# ISHOP TECHNICAL IMPLEMENTATION GUIDE

Version 3

## Revision History

| Revision | Revision Description | Revision Date |
|---|---|---|
| 3.0 | Initial Release | July 7, 2006 |
| 3.0.1 | No changes to the Technical Implementation Guide document. | $$ |

| | |
|---|---|
| Prepared by: | Dan Harrel, Prescient Technologies Inc. |
| Contributors: | Dave Callingham, Prescient Technologies Inc.; Michael Lucas, Prescient Technologies Inc.; Mike Penner, Prescient Technologies Inc.; Russ Bailey, Autovice Consulting |
| | UDP and UDDI sample code originally from Roman Zhukov, Garage Operator. |

Last revised: July 7, 2008

## Table of Contents

## Introduction

Welcome to Auto Care Association's iShop integrated shop standards! Shop Integration Standards enable computer based diagnostic and repair equipment and information servers to share data about the customer, vehicle, work order, service and repair history, as well as parts and labor information seamlessly throughout the service environment.

iShop Version 3 specification is based on XML Web services technology. There are two Web services in the specification:

1. iShopInformationServer
2. iShopSMSServer

The operations exposed by these Web services are described in the Functional Guide. Each of these operations has a request message and response message. These messages are described by way of XML Schema Definitions (XSDs). The Web service, operations and their use of messages are formally described using Web Services Definition Language (WSDL).

This document was written to assist companies wanting to write software that operates within an iShop compliant environment, whether such software consumes or provides iShopservices. It addresses technical considerations, and should be read by any party interested in creating an iShop compliant application. Functional specifications are documented in a separate document: "Auto Care Association iShop Functional Implementation Guide".

This document assumes the reader is familiar with software development in general, and Web services technology in specific.

This document also assumes familiarity with the functional aspects of iShop as described in the "Auto Care Association iShop Functional Implementation Guide".

## Document Contents

The iShop Technical Implementation Guide is organized as follows:

**Table 1 - Document Contents**

| **Documentation Section** | **Contents** |
|---|---|
| Technical Implementation Guidelines | Provides iShop technical implementation guidelines independent of a particular implementation technology and thus independent of the code samples. |
| iShop Delivered Components | Describes the files deployed with iShop.  Files are deployed by running the iShop installer program. |
| iShopBridge | Describes the iShop Version 2 to iShop Version 3 bridge.  This is a migration tool that allows an iShop Version 2 client application to use an iShop Version 3 Information Server. |
| Code Samples Quick Reference | The code samples (next section below) demonstrates basic implementations of all types of iShop Version 3 applications.  This section lists topics demonstrated by the samples and points to where in the sample code the demonstration can be found. |
| Code Samples | Documentation for working code that demonstrates basic implementations of all types of iShop Version 3 applications.  iShop technical code is isolated into a framework library that is used by all sample applications, permitting them to focus to the maximum extent possible on business functionality. |
| XML | Documents the XML files are part of iShop – e.g. WSDL describing the iShop web services, schemas defining iShop messages, and sample iShop XML messages. |

## Related Documents

In addition to this document, readers may wish to consult the following:

- Auto Care Association iShop Function Implementation Guide. This is the specification document for the iShop Version 3 standard.
- OASIS defines the UDDI standard used to find iShop Information Servers. OASIS is found on the web at http://www.oasis-open.org. UDDI information from OASIS is found at http://www.uddi.org.
- The Auto Care Association iShop web site www.ishopportal.org should be checked for up-to-date and newly released documentation.

# Technical Implementation Guidelines

This documentation section provides iShop technical implementation guidelines independent of a particular implementation technology and thus independent of the code samples. It will, however, provide references into the code samples so that a technology-specific implementation can be reviewed. The topics covered in this section include:

- Error handling, and
- Finding an iSHOP server

## Error Handling

Errors that occur within iShop server applications, or are detected by iShop server applications, should not be signaled to client applications using SOAP faults. Client applications should catch SOAP faults, however these should only occur as a result of unexpected system errors. Instead, iShop server applications signal error conditions to the client by returning a standard response message for the operation with the error condition represented in the standard response element.

Please refer to the iShop Version 3 Functional Implementation Guide document for more information on the standard response element.

See documentation section *Error Handling* for references where error handling is implemented in the code samples.

**2.2 Finding an iShop Server**

An iShop client application should not have to be configured with the URLs of the iShop web services that it will use.  Doing so creates administrative overhead should the set of available web services or their locations change.  Instead, iShop specifies two technologies for finding servers:

1. UDDI technology for finding iShop servers outside the local network.  This generally applies to finding iShop Information Servers.
2. UDP technology for finding iShop servers inside the local network.  This generally applies to finding iShop Shop Management Servers.

These two options are described in the documentation sub-sections below.

**Finding an iShop Information Server using UDDI**

UDDI (Universal Description, Discovery, and Integration) is a standard for registries listing web services hosted on the Internet.  The standard is managed by OASIS, a non-profit industry consortium, found on the web at http://www.oasis-open.org.  UDDI information from OASIS is found at http://www.uddi.org.

AUTO CARE ASSOCIATION will host the UDDI server for the purpose of finding iShop information servers.  The following URLs are available:

http://www.uddi.ishopstandards.org/uddipublic/

Web based user interface for a human to search for an iShop server.

http://www.uddi.ishopstandards.org/uddipublic/inquire.asmx

Web Service for an application to search for an iShop Server.

See documentation section *Server Discovery* for an example of UDDI searching as implemented in the code samples.

**Finding an iShop Shop Management Server using UDP**

UDP (User Datagram Protocol) is an internet network protocol for sending short messages.  UDP is not reliable as is TCP in that messages are not guaranteed to be either received or received in the order sent.

iShop uses UDP messages only for the purposes of finding an iShop server on a local network.  The client sends a UDP message consisting of a byte array of the characters “iShopSms” to a UDP group IP address.  Note that all byte arrays of characters should use ASCII encoding.  Usual rules for IP addresses apply – i.e. use IP address “255.255.255.255” to broadcast over the entire network.

An iShop server that should be found using UDP must be represented by an application that listens for such a UDP message (e.g. byte array of characters “iShopSms”). We will call this application a UDP Discovery Server.

The UDP Discovery Server joins a multicast group to listen for UDP messages. The multicast group is identified by an IP address. This address should be one that is reached by the UDP message sent by clients looking for an iShop server. IP address “255.255.255.255” should work in all cases, however the server computer’s IP address should also suffice.

When the UDP Discovery Server receives a valid request message (byte array of characters “iShopSms”), it composes a response message starting with the same bytes, immediately followed by a byte array serialization identifying the server. The serialization must be done in a implementation-independent manner – i.e. Microsoft .NET or similar serialization should not be used. iShop requires the serialization be done as per the following example:

Assume the SMS server to be published is named “ACME Shop Management System” and is accessed at http://ourhost/Acme/iShop.asmx. Then the serialization string representing this iShop SMS server is:

8|iShopSms27|ACME Shop Management System30|http://ourhost/Acme/iShop.asmx

- “8|” indicates that the next 8 characters form the first token: “iShopSms”. This token indicates that the server identified is an iShop SMS server.

  Token value “iShopIs” is reserved to identify the server an iShop Information Server, however UDDI is presently used to find Information Servers, not UDP.

- “27|” indicates that the next 27 characters form the second token: “ACME Shop Management System”. This token represents the server name.

- “30|” indicates that the next 30 characters form the third token: “http://ourhost/Acme/iShop.asmx”. This token represents the web service URL.

Continuing the above example, the full UDP message sent by the UDP Discovery Server back to the client is:

iShopSms8|iShopSms27|ACME Shop Management System30|http://ourhost/Acme/iShop.asmx

The client application should therefore wait and listen for response UDP messages once it has sent its request UDP message. In theory, multiple response messages could arrive from multiple UDP Discovery Servers, although this is unlikely in practice. It is recommended that the client application wait a pre-set amount of time for responses.

See documentation section *Server Discovery* for references where this is implemented in the code samples.

**Potential Issues**

Potential issues may occur with the iShop UDP solution if CheckPoint VPN-1 Secure Client is installed on the computer using UDP to search for an iShop UDP Discovery Server, or on the iShop UDP Discovery Server computer itself. Namely, "Disable Security Profile" will have to be configured for the application. There is also evidence that a UDP client and UDP Discovery Server cannot run simultaneously a computer with CheckPoint VPN installed. The VPN software may have problems with the loopback (127.0.0.1) interface. There is evidence that Cisco VPNs have similar problems.

It is possible that any computer with VPN software installed may have difficulties running UDP discovery.

Should this problem occur at a particular site, a knowledgeable IT resource should be consulted. One should be available if the site is sophisticated enough to run a VPN. In a worstcase scenario, manual entry of the server location may be necessary.

## iShop Delivered Components

Delivered components for the iShop Version 3.0 specification are obtained by running installation program, “iShop 3.0 - setup.EXE”. Note that the installation program name may change appropriately as maintenance releases are issued (e.g. “iShop 3.1 - setup.EXE”).

The iShop installation program is available to iShop participants through password protected sections of the iShop web site at www.ishopportal.org. Please refer to the iShop Version 3 Functional Implementation Guide document for more information on this site and membership.

The high level folder structure of installed components is as follows:

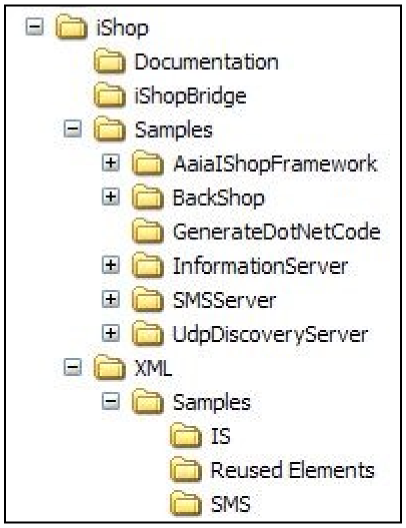


**Figure 1 - iShop Delivered Components**

Each of the above high level folders is summarized below, and fully described in the subsections that follow.

**Table 2 - Installed Components**

| Component | Summary |
|---|---|
| Documentation | Contains general iShop specification documents, including this document. |
| iShopBridge | Contains the bridge from iShop 2 to iShop 3. This bridge has also been referred to as the "Golden Gate" bridge.<br><br>In particular, the bridge is an iShop Version 2 Parts/Labor and Repair server implementation that accepts a COM iShop 2 request, translates the request to an iShop 3 XML request message, passes the request message an iShop 3 Information Server web service, receives an iShop 3 XML response message, translates the response message into an iShop 2 COM response, and returns the iShop 2 COM response to the calling iShop 2 client application. |
| Samples | Contains example iShop 3 implementations.  Implementation technology is Microsoft VB .NET 2005. |

| | |
|---|---|
| XML | Contains all XML delivery components of iShop 3 including schemas (XSD), WSDL and sample XML messages. |

## Documentation

The Documentation folder contains the two major documents that make up the iShop Version 3 specification:

**Table 3 - Documentation**

| Document | Description |
|---|---|
| Functional Implementation Guide | This document contains the iShop Version 3 functional specification. Its purpose is to educate about iShop as well as to serve as a reference for the standard. |
| Technical Implementation Guide | This is the document you are presently reading.<br><br>This document covers iShop Version 3 technical and implementation issues. Its purpose is to help implementers build an iShop Version 3 compliant solution. |

## iShopBridge

The iShopBridge folder contains the bridge from iShop 2 to iShop 3. This bridge has also been referred to as the "Golden Gate" bridge.

