- Rely on thoroughly tested and supported guidance, but production quality recommendations and code, not just samples.

Deliver strategic IT advantage






- Solve your problems today and take advantage of future Microsoft technologies with practical advice.

patterns & practices: Current Titles

October 2003


Title	Link to Online Version	Book
Patterns		
Enterprise Solution Patterns using Microsoft .NET	http://msdn.microsoft.com/practices/type/Patterns/Enterprise/default.asp	
Microsoft Data Patterns	http://msdn.microsoft.com/practices/type/Patterns/Data/default.asp	
Reference Architectures		
Application Architecture for .NET: Designing Applications and Services	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/distapp.asp	
Enterprise Notification Reference Architecture for Exchange 2000 Server	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnentdevgen/html/enraelp.asp	
Improving Web Application Security: Threats and Countermeasures	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/ThreatCounter.asp	
Microsoft Accelerator for Six Sigma	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/mso/sixsigma/default.asp	
Microsoft Active Directory Branch Office Guide: Volume 1: Planning	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/ad/windows2000/deploy/adguide/default.asp	
Microsoft Active Directory Branch Office Series Volume 2: Deployment and Operations	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/ad/windows2000/deploy/adguide/default.asp	
Microsoft Content Integration Pack for Content Management Server 2001 and SharePoint Portal Server 2001	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncip/html/cip.asp	
Microsoft Exchange 2000 Server Hosting Series Volume 1: Planning	Online Version not available	
Microsoft Exchange 2000 Server Hosting Series Volume 2: Deployment	Online Version not available	

To learn more about *patterns & practices* visit: <http://msdn.microsoft.com/practices>
 To purchase *patterns & practices* guides visit: <http://shop.microsoft.com/practices>

Title	Link to Online Version	Book
Microsoft Exchange 2000 Server Upgrade Series Volume 1: Planning	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/guide/default.asp	
Microsoft Exchange 2000 Server Upgrade Series Volume 2: Deployment	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/guide/default.asp	
Microsoft Solution for Intranets	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/mso/msi/Default.asp	
Microsoft Solution for Securing Wireless LANs	http://www.microsoft.com/downloads/details.aspx?FamilyId=CDB639B3-010B-47E7-B234-A27CDA291DAD&displaylang=en	
Microsoft Systems Architecture—Enterprise Data Center	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/edc/Default.asp	
Microsoft Systems Architecture—Internet Data Center	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/idc/default.asp	
The Enterprise Project Management Solution	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/mso/epm/default.asp	
UNIX Application Migration Guide	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnucmg/html/ucmglp.asp	
Reference Building Blocks and IT Services		
.NET Data Access Architecture Guide	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/daag.asp	
Application Updater Application Block	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/updater.asp	
Asynchronous Invocation Application Block	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/paiblock.asp	
Authentication in ASP.NET: .NET Security Guidance	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/authaspdotnet.asp	
Building Interoperable Web Services: WS-I Basic Profile 1.0	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsvcenter/html/wsi-bp_msdn_landingpage.asp	
Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/secnetlpMSDN.asp	

To learn more about *patterns & practices* visit: <http://msdn.microsoft.com/practices>
To purchase *patterns & practices* guides visit: <http://shop.microsoft.com/practices>

Title	Link to Online Version	Book
Caching Application Block	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/Cachingblock.asp	
Caching Architecture Guide for .Net Framework Applications	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/CachingArch.asp?frame=true	
Configuration Management Application Block	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/cmab.asp	
Data Access Application Block for .NET	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/daab-rm.asp	
Designing Application-Managed Authorization	http://msdn.microsoft.com/library/?url=/library/en-us/dnbda/html/damaz.asp	
Designing Data Tier Components and Passing Data Through Tiers	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/BOAGag.asp	
Exception Management Application Block for .NET	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/emab-rm.asp	
Exception Management Architecture Guide	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/exceptdotnet.asp	
Microsoft .NET/COM Migration and Interoperability	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/cominterop.asp	
Microsoft Windows Server 2003 Security Guide	http://www.microsoft.com/downloads/details.aspx?FamilyId=8A2643C1-0685-4D89-B655-521EA6C7B4DB&displaylang=en	
Monitoring in .NET Distributed Application Design	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/monitordotnet.asp	
New Application Installation using Systems Management Server	http://www.microsoft.com/business/reducecosts/efficiency/manageability/application.mspix	
Patch Management using Microsoft Systems Management Server - Operations Guide	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/msm/swdist/pmsms/pmsmsog.asp	
Patch Management Using Microsoft Software Update Services - Operations Guide	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/msm/swdist/pmsus/pmsusog.asp	
Service Aggregation Application Block	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/serviceagg.asp	
Service Monitoring and Control using Microsoft Operations Manager	http://www.microsoft.com/business/reducecosts/efficiency/manageability/monitoring.mspix	

Title	Link to Online Version	Book
User Interface Process Application Block	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/uip.asp	
Web Service Façade for Legacy Applications	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/wsfacadelegacyapp.asp	
Lifecycle Practices		
Backup and Restore for Internet Data Center	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/ittasks/maintain/backuprest/Default.asp	
Deploying .NET Applications: Lifecycle Guide	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/DALGRoadmap.asp	
Microsoft Exchange 2000 Server Operations Guide	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/exchange/exchange2000/maintain/operate/opsguide/default.asp	
Microsoft SQL Server 2000 High Availability Series: Volume 1: Planning	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/deploy/confeat/sqlha/SQLHALP.asp	
Microsoft SQL Server 2000 High Availability Series: Volume 2: Deployment	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/deploy/confeat/sqlha/SQLHALP.asp	
Microsoft SQL Server 2000 Operations Guide	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sql/maintain/operate/opsguide/default.asp	
Operating .NET-Based Applications	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/net/maintain/opnetapp/default.asp	
Production Debugging for .NET-Connected Applications	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/DBGrm.asp	
Security Operations for Microsoft Windows 2000 Server	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/prodtech/win2000/secwin2k/default.asp	
Security Operations Guide for Exchange 2000 Server	http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/prodtech/mailexch/opsguide/default.asp	
Team Development with Visual Studio .NET and Visual SourceSafe	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/tdlg_rm.asp	



This title is available as a Book

To learn more about *patterns & practices* visit: <http://msdn.microsoft.com/practices>
To purchase *patterns & practices* guides visit: <http://shop.microsoft.com/practices>