The iShopBridge is an iShop Version 2 Parts/Labor and Repair server implementation. As per the iShop Version 2 specification, it is a Microsoft COM object. Its behavior is as described in the document "iShop Specifications Version 2.0" dated May 20, 2004.

In particular, the iShopBridge accepts a COM iShop 2 request, translates the request to an iShop 3 XML request message, passes the request message an iShop 3 Information Server web service, receives an iShop 3 XML response message, translates the response message into an iShop 2 COM response, and returns the iShop 2 COM response to the calling iShop 2 client application.

iShopBridge is written in Microsoft Visual Studio .NET 2005 as a COM Interop program. Therefore the Microsoft .NET Framework version 2 must be installed on any computer that iShopBridge runs on.

**iShopBridge Distributed Files**

**File: iShopBridgeSetup.msi**

This file is a Microsoft Windows Installer Package that will install the iShopBridge on a Microsoft

Windows computer. By default, the iShopBridge will install into directory "c:\Program Files\iShop\iShopBridge".

The installer package registers the iShopBridge DLLs as required by COM. It also installs a configuration file that must be edited to identify the iShop Version 3 Information Server to bridge to – see section *Using the iShopBridge*, below.

**Using the iShopBridge**

iShopBridge will be used by an iShop Version 2 client application to access an iShop Version 3 Information Server. This client application will typically be a Shop Management System (SMS) application. It is an interim solution, as it is expected that Shop Management Systems that support iShop will be upgraded to support iShop Version 3 directly.

Incorporate iShopBridge into a situation as described above using the following steps.

1. Run the "iShopBridgeSetup.msi" file to install the iShopBridge on the computer running an iShop Version 2 client application.
2. Edit the installed "iShopBridge.config" file to include correct credentials and the location of the Version 3 InformationServer you want the iShopBridge instance to communicate with. Setting keys are as follows:

**Table 4 - iSHOP Bridge Settings**

| Setting Key | Description |
|---|---|
| InformationServerURL | URL of the iShop Version 3 Information Server that the iShopBridge should bridge to. |
| UserName | User name to pass to the iShop Version 3<br><br>Information Server in the Open operation. |

| | |
|---|---|
| Password | Password for setting key UserName. |

3. Update the registry on the iShop Version 2 application computer such that the application can find the iShopBridge. Registry settings are according to the iShop Version 2 standard. The following registry export file is an example:

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\AAIA\IShop]

[HKEY_LOCAL_MACHINE\SOFTWARE\AAIA\IShop\Clients]

[HKEY_LOCAL_MACHINE\SOFTWARE\AAIA\IShop\Clients\Default]

[HKEY_LOCAL_MACHINE\SOFTWARE\AAIA\IShop\Clients\Default\PartsLabor]

"01"="iShopBridge"

[HKEY_LOCAL_MACHINE\SOFTWARE\AAIA\IShop\Clients\Default\Repair]

"01"="iShopBridge"

[HKEY_LOCAL_MACHINE\SOFTWARE\AAIA\IShop\Servers]

[HKEY_LOCAL_MACHINE\SOFTWARE\AAIA\IShop\Servers\PartsLabor]

[HKEY_LOCAL_MACHINE\SOFTWARE\AAIA\IShop\Servers\PartsLabor\iShopBridge]

"ApplicationTitle"="iShopBridge PartsLabor Server"

"HostName"="localhost"

"ProgId"="AAIA.iShopBridge.PartsLaborServer"
[HKEY_LOCAL_MACHINE\SOFTWARE\AAIA\IShop\Servers\Repair]

[HKEY_LOCAL_MACHINE\SOFTWARE\AAIA\IShop\Servers\Repair\iShopBridge]

"ApplicationTitle"="iShopBridge Repair Server"

"HostName"="localhost"

"ProgId"="AAIA.iShopBridge.RepairServer"
```

4. The iShop Version 2 client application should now be able to use iShopBridge as an iShop Version 2 server.

## Code Samples Quick Reference

The next documentation section, Code Samples, describes the sample code delivered with iShop 3. This quick reference section lists topics demonstrated by the samples and points to where in the sample code the demonstration can be found.

Quick reference documentation talks about "framework code". The sample code is structured such that as much reusable iShop functionality is isolated in a framework DLL that can be used directly by an iShop implementer, extended, or used as an example. The framework is described in documentation section AaiaIShopFramework (page 16).

**Business Functionality**

**Table 5 - Code Sample Business Functionality**

| Topic | Documentation Reference |
|---|---|
| Creating an Information Server request message. | "ExampleISCalls.vb". |
| Invoking an Information Server. | "ExampleISCalls.vb". |
| Processing an Information Server request message and composing a response. | Web Operation Implementation Code Files. |
| Information Server Session Management | Session management code is encapsulated in "Session.vb".<br><br>Session is opened by web operation "Open.vb" and closed by web operation "Close.vb". All web operations that require a session validate one is open. All web operations are documented in section<br><br>Web Operation Implementation Code Files. |
| Creating a Shop Management Server request message. | "ExampleSMSCalls.vb". |
| Invoking a Shop Management Server. | "ExampleSMSCalls.vb". |

| | |
|---|---|
| Processing a Shop Management Server request message and composing a response. | Web Operation Implementation Code Files. |

**Technical Functionality to enable Business Functionality**

**Table 6 - Code Sample Technical Functionality**

| Topic | Documentation Reference |
|---|---|
| Client application invoking framework code that in turn calls an iSHOP web service. | “ExampleISCalls.vb” (page 27) and “ExampleSMSCalls.vb” (page 28). |
| Framework code that calls an iSHOP web service. | Framework code iShopISInterfaceClient.vb and “iShopSmsInterfaceClient.vb” invoke generated web service proxy code.<br><br>Generated web service proxy code are files “iShopISWebserviceClient.vb” and "iShopSmsWebserviceClient.vb”. |
| Web service code that receives client messages and invokes business functionality code. | “App_Code\iShopInformationServer.vb” and “App_Code\iShopSmsServer.vb”.<br><br>Business functionality code is described in documentation section *Web Operation Implementation Code Files* (page 36 for Information Server, page 39 for Shop Management Server). |

**Error Handling**

**Table 7 - Error Handling**

| Topic | Documentation Reference |
|---|---|

| | |
|---|---|
| Populating a standard response element | See file “iShopMessageResponse.vb” for framework functionality to create response elements and how this is used to construct framework issued errors.<br><br>See “Web Methods Code Files” for the sample<br><br>Information Server and Shop Management Server for examples of signaling “invalid XML” errors. |

**Server Discovery**

**Table 8 - Server Discovery**

| Topic | Documentation Reference |
|---|---|
| Searching for an Information Server using UDDI | UDDI searching functionality is implemented by framework code “iShopServerDiscoveryUddi.vb”. |
| | This framework UDDI searching functionality is used by “MainForm.vb”. |
| Searching for a Shop Management Server using UDP | UDP searching functionality is implemented by framework code “iShopServerDiscoveryUdp.vb”.<br><br>This framework UDP searching functionality is used by “MainForm.vb”.<br><br>UDP search requests must be responded-to by a UDP discovery server. This application is described in documentation section *UdpDiscoveryServer*. |

## Code Samples

The Samples folder contains working code that demonstrates basic implementations of all types of iShop Version 3 applications – i.e. an Information Server, Shop Management Server, and Back Shop equipment application.

The purpose of the sample code is to fully demonstrate working iShop Version 3 applications, focusing on the required technical plumbing. For example: creating iShop XML messages, interpreting iShop XML messages, finding an iShop web service, invoking an iShop web service, and error handling.

The purpose of the sample code is NOT to demonstrate the business processes of acting on a message to formulate a response. This is simple business batch processing, and it is assumed that all implementers are capable of writing such code. The sample code IS written to demonstrate how to make an easily interpreted message available to act on, and how to make an easily interpreted response message available to the recipient.

The sample code is structured such that as much reusable iShop functionality is isolated in a framework DLL that can be used directly by an iShop implementer, extended, or used as an example. The framework is contained in sub-folder AaiaIShopFrameowork.

Sub-folders of the Samples folder hold the various sample deliverables. These sub-folders are summarized below, and fully described in the sub-sections that follow. All sample code is written in Microsoft VB .NET 2005. Each code and related file is documented, however more detail is found in comments embedded in the files themselves.

**Table 9 - Sample Code Description**

| Application | Summary |
|---|---|
| AaiaIShopFramework | This folder contains a library of framework code that is used by all the other sample applications – both clients and servers. The other sample applications reference the DLL compiled from this code.<br><br>All effort has been made to encapsulate iShop plumbing into this library. This library can be used directly by an iShop implementer, and extended, or it can serve as an example. |
| BackShop | This folder contains a sample Back Shop equipment application.<br><br>It demonstrates calling both the iShop Information Server and Shop Management Server sample applications. It also demonstrates finding an iShop Information Server using UDDI, and an iShop Shop Management Server using UDP. |

| | |
|---|---|
| GenerateDotNetCode | This folder does not represent a sample application. It contains batch files to generate Microsoft .NET code used by the sample applications. |
| InformationServer | This folder contains a sample iShop Information Server application. |
| SMSServer | This folder contains a sample iShop Shop Management Server application. |
| UdpDiscoveryServer | iShop specifies that a Shop Management Server is found by sending a UDP inquiry message over a network. As a web service application, the SMS itself will not listen for UDP requests. This folder contains a sample application that listens for UDP requests, and sends a UDP reply message back to the requestor identifying an SMS URL. |

**AaiaIShopFramework**

This folder contains a library of framework code that is used by all the other sample applications – both clients and servers. The other sample applications reference the DLL compiled from this code.

All effort has been made to encapsulate iShop plumbing into this library. This library can be used directly by an iShop implementer, and extended, or it can serve as an example.

**Microsoft UDDI SDK**

The Microsoft UDDI SDK must be installed on your computer if you wish to compile the AaiaIShopFramework project. The 2.0 Beta version of the SDK is deployed by the iSHOP installation program as file "Samples\AaiaIShopFramework\uddisdksetup.exe".

**Generated Code Files**

Generated code files are those which have been generated using tools. They are contained in folder "Samples\AaiaIShopFramework\AAIA\Generated" folder.

| | |
|---|---|
| **File:** | **iShopISWebserviceClient.vb** |
| **Namespace:** | **AAIA.iShop.ServerInterface.Client.IS** |

This file is generated by Microsoft Visual Studio .NET tool "wsdl.exe" from the iShop Information Server WSDL file "iShopInformationServer.wsdl" - see documentation section *WSDL Files*. The command line to generate this file is in batch file "GenerateProxies.bat".

This generated file contains proxy classes[1] used by a client application to invoke an iShop Information Server web service. The code in this file is used by the sample Back Shop application (file “ExampleISCalls.vb”).

| | |
|---|---|
| **File:** | **iShopISWebserviceServer.vb** |
| **Namespace:** | **AAIA.iShop.ServerInterface.Server.IS** |

This file is generated by Microsoft Visual Studio .NET tool “wsdl.exe” from the iShop Information Server WSDL file “iShopInformationServer.wsdl” - see documentation section *WSDL Files*. The command line to generate this file is in batch file “GenerateProxies.bat”.

This generated file defines an interface “IIShopInformationServerSoap” that forms the basis for an iShop Information Server web service. In particular, it defines all the Information Server web methods. This interface should be implemented by a Microsoft .NET application implementing an iShop Information Server web service.

The interface defined in this file is implemented by the sample Information Server application (file “App_Code\iShopInformationServer.vb”).

| | |
|---|---|
| **File:** | **iShopMessages.vb** |
| **Namespace:** | **AAIA.iShop.Messages** |

This file is generated by Microsoft Visual Studio .NET tool “xsd.exe” from the iShop XML message schema files - see documentation section XML Schema Files (page 43). The command line to generate this file is in batch file “GenerateMessages.bat” (page 32).

The generated code defines classes representing all messages that pass between iShop applications. It supports serializing instances of the classes into iShop XML messages by a sender, and de-serializing the XML messages back into instances of the classes by a receiver.

For example, the following client code creates a class representing a request message to an iShop SMS server application:

```
Dim requestMsg As New AAIA.iShop.Messages.GetCustomerRequestMessage
requestMsg.CustomerId = "004429"
```

1. A proxy class contains methods that stand in for web service methods and are called using normal programming techniques. The proxy class methods in turn call the web service method, handling all the requisite complexity. Therefore the proxy class makes it as easy to call a web service as it is to call a regular function.

Framework functions are available in file “iShopXmlHelper.vb” (page 23) to assist with serialization to XML and de-serialization back to objects. For example, the following code serializes object “requestMsg” into XML:

```
Dim requestXml As String = AAIA.iShop.XML.XmlHelper.Serialize(requestMsg)
```

The above serialization is done within framework code invoked by the client application – see files “iShopSmsInterfaceClient.vb” (page 22) and “iShopISInterfaceClient.vb” (page 20).

The following code in the receiving iShop SMS web service application de-serializes the XML back into an object:

```
Dim requestMsg As AAIA.iShop.Messages.GetCustomerRequestMessage

requestMsg = CType(AAIA.iShop.XML.XmlHelper.Deserialize(GetCustomerRequest,
_GetType(iShopMessages.GetCustomerRequestMessage)),
_iShopMessages.GetCustomerRequestMessage)
```

The above de-serialization is done within the sample IS and SMS server applications – see SMS sample application file “App_Code\iShopSmsServer.vb” and IS sample application file “App_Code\iShopInformationServer.vb”.

| | |
|---|---|
| **File:** | **iShopSmsWebserviceClient.vb** |
| **Namespace:** | **AAIA.iShop.ServerInterface.Client.SMS** |

This file is generated by Microsoft Visual Studio .NET tool “wsdl.exe” from the iShop Shop Management Server WSDL file “iShopSmsServer.wsdl” - see documentation section *WSDL Files*. The command line to generate this file is in batch file “GenerateProxies.bat”.

This generated file contains proxy classes[2] used by a client application to invoke an iShop Shop Management Server web service. The code in this file is used by the sample Back Shop application (file “ExampleSMSCalls.vb”).

| | |
|---|---|
| **File:** | **iShopSmsWebserviceServer.vb** |
| **Namespace:** | **AAIA.iShop.ServerInterface.Server.SMS** |

This file is generated by Microsoft Visual Studio .NET tool “wsdl.exe” from the iShop Shop

Management Server WSDL file “iShopSmsServer.wsdl” - see documentation section *WSDL Files*. The command line to generate this file is in batch file “GenerateProxies.bat”.

2 See definition for proxy class in the footnote for documentation on file “iShopISWebserviceClient.vb”.

This generated file defines an interface “IIShopSmsServerSoap” that forms the basis for an iShop Shop Management Server web service. In particular, it defines all the Shop Management Server web methods. This interface should be implemented by a Microsoft .NET application implementing an iShop Shop Management Server web service.

The interface defined in this file is implemented by the sample Shop Management Server application (file “App_Code\iShopSmsServer.vb”).

**Authored Code Files**

Authored code files are those which have been hand written as opposed to generated. They are contained in folder “Samples\AaiaIShopFramework\AAIA”.

| | |
|---|---|
| **File:** | **iShopConfigManager.vb** |
| **Namespace:** | **AAIA.iShop.Configuration** |

This file contains code to assist accessing <appSettings> configuration settings encoded within an application configuration file – e.g. “web.config” for an Microsoft ASP.NET web application such as a web service and “app.config” for a Microsoft .NET non-web application.

| | |
|---|---|
| **File:** | **iShopConstants.vb** |
| **Namespace:** | **AAIA.iShop** |

This file defines constants usable by IS and SMS client and/or and server implementations. For example, this file defines the following enumerated types:

**Table 10 - Enumerated Types**

| Type | Description |
|---|---|
| iShopServerType | Types of iShop servers – e.g. “Information Server”, “Shop Management Server”. |
| iShopISOperation | List of all Information Server web service operations. |
| iShopSmsOperation | List of all Shop Management Server web service operations. |

| | |
|---|---|
| **File:** | **iShopDiscoveryServerUdp.vb** |
| **Namespace:** | **AAIA.iShop.Discovery** |

This file contains code that forms the basis of an application that listens on a network for UDP requests for iShop Shop Management Servers, and replies with one or more that are available. Such an application is called an iShop UDP Discovery Server.

Documentation section Finding an iShop Shop Management Server using UDP (page 8) contains background information on this subject.

See documentation embedded in the sample code itself for implementation details. Although the sample code is based on the Microsoft .NET framework, the steps taken should be portable to other technologies.

The following settings may be used in the configuration file of a UDP Discovery Server application that uses this functionality:

**Table 11 - UDP Settings**

| Setting Key | Description |
|---|---|
| AAIA.iShop.Discovery.UdpGroupAddress<br><br>AAIA.iShop.Discovery.ServerPort | Settings "UdpGroupAddress" and "ServerPort" specify the IP address and port number respectively that the UDP Discovery Server listens on.<br><br>The default ServerPort value is 1235. This number must be the same as the one the client applications send UDP messages to.<br><br>Default UdpGroupAddress value is the IP address of the UDP Discovery Server computer. |
| AAIA.iShop.Discovery.DefaultGroupTimeToLive | The UDP Discovery Server joins a multicast group to listen for UDP messages. This value is specified as the "Time to Live" (TTL) parameter measured in router hops. Valid values are from 0 to 255.<br><br>Default value is 10. |

This functionality is used by the sample application UdpDiscoveryServer.

| | |
|---|---|
| **File:** | **iShopInterfaceClient.vb** |
| **Namespace:** | **AAIA.iShop.ServerInterface.Client** |

This file contains code applicable to interfacing to either an iShop Information Server OR an iShop Shop Management Server.

In particular, this code interprets exceptions thrown when invoking a web service.

| | |
|---|---|
| **File:** | **iShopISInterfaceClient.vb** |

| | |
|---|---|
| **Namespace:** | **AAIA.iShop.ServerInterface.Client.IS** |

This file contains code for interfacing to an iShop Information Server, or more specifically, to the generated proxy class that invokes an iShop Information Server web service (“iShopISWebserviceClient.vb”).

The generated proxy class contains a method for each web operation. Each such method accepts a string containing an XML input message and returns a string containing an XML output message. This class provides value-added in that it contains a method for each web operation that accepts an object representing an input message and returns an object representing an output message. Each of these objects is an instance of a class declared in file “iShopMessages.vb”. Each web operation method in this class serializes the input object to a string containing XML, calls the corresponding proxy method, and then de-serializes the proxy method response string into a return object for the caller.

| | |
|---|---|
| **File:** | **iShopMessageResponse.vb** |
| **Namespace:** | **AAIA.iShop.Messages** |

This file contains code to create Response element content. Response elements are included in all response messages from iShop servers. For example, the following Shop Management Server response message contains a response element indicating an error:

```
<?xml version="1.0" encoding="UTF-8"?>
<CreatePhotoResponse xmlns="http://www.aftermarket.org/iShop/V3/XMLSchema"
xmlns:ishop="http://www.aftermarket.org/iShop/V3/XMLSchema"
xmlns:vehicle="http://www.aftermarket.org/iShop/V3/ACESVehicle"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.aftermarket.org/iShop/V3/XMLSchema iShopMessages.xsd">
        <PhotoId></PhotoId>
        <Response>
                <Status>Fail</Status>
                <MessageList>
                        <Message>
                                <Severity>Error</Severity>
                                <Code>10</Code>
                                <ShortDescription>Short message description.</ShortDescription>
                                <Description>Full message description.</Description>
                        </Message>
                </MessageList>
        </Response>
```

</CreatePhotoResponse>

See the iShop Version 3 Functional Implementation Guide for more information on the Response element.

| | |
|---|---|
| **File:** | **iShopServerDescriptor.vb** |
| **Namespace:** | **AAIA.iShop** |

This file defines class "iShopServiceDescriptor" which identifies an iShop server. The properties of this class are:

**Table 12 - Server Descriptor**

| Property | Description |
|---|---|
| ServerType | Indicates whether the iShop server is an Information Server or a Shop Management Server. |
| ApplicationName | Name of the iShop server as defined by the server author. |
| URL | URL identifying where the iShop server web service may be found. |

This class is used by the server discovery functionality to communicate the set of iShop servers found – see files "iShopServerDiscoveryUddi.vb", "iShopServerDiscoveryUdp.vb", "iShopDiscoveryServerUdp.vb".

| | |
|---|---|
| **File:** | **iShopServerDiscoveryUddi.vb** |
| **Namespace:** | **AAIA.iShop.Discovery** |

This file contains code to find one or more iShop Information Servers using UDDI.

Documentation section Finding an iShop Information Server using UDDIcontains background information on this subject.

See documentation embedded in the sample code itself for implementation details. Although the sample code is based on the Microsoft .NET framework, the steps taken should be portable to other technologies.

This functionality is used by the sample Back Shop equipment application – file "MainForm.vb".

| | |
|---|---|
| **File:** | **iShopServerDiscoveryUdp.vb** |
| **Namespace:** | **AAIA.iShop.Discovery** |

This file contains code to find one or more iShop Shop Management Servers by sending a network request using UDP.

Documentation section Finding an iShop Shop Management Server using UDP (page 8) contains background information on this subject. This documentation section calls the application that listens for UDP requests an iShop UDP Discovery Server.

See documentation embedded in the sample code itself for implementation details. Although the sample code is based on the Microsoft .NET framework, the steps taken should be portable to other technologies.

The following settings may be used in the configuration file of an application that uses this functionality:

**Table 13 - UDP Settings**

| Setting Key | Description |
|---|---|
| AAIA.iShop.Discovery.UdpGroupAddress<br>AAIA.iShop.Discovery.ServerPort | Settings "UdpGroupAddress" and "ServerPort" specify the IP address and port number respectively that the UDP message is sent to.<br>The default ServerPort value is 1235. This number must be the same as the port the UDP Discovery Server listens on.<br>The default UdpGroupAddress value is 255.255.255.255 indicating a general network broadcast. |
| AAIA.iShop.Discovery.DefaultMillisecondUdpResponseWait | Indicates how long this functionality should wait after sending the UDP for responses to come back.<br>The default value is 3000 (3 seconds). |

This functionality is used by the sample Back Shop equipment application – file "MainForm.vb".

| | |
|---|---|
| **File:** | **iShopServerSecurity.vb** |
| **Namespace:** | **AAIA.iShop.Server.Security** |

This file contains code to support security for interfacing from a client to an iShop Server. There is minimal code in this file at present since the current release of iShop does not use security conventions requiring implementation in code. This file should provide standard usages of such conventions should any be implemented in the future.

| | |
|---|---|
| **File:** | **iShopSmsInterfaceClient.vb** |
| **Namespace:** | **AAIA.iShop.ServerInterface.Client.SMS** |

This file contains code for interfacing to an iShop Shop Management Server, or more specifically, to the generated proxy class that invokes an iShop Shop Management web service ("iShopSmsWebserviceClient.vb").

The generated proxy class contains a method for each web operation. Each such method accepts a string containing an XML input message and returns a string containing an XML output message. This class provides value-added in that it contains a method for each web operation that accepts an object representing an input message and returns an object representing an output message. Each of these objects is an instance of a class declared in file "iShopMessages.vb". Each web operation method in this class serializes the input object to a string containing XML, calls the corresponding proxy method, and then de-serializes the proxy method response string into a return object for the caller.

| | |
|---|---|
| **File:** | **iShopUtility.vb** |
| **Namespace:** | **AAIA.iShop.Utility** |

This file contains general utility functions that may be used within the framework or by any application using the framework. Functionality implemented includes:

| Function | Description |
|---|---|
| EncodeFileToHexBinary | Some XML messages to and/or from iShop servers include hexbinary encodings of files such as pictures (jpegs) or zip files. |
| | This function accepts a file path and returns an embeddable hexbinary byte array of the file contents. |
| EncodeTokens,<br>DecodeTokens | Contains functionality to encode an array of strings into a single string, and decode that string back into an array of strings.<br><br>This functionality is used to create and interpret the message sent from a UDP Discovery Server identifying an iShop server available on the network – see code file "iShopDiscoveryServerUdp.vb". |

| | |
|---|---|
| **File:** | **iShopXmlHelper.vb** |
| **Namespace:** | **AAIA.iShop.XML** |

This file contains XML handling code usable by both iShop servers and their clients – including functionality to serialize objects representing messages to strings containing XML, and deserialize strings containing XML back to objects.

Classes for the generated message objects are declared in "iShopMessages.vb".

**BackShop**

This folder contains a sample Back Shop equipment application. It demonstrates calling both the iShop Information Server and Shop Management Server sample applications.

The sample Back Shop equipment application also demonstrates finding an iShop Information Server using UDDI, and an iShop Shop Management Server using UDP. Documentation section Error Handling contains background information on this subject.

The sample Back Shop equipment application is a Microsoft .NET windows form application. Therefore it does not contain code to preserve session state as would be required if it were a web application.

**Running the Sample Back Shop Equipment Application**

You may run the sample Back Shop equipment application either from within Microsoft Visual Studio 2005, or by running “BackShop.exe” within the “Bin” sub-folder.

The sample Back Shop equipment application can be used to communicate with any iShop Version 3 Information Server or Shop Management Server. However, if you want it to communicate with the sample iShop servers, you will have to create Microsoft Internet Information Services (IIS) virtual directories that reference the folders containing the “asmx” web service files “iShopInformationServer.asmx” and “iShopSmsServer.asmx”.

The main screen of the sample Back Shop Equipment application appears as follows:

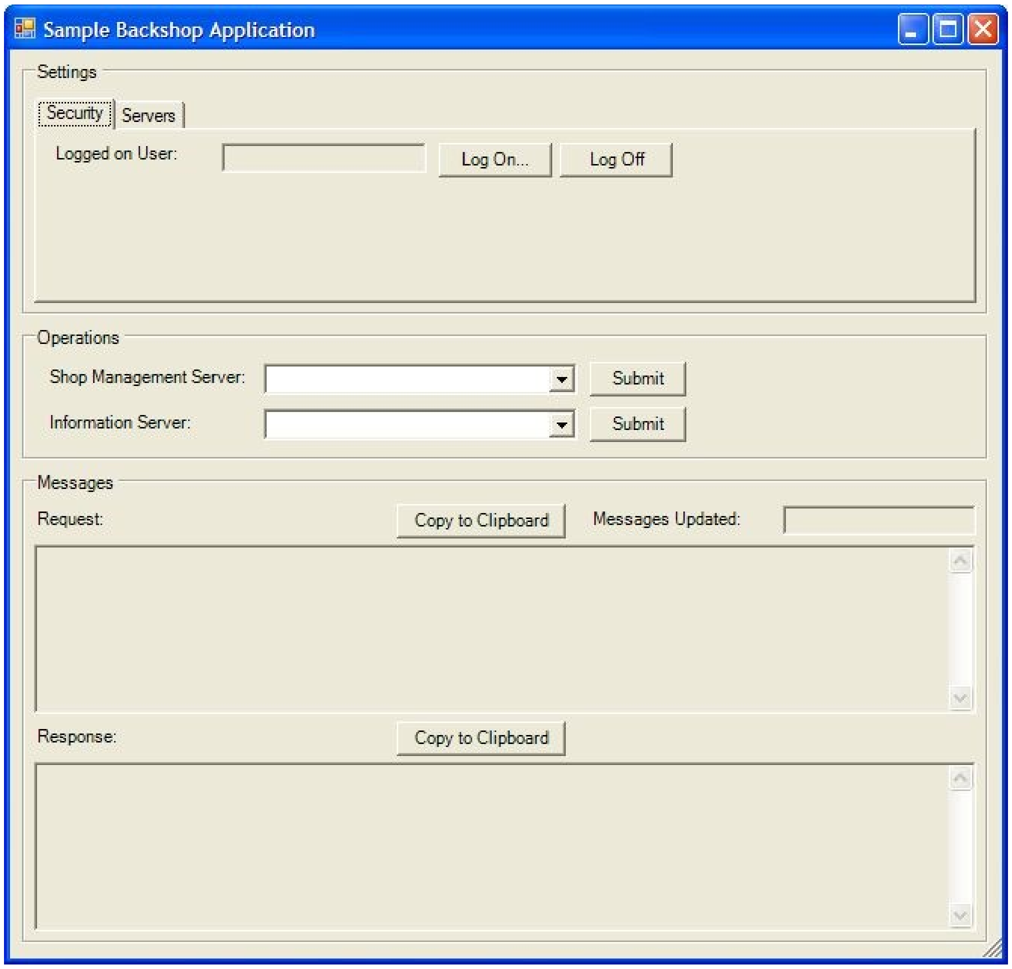
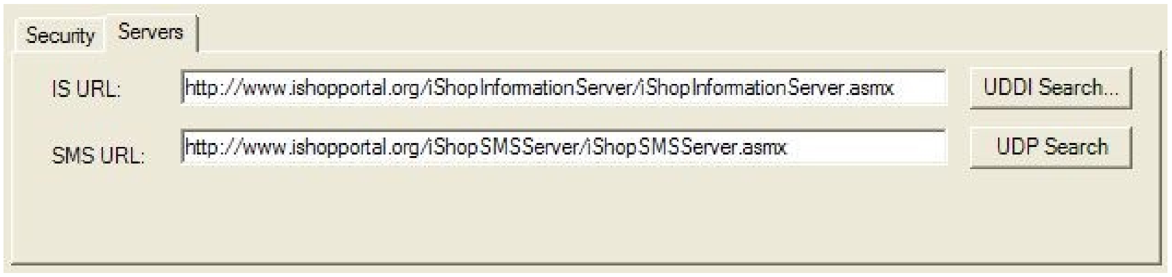


**Figure 2 - Sample Backshop UI**

The “Security” tab in the “Settings” group is used to set or clear a logon user name and password that is sent to the Information Server Open operation.  Default values may be specified for these values in the “app.config” file.

The “Servers” tab in the “Settings” group appears as follows:



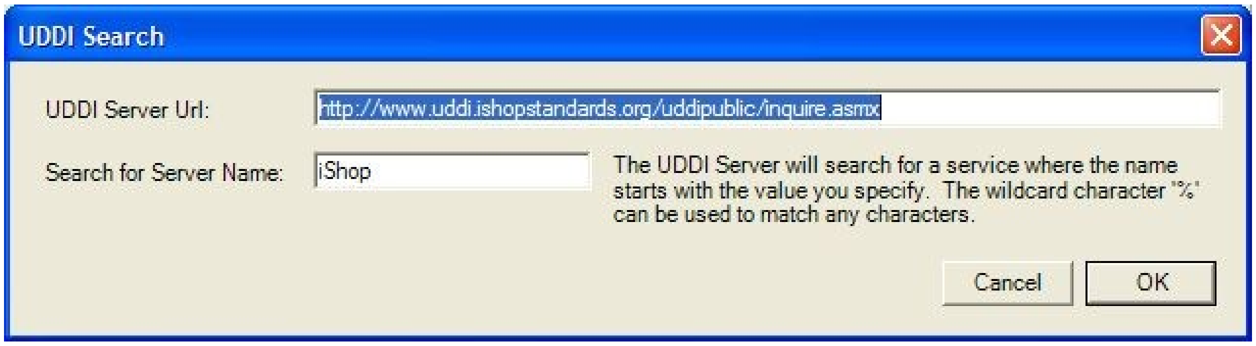
IS URL: http://www.ishopportal.org/iShopInformationServer/iShopInformationServer.asmx

SMS URL: http://www.ishopportal.org/iShopSMSServer/iShopSMSServer.asmx

**Figure 3 - Sample Backshop Settings**

This tab identifies the iShop Version 3 servers that the sample Back Shop equipment application will call when the user clicks a “Submit” button in the “Operations” group. Default values may be specified for these values in the “app.config” file.

Clicking the “UDDI Search…” button opens a dialogue window that asks for a UDDI Server URL and a service name search string. This window appears as follows:



UDDI Server Url: http://www.uddi.ishopstandards.org/uddipublic/inquire.asmx

Search for Server Name: iShop

The UDDI Server will search for a service where the name starts with the value you specify. The wildcard character '%' can be used to match any characters.

**Figure 4 – UDDI Search Dialogue Window**

Default values are specified for the above fields in the “app.config” file. Clicking the

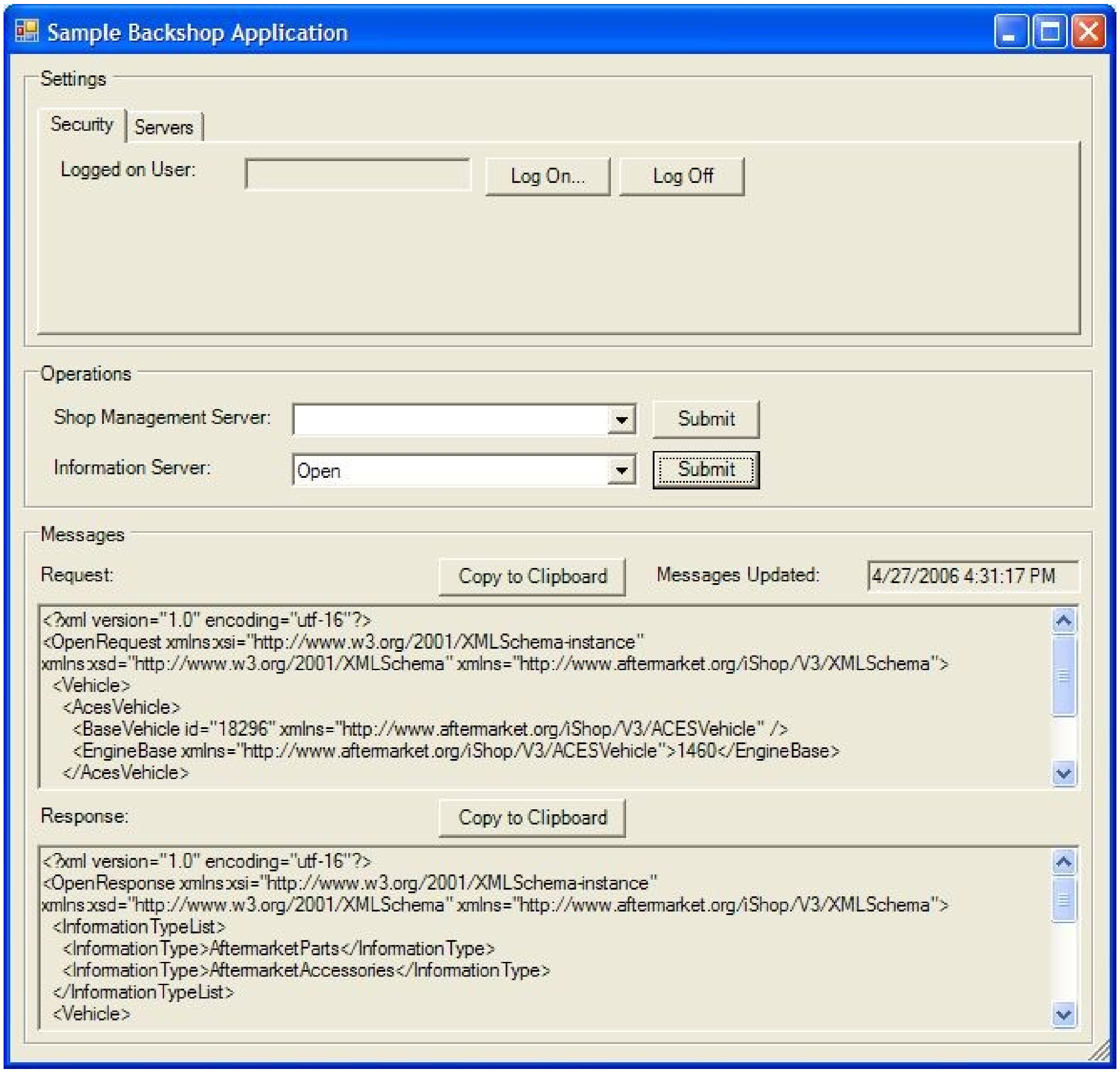
“OK” button in this dialogue window invokes framework code “iShopServerDiscoveryUddi.vb” to search for an Information Server by server name. This process is described in documentation section Finding an iShop Information Server using UDDI.

Clicking the “UDP Search” button will invoke framework code “iShopServerDiscoveryUdp.vb”

(page 21) to search for an SMS Server. This process is described in documentation section Finding an iShop Shop Management Server using UDP. The following are required in order for this to work:

1. An iShop UDP Discovery Server must be running and accessible on the network. A sample server is deployed with iShop and is described in documentation section UdpDiscoveryServer.
2. “app.config” file values such as “UdpGroupAddress” and “ServerPort” must be set and must be compatible with the configuration of the UDP Discovery Server.

Once the above settings are in place, the sample BackShop equipment application is used by selecting a Shop Management Server or Information Server setting in the “Operations” group and clicking the adjacent “Submit” button. The sample Back Shop equipment application will construct and send a request message and wait for a response. Both the request and response messages are then displayed in the “Messages” group as per the following example:



Logged on User:

Shop Management Server:

Information Server: Open

Request:

Messages Updated: 4/27/2006 4:31:17 PM

```
<?xml version="1.0" encoding="utf-16"?>
<OpenRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.aftermarket.org/iShop/V3/XMLSchema">
  <Vehicle>
    <AcesVehicle>
      <BaseVehicle id="18296" xmlns="http://www.aftermarket.org/iShop/V3/ACESVehicle" />
      <EngineBase xmlns="http://www.aftermarket.org/iShop/V3/ACESVehicle">1460</EngineBase>
    </AcesVehicle>
```

Response:

```
<?xml version="1.0" encoding="utf-16"?>
<OpenResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.aftermarket.org/iShop/V3/XMLSchema">
  <InformationTypeList>
    <InformationType>AftermarketParts</InformationType>
    <InformationType>AftermarketAccessories</InformationType>
  </InformationTypeList>
  <Vehicle>
```

**Figure 5 - Sample Backshop Monitoring**

**Code Files**

**File: App_Code\General\AppGlobal.vb**

This file declares a class to hold global variables used by the sample Back Shop equipment application. A single instance is created by code in file “MainModule.vb”. A public variable containing this instance makes the global variables available to all code in the sample application, and preserves the global variables throughout the life of the sample application.

This class supports the following public properties:

**Table 14 - Global Variables**

| **Property** | **Description** |
|---|---|
| LogonUserName | User name and password specified by the user using the “Log On…” prompt. This information is not presently passed to the sample servers. |
| LogonUserPassword (ReadOnly) | |
| ISUrl | Information Server URL. |
| SmsUrl | Shop Management Server URL. |
| UddiServerUrl | UDDI Server and service name search string values used to search for iShop servers using UDDI. See framework code “iShopServerDiscoveryUddi.vb”. |
| UddiServerNameSrch | |
| ISUserarea | The sample Information Server application returns a top-level UserArea element and a Vehicle element in most response messages. It expects the client application to save this data, and return it to the Information Server on its next request. These properties are used to hold this data during this interval.<br><br>See documentation section *UserArea and Vehicle Preservation* for more information on this convention. |
| Vehicle | |

**File: App_Code\WebOperations\ExampleISCalls.vb**

This file contains code demonstrating how to call each Information Server web operation. Code to call each web operation is encapsulated into its own function. Each function takes the following steps:

1. Create an object representing a request message.

   Classes for all messages (request and response) are defined by code generated from the iShop XML schemas – see documentation section *XML Schema Files* regarding the schemas, and file “iShopMessages.vb” regarding the class generated from these schemas.

This step is valuable as it demonstrates how to construct the various Information Server request messages using the classes generated from iShop XML schemas.

Preserved vehicle and top-level UserArea elements are populated into the request message object as per documentation section *UserArea and Vehicle Preservation*.

2. Invoke the iShop Information Server web service.

   This is done using functionality from framework class iShopISInterfaceClient – see “iShopISInterfaceClient.vb”.

3. Process the response message.

   Processing includes extracting out returned vehicle and top-level UserArea elements that should be passed back on the next request. As mentioned above, this convention is described in documentation section *UserArea and Vehicle Preservation*.

4. Finish, returning strings representing request and response XML.

   The sample Back Shop equipment application displays these XML messages to the user for the purposes of demonstration only. These values would not typically be returned in a real Back Shop equipment application.

Note that this code does not concern itself with any of the technical aspects of web services. This is intentional. It can therefore concern itself only with business processing. Since it is not the objective of these samples to demonstrate business processing, each web operation is called with a hard-coded message, and no real processing is done on the response.

Example code is as follows:

```
Imports iShopMessages = AAIA.iSHOP.Messages
Imports iShopXml = AAIA.iShop.XML
Imports Client = AAIA.iSHOP.ServerInterface.Client.IS

Sub Navigate(ByRef requestXml As String, ByRef responseXml As String)
        ' 1) Build request message

        Dim requestMsg As New iShopMessages.NavigateRequestMessage
        requestMsg.NavigationItem = New iShopMessages.NavigationItemType
        requestMsg.NavigationItem.Text = "PROBE"
        requestMsg.NavigationItem.ServerItemId = "22"
```

```
            ' Pass in standard elements that should be sent on subsequent calls
            requestMsg.Vehicle = GlobalVars.Vehicle
            requestMsg.UserArea = GlobalVars.ISUserarea

            ' 2) Invoke iSHOP Server

            Dim server As New Client.iShopIsInterfaceClient(GlobalVars.ISUrl)
            Dim responseMsg As iShopMessages.NavigateResponseMessage
            responseMsg = server.Navigate(requestMsg)

            ' 3) Process response message

            ' Save standard elements that should be sent on subsequent calls
            GlobalVars.Vehicle = responseMsg.Vehicle
            GlobalVars.ISUserarea = responseMsg.UserArea

            ' 4) Done - return request/response XML
            requestXml = iShopXml.XmlHelper.Serialize(requestMsg)
            responseXml = iShopXml.XmlHelper.Serialize(responseMsg)
        End Sub
```

**File: App_Code\WebOperations\ExampleSMSCalls.vb**

This file contains code demonstrating how to call each Shop Management Server web operation. Code to call each web operation is encapsulated into its own function. Each function takes the following steps:

1. Create an object representing a request message.

   Classes for all messages (request and response) are defined by code generated from the iShop XML schemas – see documentation section *XML Schema Files* regarding the schemas, and file "iShopMessages.vb" regarding the class generated from these schemas.

   This step is valuable as it demonstrates how to construct the various Shop Management Server request messages using the classes generated from iShop XML schemas.

2. Invoke the iShop Shop Management Server web service.

   This is done using functionality from framework class iShopSmsInterfaceClient – see "iShopSmsInterfaceClient.vb".

3. Process the response message.

4. Finish, returning strings representing request and response XML.

   The sample Back Shop equipment application displays these XML messages to the user for the purposes of demonstration only. These values would not typically be returned in a real Back Shop equipment application.

Note that this code does not concern itself with any of the technical aspects of web services. This is intentional. It can therefore concern itself only with business processing. Since it is not the objective of these samples to demonstrate business processing, each web operation is called with a hard-coded message, and no real processing is done on the response.

Example code is as follows:

```
Imports iShopMessages = AAIA.iSHOP.Messages
Imports iShopXml = AAIA.iShop.XML
Imports Client = AAIA.iSHOP.ServerInterface.Client.SMS

Sub GetCustomer(ByRef requestXml As String, ByRef responseXml As String)
        ' 1) Build request message

        Dim requestMsg As New iShopMessages.GetCustomerRequestMessage
        requestMsg.CustomerId = "004429"

        ' 2) Invoke iSHOP Server

        Dim server As New Client.iShopSmsInterfaceClient(GlobalVars.SmsUrl)
        Dim responseMsg As iShopMessages.GetCustomerResponseMessage
        responseMsg = server.GetCustomer(requestMsg)

        ' 3) Process response message

        ' 4) Done - return request/response XML
        requestXml = iShopXml.XmlHelper.Serialize(requestMsg)
        responseXml = iShopXml.XmlHelper.Serialize(responseMsg)
End Sub
```

**File: LogonForm.vb**

This windows form accepts user id and password values that are passed to the Information

Server Open operation. The values entered are preserved in global variables – see “AppGlobal.vb”.

**File: MainForm.vb**

This is the main form of the sample Back Shop equipment application. See documentation section *Running the Sample Back Shop Equipment Application* for the appearance of this form, and documentation on how it should be used.

The following event scripts are noteworthy in terms of demonstrating iShop functionality:

**Table 15 - Event Scripts**

| Event Script | Description |
|---|---|
| btnSmsSubmit_Click | Invokes the Shop Management Server operation selected by the user. This is done by calling the appropriate function in file “ExampleSMSCalls.vb”. |
| btnISSubmit_Click | Invokes the Information Server operation selected by the user. This is done by calling the appropriate function in file “ExampleISCalls.vb”. |
| btnIsUddiSearch_Click | Initiates a UDDI search for iShop Information Servers. UDDI searching is done by framework code in file “iShopServerDiscoveryUddi.vb”. This event’s code shows how to invoke the framework function, and how to interpret the results. |
| btnSmsUdpSearch_Click | Initiates a UDP search for iShop Shop Management Servers. UPP searching is done by framework code in file “iShopServerDiscoveryUdp.vb”. This event’s code shows how to invoke the framework function, and how to interpret the results. |

**File: MainModule.vb**

The starting point of the sample Back Shop equipment application is configured to be procedure “Main” in this code file.

The starting functionality initializes global variables (see “AppGlobal.vb”) and opens the main form of the sample application (see “MainForm.vb”, page 29).

**File: app.config**

This is the application configuration file for the sample Back Shop equipment application.

Settings are read from this file using framework code “iShopConfigManager.vb”. Available settings are as follows:

**Table 16 - Backshop Equipment App Settings**

| **Setting Key** | **Description** |
|---|---|
| TestInformationServerUrl | Default iShop server URLs to send requests to. These values may be changed by the user in the sample Back Shop equipment application user interface. |
| TestSmsServerUrl | |
| DefaultLogonUser | Default user name and password passed to the Information Server *Open* operation. These values may be changed by the user in the sample Back Shop equipment application user interface. |
| DefaultLogonPassword | |
| OpenVehicleType | The sample Back Shop equipment application |
| | passes a Vehicle element to the Information Server Open operation. Such elements may identify vehicles using either the legacy AAIA identifier, or newer ACES nomenclature – see the iShop Functional Implementation Guide for more details. This setting accepts values “Legacy” or “ACES” indicating which identifier to use. The default value for this setting is “ACES”. |
| AAIA.iShop.Discovery.UdpGroupAddress | |
| AAIA.iShop.Discovery.ServerPort | |

| | |
|---|---|
| AAIA.iShop.Discovery.DefaultMillisecondUdpResponseWait | These are settings to framework code “iShopServerDiscoveryUdp.vb”. They are described in the documentation for that code. |
| UddiInquireUrl | URL of the UDDI server that registers iShop Information Servers and that should be searched. The default value for this setting references the AAIA UDDI server as described in documentation section *Finding an iShop Information Server using UDDI*.<br><br>The value specified here is a default value that can be overridden within a dialogue window of the sample BackShop equipment application. |
| UddiServiceNameSrch | Service name search string for finding iShop Information Servers registered by the UDDI server to search. See documentation section *Finding an iShop Information Server using UDDI* for more details.<br><br>The UDDI server will return all web services starting with the specified search string value. Wildcard character “%” can be used to match a sequence of any characters.<br><br>The value specified here is a default value that can be overridden within a dialogue window of the sample BackShop equipment application. |

**GenerateDotNetCode**

This folder does not represent a sample application. It contains batch files to generate Microsoft .NET code used by the sample applications.

In order to run, this directory must contain copies of the iShop WSDL and schema (XSD) files.

These files are described in documentation section *XML*.

**Batch Files**

**File: GenerateMessages.bat**

This batch file runs Microsoft Visual Studio .NET tool “xsd.exe” to generate code files of classes from iShop schema (XSD) files. This batch file must be run from a “Visual Studio 2005 Command Prompt” in order for the “xsd.exe” program to be found and run correctly. Copies of the iShop schema files must be included in this directory – see documentation section *XML Schema Files*.

The following files are generated by this batch file:

| File | Description |
|---|---|
| iShopMessages.vb | This file is generated solely from iShop standard schema files, and is compiled into the framework application. See documentation for framework file “iShopMessages.vb”. |
| MyCompanyCustom.vb | This file is generated from the non-iShop standard schema file “MyCompanyCustom.xsd” that defines valid content for the highlevel UserArea element of messages to and from the sample Information Server application used to preserve state. See documentation section *UserArea and Vehicle Preservation* for more information.<br><br>This file could be compiled into the sample Information Server application, or the sample Back Shop application that uses it. It would not, however, be appropriate to compile these companyspecific classes into the framework. |

**File: GenerateProxies.bat**

This batch file runs Microsoft Visual Studio .NET tool “wsdl.exe” to create client and server web service code from the iShop WSDL files. This batch file must be run from a “Visual Studio 2005 Command Prompt” in order for the “wsdl.exe” program to be found and run correctly. Copies of the WSDL files must be included in this directory – see documentation section *WSDL Files*.

The following framework files are generated by this batch file:

| File | Description |
|---|---|

| | |
|---|---|
| iShopISWebserviceClient.vb | Client proxy file for an iShop Information Server web service. See documentation for framework file “iShopISWebserviceClient.vb”. |
| iShopSmsWebserviceClient.vb | Client proxy file for an iShop Shop Management Server web service. See documentation for framework file “iShopSmsWebserviceClient.vb”. |
| iShopISWebserviceServer.vb | Declares an interface that should be implemented by an iShop Information Server web service. See documentation for framework file “iShopISWebserviceServer.vb”. |
| iShopSmsWebserviceServer.vb | Declares an interface that should be implemented by an iShop Shop Management Server web service. See documentation for framework file “iShopSmsWebserviceServer.vb”. |

**InformationServer**

This folder contains a sample iShop Information Server application.

The sample Information Server application is a Microsoft ASP.NET web service application. To make this sample web service available, create a Microsoft Internet Information Services (IIS) virtual directory referencing the folder containing file “iShopInformationServer.asmx”.

**UserArea and Vehicle Preservation**

An iShop Version 3 Information Server is a statefull web service in that a client application opens a session, invokes operations to search for information, and then closes the session. Therefore session state information must be preserved for each client.

The sample Information Server application adopts the approach recommended by ALLDATA whereby a session identifier is returned by the Information Server embedded in user-defined XML in the top-level UserArea element for all response messages that occur within the scope of a conversation.

For example, see the <mycompany:SessionId> element in the response message from the Open operation, below:

```
<?xml version="1.0" encoding="utf-16"?>
<OpenResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.aftermarket.org/iShop/V3/XMLSchema">
        <InformationTypeList>
                <InformationType>AftermarketParts</InformationType>
                <InformationType>AftermarketAccessories</InformationType>
        </InformationTypeList>
        <Vehicle>
                <AcesVehicle>
                        <BaseVehicle id="18296"
                        xmlns="http://www.aftermarket.org/iShop/V3/ACESVehicle"/>
                        <EngineBase
                        xmlns="http://www.aftermarket.org/iShop/V3/ACESVehicle">1460</EngineBase>
                </AcesVehicle>
        </Vehicle>
        <Response>
                <Status>OK</Status>
        </Response>
        <UserArea>
                <mycompany:SessionId
                xmlns:mycompany="http://www.mycompany.com/iShop/V3/XMLSchema">8645</m
                ycompany:SessionI d>
        </UserArea>
</OpenResponse>
```

The content “<mycompany:SessionId>” is validated by custom schema “MyCompanyCustom.xsd”.

This convention further requires the client embed the top-level UserArea element containing the session identifier in the request message for the next operation. For example:

```
<?xml version="1.0" encoding="utf-16"?>
<NavigateRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.aftermarket.org/iShop/V3/XMLSchema">
        <NavigationItem>
                <Text>PROBE</Text>
                <ServerItemId>22</ServerItemId>
        </NavigationItem>
        <Vehicle>
                <AcesVehicle>
                        <BaseVehicle id="18296"
                        xmlns="http://www.aftermarket.org/iShop/V3/ACESVehicle" />
                        <EngineBase
                        xmlns="http://www.aftermarket.org/iShop/V3/ACESVehicle">1460</EngineBase>
                </AcesVehicle>
        </Vehicle>
        <UserArea>
                <mycompany:SessionId
                xmlns:mycompany="http://www.mycompany.com/iShop/V3/XMLSchema">8645</m
                ycompany:SessionI d>
        </UserArea>
</NavigateRequest>
```

Notice that the Vehicle element returned by the Information Server is also passed back in the next request. This is another convention implemented by the sample code that is recommended by ALLDATA. Using this convention, information known about the vehicle, and thus embedded within the Vehicle element, may be refined by the Information Server when navigating down the information hierarchy.

Each web operation implementation is responsible for extracting vehicle and top-level UserArea from the request message, validating the session, and re-embedding updated top-level UserArea and vehicle information into the response message. See documentation section *Web Operation Implementation Code Files* for more details. The client for each web operation is responsible for holding the returned vehicle and top-level UserArea elements, and then passing them back when calling the next operation in the session.

**Web Methods Code Files**

| | |
|---|---|
| **File:** | **App_Code\iShopInformationServer.vb** |
| **Namespace:** | **MyCompany.iShop.ServerInterface.Server.IS** |

This file declares the class “iShopInformationServer” that implements the sample Information Server web service. It contains a public web method for each Information Server web operation.

As required by Microsoft ASP.NET web services, class “iShopInformationServer” inherits from “System.Web.Services.WebService”. It also is the “code behind” file for the ASP.NET web service page “iShopInformationServer.asmx”.

In order to implement the operations of an iShop Information Server web service, this class implements interface “IIShopInformationServerSoap” declared in framework file

“iShopISWebserviceServer.vb” that was generated from the iShop Information Server WSDL.

Each web method implementation takes the following steps (a code example follows):

1. De-serialize the received request XML string into a request message object.

   De-serialization will fail if the XML does not match the structure of the expected request message class. Accordingly, a de-serialization exception is caught by code that causes a standard “Invalid iShop XML request document” error response being returned to the client.

   Note that framework code “iShopXmlHelper.vb” (page 23) is used to de-serialize the request XML and throw a custom exception when the XML is invalid. Framework code “iShopMessageResponse.vb” (page 20) is used to construct the “invalid document” error response.

2. Invoke the web operation business logic passing the request message object, and receiving back a response message object.

   Each web operation is implemented by a class declared in its own file – see documentation section Web Operation Implementation Code Files (page 36).

3. Serialize the response message object into an XML string, and return that string to the web service caller.

   Framework code “iShopXmlHelper.vb” (page 23) is used to assist with serialization.

Example code implementing the GetInformationType operation is as follows:

```
Imports iShopMessages = AAIA.iShop.Messages
Imports iShopXml = AAIA.iSHOP.XML

Public Function GetInformationTypes(ByVal GetInformationTypesRequest As String _ ) As String
Implements
AAIA.iShop.ServerInterface.Server.IS.IIShopInformationServerSoap.GetInformationTypes
        Dim requestMsg As iShopMessages.GetInformationTypesRequestMessage = Nothing
```

```
            Dim responseMsg As iShopMessages.GetInformationTypesResponseMessage =
            Nothing
            Dim errorResponse As iShopMessages.ResponseType = Nothing

    ' De-serialize request message
    Try
            requestMsg = CType(iShopXml.XmlHelper.Deserialize(GetInformationTypesRequest, _
            GetType(iShopMessages.GetInformationTypesRequestMessage)), _
            iShopMessages.GetInformationTypesRequestMessage)
            Catch ex As XmlMessageException
            ' Request message from client is not valid (cannot be deserialized
            ' into required type)
            errorResponse = _
            iShopMessages.iShopMessageResponse.ExceptionResponse_InvalidDocument(True, _
            ex.Message)
     End Try

     If errorResponse Is Nothing Then
            ' Invoke business operation to obtain response message
            Dim implementation As New WebMethods.GetInformationTypes
            responseMsg = implementation.Operation(requestMsg)
     Else
            ' Package up error into a response message of the correct type
            responseMsg = New iShopMessages.GetInformationTypesResponseMessage
            responseMsg.Response = errorResponse
     End If

            ' Serialize and return response message
            Dim responseXml As String = iShopXml.XmlHelper.Serialize(responseMsg)
            Return responseXml
    End Function
```

**General Code Files**

The following code files are in the “App_Code\General” folder.

| | |
|---|---|
| **File:** | **MessageResponse.vb** |
| **Namespace:** | **MyCompany.iShop.Messages** |

This file contains code to create Response element content.

This file differs from framework file “iShopMessageResponse.vb” in that it implements implementation-specific responses as opposed to general framework responses. Code in this file uses support functionality from framework file “iShopMessageResponse.vb”.

| | |
|---|---|
| **File:** | **Session.vb** |
| **Namespace:** | **MyCompany.iShop** |

This file declares class “Session” that represents an Information Server session, and contains functionality to create and restore an Information Server session.

This functionality is not included in the framework since the iShop specification does not specify how Information Server sessions should be managed.

The *Open* web operation uses this class to start a session, the *Close* web operation uses this class to end a session, and all web operations that must be called within the scope of a session use this class to ensure a valid session is in place. See documentation section *Web Operation Implementation Code Files* for more details.

| | |
|---|---|
| **File:** | **XmlHelper.vb** |
| **Namespace:** | **MyCompany.iShop.XML** |

This file contains XML handling code that is particular to this Information Server implementation. Code in this file uses functionality from framework file “iShopXmlHelper.vb”.

In particular, this file contains a method to return a Microsoft .NET

“System.Xml.XmlNamespaceManager” object that includes the standard iShop message namespaces plus the custom namespace used to represent UserArea content as described in documentation section *UserArea and Vehicle Preservation*.

**Web Operation Implementation Code Files**

The sample Information Server application contains a separate code file for each web operation, each of which defines a class named after the web operation. These files are in folder “App_Code\Web Operations”.

Each web operation class contains a method “Operation” that accepts an object representing a request message, and returns an object representing a response message. The classes defining these objects are declared by code generated from the iShop XML schemas – see documentation section *XML*

*Schema Files* regarding the schemas, file “iShopMessages.vb” regarding the class generated from these schemas.

Note that the code implementing method “Operation” need not concern itself with any of the technical aspects of web services. This is intentional. The method implementation can therefore concern itself only with business processing. Since it is not the objective of these samples to demonstrate business processing, each operation implementation returns a hardcoded message. These implementations are still worth reviewing as they demonstrate how to construct response messages using the classes generated from the iShop XML schemas.

Each web operation extracts the vehicle and top-level UserArea information that should have been sent in the request message by the client[3] – see documentation section *UserArea and Vehicle Preservation*. If the operation must be called within the scope of an Information Server session, the web operation ensures that a valid session token has been encapsulated in the received top-level user area. Web operation functionality may update the top-level UserArea or vehicle information. Just before finishing, final instances of this information should be embedded into the response message.

Example code from “GetInformationType.vb” for the GetInformationType operation is as follows:

```
Imports iShopMessages = AAIA.iSHOP.Messages

Public Function Operation( _
        ByVal requestMsg As iShopMessages.GetInformationTypesRequestMessage _ ) As
        iShopMessages.GetInformationTypesResponseMessage
        ' Vehicle is neither received by nor returned from this Operation

        Dim responseMsg As New iShopMessages.GetInformationTypesResponseMessage
        Dim lclSession As New Session(requestMsg.UserArea)
        If lclSession.SessionId Is Nothing Then
                ' Operation must take place within the scope of a session
                responseMsg.Response = Messages.MessageResponse.NoSession()
                Return responseMsg
        End If

        Me.PopulateExampleResult(responseMsg)
```

3 Not all request messages accept Vehicle elements. Message structure is documented in the iSHOP Functional Implementation Guide.

```
        responseMsg.Response = iShopMessages.iShopMessageResponse.Response_OK()
        ' Always pass back received Vehicle, possibly modified by
        ' operation functionality, and top-level UserArea.
        responseMsg.UserArea = requestMsg.UserArea

        Return responseMsg
    End Function

    Private Sub PopulateExampleResult( _
        ByVal responseMsg As iShopMessages.GetInformationTypesResponseMessage _ )
        Dim InformationTypeList(2) As iShopMessages.InformationType
        InformationTypeList(0) = iShopMessages.InformationType.LaborInformation
        InformationTypeList(1) = iShopMessages.InformationType.OEParts
        InformationTypeList(2) = iShopMessages.InformationType.RepairInformation

        responseMsg.InformationTypeList = InformationTypeList

        Return
    End Sub
```

**Other Files**

**File: iShopInformationServer.asmx**

This is the Microsoft ASP.NET file that represents the iShop Information Server web service.

**SMSServer**

This folder contains a sample iShop Shop Management Server application.

The sample Shop Management Server application is a Microsoft ASP.NET web service application. To make this sample web service available, create a Microsoft Internet Information Services (IIS) virtual directory referencing the folder containing file “iShopSmsServer.asmx”.

**Web Methods Code Files**

| | |
|---|---|
| **File:** | **App_Code\iShopSmsServer.vb** |
| **Namespace:** | **MyCompany.iShop.ServerInterface.Server.SMS** |

This file declares the class "iShopSmsServer" that implements the sample Shop Management Server web service. It contains a public web method for each Shop Management web operation.

As required by Microsoft ASP.NET web services, class "iShopSmsServer" inherits from "System.Web.Services.WebService". It also is the "code behind" file for the ASP.NET web service page "iShopSmsServer.asmx".

In order to implement the operations of an iShop Shop Management web service, this class implements interface "IIShopSmsServerSoap" declared in framework file "iShopSmsWebserviceServer.vb" that was generated from the iShop Shop Management Server WSDL.

Each web method implementation takes the following steps (a code example follows):

1. De-serialize the received request XML string into a request message object.

   De-serialization will fail if the XML does not match the structure of the expected request message class. Accordingly, a de-serialization exception is caught by code that causes a standard "Invalid iShop XML request document" error response being returned to the client.

   Note that framework code "iShopXmlHelper.vb" is used to de-serialize the request XML and throw a custom exception when the XML is invalid. Framework code "iShopMessageResponse.vb" is used to construct the "invalid document" error response.

2. Invoke the web operation business logic passing the request message object, and receiving back a response message object.

   Each web operation is implemented by a class declared in its own file – see documentation section *Web Operation Implementation Code Files*.

3. Serialize the response message object into an XML string, and return that string to the web service caller.

   Framework code "iShopXmlHelper.vb" is used to assist with serialization.

Example code implementing the GetTechnicianList operation is as follows:

```
Imports iShopMessages = AAIA.iShop.Messages
Imports iShopXml = AAIA.iSHOP.XML
```

```
Public Function GetTechnicianList(ByVal GetTechnicianListRequest As String _ ) As String
Implements _ AAIA.iShop.ServerInterface.Server.SMS.IIShopSmsServerSoap.GetTechnicianList
        Dim requestMsg As iShopMessages.GetTechnicianListRequestMessage = Nothing
        Dim responseMsg As iShopMessages.GetTechnicianListResponseMessage = Nothing
        Dim errorResponse As iShopMessages.ResponseType = Nothing

        ' De-serialize request message
        Try
                requestMsg =
                CType(iShopXml.XmlHelper.Deserialize(GetTechnicianListRequest, _
                GetType(iShopMessages.GetTechnicianListRequestMessage)), _
                iShopMessages.GetTechnicianListRequestMessage)
        Catch ex As XmlMessageException
                ' Request message from client is not valid (cannot be deserialized into
                ' required type)
                errorResponse = _
                iShopMessages.iShopMessageResponse.ExceptionResponse_InvalidDocument
                (True, _ ex.Message)
        End Try

        If errorResponse Is Nothing Then
                ' Invoke business operation to obtain response message
                Dim implementation As New WebMethods.GetTechnicianList
                ResponseMsg = implementation.Operation(requestMsg)
        Else
                ' Package up error into a response message of the correct type
                responseMsg = New iShopMessages.GetTechnicianListResponseMessage
                responseMsg.Response = errorResponse
        End If

        ' Serialize and return response message
        Dim responseXml As String = iShopXml.XmlHelper.Serialize(responseMsg)
        Return responseXml
End Function
```

**General Code Files**

No code files are in the “App_Code\General” folder for the sample iShop Shop Management Server.

**Web Operation Implementation Code Files**

The sample Shop Management Server application contains a separate code file for each web operation, each of which defines a class named after the web operation. These files are in the “App_Code\Web Operations” folder.

Each web operation class contains a method “Operation” that accepts an object representing a request message, and returns an object representing a response message. The object classes are defined by code generated from the iShop XML schemas – see documentation section *XML Schema Files* regarding the schemas, file “iShopMessages.vb” regarding the class generated from these schemas.

Note that the code implementing method “Operation” need not concern itself with any of the technical aspects of web services. This is intentional. The method implementation can therefore concern itself only with business processing. Since it is not the objective of these samples to demonstrate business processing, each operation implementation returns a hardcoded message. These implementations are still worth reviewing as they demonstrate how to construct response messages using the classes generated from the iShop XML schemas.

Example code from “GetTechnicianList.vb” for the GetTechnicianList operation is as follows:

```
Imports iShopMessages = AAIA.iSHOP.Messages

Public Function Operation( _
        ByVal requestMsg As iShopMessages.GetTechnicianListRequestMessage _ ) As
        iShopMessages.GetTechnicianListResponseMessage
        Dim responseMsg As New iShopMessages.GetTechnicianListResponseMessage

        Me.PopulateExampleResult(responseMsg)
        responseMsg.Response = iShopMessages.iShopMessageResponse.Response_OK()
        Return responseMsg
End Function

Private Sub PopulateExampleResult( _
        ByVal responseMsg As iShopMessages.GetTechnicianListResponseMessage _ )
        Dim TechnicianList() As iShopMessages.CodeNameType
```

```
                ReDim TechnicianList(1)

                TechnicianList(0) = New iShopMessages.CodeNameType
                TechnicianList(0).Code = "bbrown"
                TechnicianList(0).Name = "Bob Brown"

                TechnicianList(1) = New iShopMessages.CodeNameType
                TechnicianList(1).Code = "jjones"
                TechnicianList(1).Name = "John Jones"

                responseMsg.TechnicianList = TechnicianList

                Return
        End Sub
```

**Other Files**

**File: iShopSmsServer.asmx**

This is the Microsoft ASP.NET file that represents the iShop Shop Management Server web service.

**UdpDiscoveryServer**

iShop specifies that a Shop Management Server is found by sending a UDP inquiry message over a network. As a web service application, the Shop Management Server itself will not listen for UDP requests. This folder contains a sample application that listens for UDP requests, and sends a UDP reply message back to the requestor identifying a Shop Management Server URL.

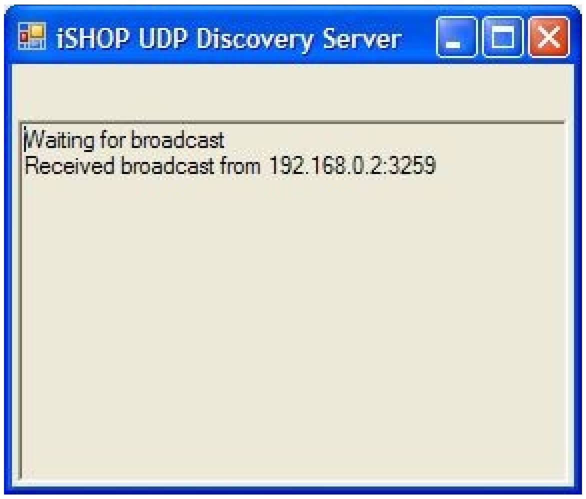
Documentation section *Finding an iShop Shop Management Server using UDP* contains background information on this subject. This documentation calls this type of application an iShop UDP Discovery Server.

The sample UDP Discovery Server application is a Microsoft .NET windows form application.

**Code Files**

**File: MainForm.vb**

This is the main form of the sample UDP Discovery Server application. The user interface simply displays an activity log. In the following example, a single UDP request message was received:



```
Waiting for broadcast
Received broadcast from 192.168.0.2:3259
```

**Figure 6 - UDP Discovery log**

The core functionality of this application is coded into this form although UDP functionality is used from framework object “iShopDiscoveryServerUdp.vb”.

**File: app.config**

This is the application configuration file for the sample UDP Discovery Server application.

Settings are read from this file using framework code “iShopConfigManager.vb”. Available settings are as follows:

**Table 17 - UDP Server Settings**

| **Setting Key** | **Description** |
|---|---|
| ServerName | The purpose of the UDP Discovery Server is to |
| ServerUrl | respond with the URL of an iShop server. These settings specify the URL value returned, and a name for the server represented by the URL. The server name is a text value for user consumption. |
| AAIA.iShop.Discovery.UdpGroupAddress | These are settings to framework code “iShopDiscoveryServerUdp.vb” (page 19). They are described in the documentation for that code. |
| AAIA.iShop.Discovery.ServerPort | |
| AAIA.iShop.Discovery.DefaultGroupTimeToLive | |

## XML

The XML folder contains files pertaining to the definition of XML messages that pass between an iShop client and iShop server application.

The types of files contained in this folder are summarized below, and fully described in the subsections that follow.

**Table 18 - XML Files**

| File Type | Description |
|---|---|
| XML Files: *.xml | These files are samples of the messages that can be passed between iShop client and iShop server applications. A sample is provided for each type of message. |
| XMLSpy Project Files: *.spp | XMLSpy by Altova Gmbh. is a leading interactive tool for viewing, editing and manipulating all types of XML files, including all the XML files deployed with iShop. |
| WSDL Files: *.wsdl | WSDL is an XML standard format for describing web services. As such, WSDL files can be and are used to describe the iShop web services. |
| Schema Files: *.xsd | Schema files are XML files that define valid content for other XML files. They are used in iShop to define what constitutes a valid XML message that can be passed in or out of any of the iShop web service operations. |

### Samples

The samples folder and its sub-folders contain examples of all the request and response messages that can pass between iShop client and iShop server applications. The examples are XML files, and are grouped into the following sub-folders:

**Table 19 - XML Sub Folders**

| Sub-folder | Description |
|---|---|
| IS | This folder contains a sample XML file for each request message to and response message from an iShop Information Server. The file is named after the message it represents, which is also the root element of the XML contained within. |
| Reused Elements | This folder contains sample iShop XML message files that demonstrate elements that are usable in both Information Server and Shop Management Server messages. |
| SMS | This folder contains a sample XML file for each request message to and response message from an iShop Shop Management Server. The file is named after the message it represents, which is also the root element of the XML contained within. |

**WSDL Files**

WSDL is an XML standard format for describing web services. As such, WSDL files can be and are used to describe the iShop web services.

WSDL is a W3C standard. Version 1.1 specifications are available on the web at http://www.w3.org/TR/wsdl. Version 2.0 candidate release specifications are available on the web at http://www.w3.org/TR/wsdl20/, however iShop does not use any such features.

WSDL files “iShopInformationServer.wsdl” and “iShopSMSServer.wsdl” define the iShop Information Server and Shop Management Server web services respectively, including all web service operations. The WSDL files name the web services “iShopInformationServer” and “iShopSMSServer” respectively.

Each operation accepts a single string input parameter, and returns a string result. These strings should hold valid XML request and response messages for the operation; however no message content validation is built into the WSDL. iShop Web service implementers are responsible for validating messages themselves using the iShop schema files – see XML Schema Files, below.

Note that message validation is built into the iShop sample code by using framework code “iShopXmlHelper.vb” to serialize and de-serialize between XML strings and instances of classes declared in “iShopMessages.vb”. Recall that file “iShopMessages.vb” is generated from the iShop schema files that define message content.

### 3.5.3 XML Schema Files

Schema files are XML files that define valid content for other XML files. They are used in iShop to define what constitutes a valid XML message that can be passed in or out of any of the iShop web service operations.

The following diagram illustrates the relationship between the iShop XML schema files:

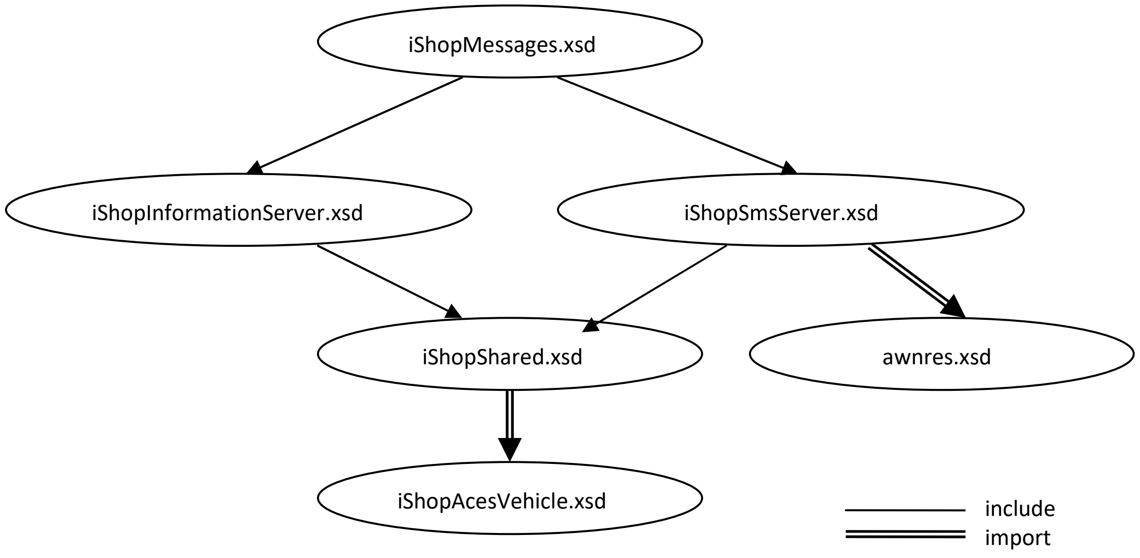


**Figure 7 - Schema file relationship**

The individual schema files are described further below.

**File: awnres.xsd**

This schema defines the ASANET/iShop standard for representing diagnostic results in messages between an iShop client application and an iShop Shop Management Server.

The ASANET/iShop standard is defined in document, “Definition for inspection results in asanetwork”, edition 3.0 dated “14.11.2002”. This document is presently available at: http://www.axonet.de/public_doc/xml_v30_en.pdf.

**File: iShopAcesVehicle.xsd**

This schema is used to define a vehicle using the ACES structure. It has been adapted from the “aces.xsd” schema distributed as part of the “ACES Delivery Specifications”, Version 1.07. The extent of deviations from the pure ACES schema as well as the reasons for doing so is documented in the iShop Functional Implementation Guide.

**File: iShopInformationServer.xsd**

This schema defines all the types used to define the iShop Information Server messages defined in schema file “iShopMessages.xsd”.

**File: iShopMessages.xsd**

This schema defines elements for each the message that can pass between iShop client applications and an iShop Information Server or an iShop Shop Management Server. Each

such element is defined in terms of a type that is itself defined in either “iShopInformationServer.xsd” or “iShopSMSServer.xsd”.

**File: iShopShared.xsd**

This schema defines types that are used in both iShop Information Server and iShop Shop Management Server messages.

**File: iShopSMSServer.xsd**

This schema defines all the types used to define the iShop Shop Management Server messages defined in schema file “iShopMessages.xsd”.

**Other Files**

**File: iShop.spp**

This is the XMLSpy project file that can be used to view all the files in the XML folder.

XMLSpy by Altova Gmbh. is a leading interactive tool for viewing, editing and manipulating all types of XML files. XMLSpy version 2005 was used to build this project file as well as to build and validate all the iShop XML files.

# Appendices

## Appendix A – Computing a MachineKey Value

MachineKey is a value used to enforce licensing of iShop Information Server applications. In particular, it is used to uniquely identify a computer so that the Information Server can verify that it is licensed to request information. Please refer to the iShop Version 3 Functional Implementation Guide document for background on how this value is used. This documentation section provides technical information for computing a value.

As described in the Functional Implementation Guide, the MachineKey value for a computer should be constructed by concatenating together all of the computer's MAC addresses, in ascending order alphabetically, and separating each address with a pipe symbol (|). All letter hexadecimal characters (A-F) must be uppercase.

For example, if your computer has 2 MAC addresses (all values in hex):

- 10.20.4D.6f.7A.BD
- 00.4A.6D.22.AA.43

You would compute the following MachineKey value:

00.4A.6D.22.AA.43|10.20.4D.6F.7A.BD

GetAdaptersInfo is the easiest Win32 API method for gathering MAC addresses. This is available on current recent versions of Microsoft Windows OS[4].

The following MFC C++ code construct MachineKey values as per the specification:

```
CString getMachineKey()
{
        // Define the objects which will hold the adapter information
        IP_ADAPTER_INFO *info = new IP_ADAPTER_INFO;
        unsigned long  bufferLength = sizeof(IP_ADAPTER_INFO);

        // Make sure we have a big enough buffer
        if (GetAdaptersInfo(info, &bufferLength) != ERROR_SUCCESS)
 {
```

4 GetAdaptersInfo is not available on Windows NT 4.0, however this version of Windows is no longer supported by Microsoft.

```
            delete info;
            info = reinterpret_cast<IP_ADAPTER_INFO *>
            (new char[bufferLength]);
    }

            // Now, get the real information
            if (GetAdaptersInfo(info, &bufferLength) != ERROR_SUCCESS)
    {
            AfxMessageBox("Unable to locate any adapters");
            delete info;
             return("");
    }

            // Loop through each adapter
            IP_ADAPTER_INFO *adapter = info;
            while (adapter)
            {
                        // Only do this for ethernet adapters
                        if (adapter->Type == MIB_IF_TYPE_ETHERNET)
                        {
                                    // TODO: Store the MAC addresses someplace
                                    // so they can be retrieved and concatenated
                                    // into a string with the addresses in
                                    // ascending order.
                        }

                        // And go on to the next adapter
                        adapter = adapter->Next;
             }
            delete info;

            // TODO: Put the strings together and return to combined string
    }
```

The following C# method does the same as above. This has a limitation in that it is built on the WMI functionality, which is not shipped with Win9x versions of Microsoft Windows[5]. In this case, however, WMI functionality is available from MicroSoft for download.

```
private void goButton_Click(object sender, System.EventArgs e)
{
        // Use the WMI system to get the MAC Addresses.  This means
        // that it won't work with Win9x OS's, unless the WMI
        // package has been installed on that computer.
        // Note:  In .NET 2.0, there is a native mechanism to do this.
        System.Management.ManagementObjectSearcher searcher =
                new System.Management.ManagementObjectSearcher
                ("Select * From Win32_NetworkAdapterConfiguration Where IpEnabled = true");
        System.Management.ManagementObjectCollection collection =     searcher.Get();
        if (collection != null)
        {
                // Loop through the collection to get each element
                foreach(System.Management.ManagementObject item in collection)
                {
                        // Get the MAC Address
                         System.String address = item["MACAddress"] as System.String;
                        if (address != null)
                        {
                                // TODO: Store the MAC addresses someplace
                                // so they can be retrieved and concatenated
                                // into a string with the addresses in
                                // ascending order.
                        }
                }
         }

// TODO: Put the strings together and return to combined string
}
```

5 Win9x versions of Microsoft Windows are no longer supported by Microsoft.