STAR Web Services

Quick Start Guide

2010v1

STAR Web Services: Quick Start Guide: 2010v1

Editor:
David Carver, STAR

Jason Loeffler, Karmak

Contributors:
Jonathan Wilson, Navistar

Russell Shephard, T-Systems

Daniel Hicks, Honda

Charlie Quirt, CIECA

Michelle Vidanes, STAR

Table of Contents

List of Figures

List of Examples

Revision History

Revision History		
Revision 2008v2	2008	NP
This is the first version of this document. As each subsequent document is published changes will be listed in this section.		
Revision 2009v1	March 2009	DAC
• Converted to DocBook format. • Split .NET and Java examples into separate chapters. • Added Revision History as its own section.		

Preface

II.I. Purpose

The purpose of this document is to provide a guide for quickly getting up and running with the STAR Web Services.

II.II. Scope

The STAR Web Services Quick-Start Guide contains a brief introduction to the STAR web services and the development tools that can be used to create them, as well as some implementation examples. This document is not intended to be a comprehensive design specification or development reference. It is only intended to give the reader the information necessary to gain a basic understanding of the STAR web services.

II.III. Audience

This document is intended for architects and web developers who are interested in implementing the STAR web services for STAR BOD document exchange. The document was designed for developers with existing knowledge of implementation practices of Web Services.

II.IV. STAR Organization

The goal of the Standards for Technology in Automotive Retail (STAR) organization is to develop and promote the use of voluntary information technology (IT) standards as a catalyst to fulfill the business information needs of dealers and manufacturers while reducing the time and effort previously required.

To accomplish this goal, STAR has developed a series of Special Interest Groups (SIGs) to address specific points of interest in the retail automotive industry. These SIGs, comprised of voluntary participants from member organizations, are chartered with:

1. Developing and maintaining standard messaging formats for dealer communications. STAR currently supports two formats:

2. Establishing a common messaging architecture for transmitting standard message formats promoting interoperability among retail system providers (RSPs) and original equipment manufacturers (OEMs) throughout the retail automotive industry. The Architecture SIG produces a series of documents referred to as the Transport Package that includes:

Chapter 1. Web Services

Table of Contents

1.1. Web Services Overview

The STAR Web Services transport was designed to provide a platform for secure and reliable delivery of any type of content in a standardized manner. The chosen architecture neither precludes nor requires attachments outside the body of the SOAP message for transportation of content. The chosen packaging methodology is well supported by all major Web Services toolkits and infrastructures and meets STAR's transport requirements.

1.2. Web Services Benefits

1. Web services do not require the adoption of a common platform, but adherence to the standard protocols. These protocols can be adhered to on most platforms.

2. Web services are very versatile by design.

3. Web services provide for code re-use in which the same web service can be used for several different clients.

4. Web services can incorporate failover functionality.

5. Web services provide easy integration with external data sources.

Chapter 2. Web Services Development Frameworks

Table of Contents

2.1. Overview

A development framework can be defined as a set of patterns, definitions and guidelines that assist how application development is performed. Many software development tool vendors provide development frameworks along with their products that include best practices for program design and sample code or code generators that automatically create commonly used code fragments.

This section contains an overview of some of the most popular development frameworks for web services. It can assist you in the selection of a framework for your STAR web services development activities.

2.2. Microsoft .NET

The Microsoft .NET Framework is a software component that can be added to or is included with Microsoft Windows operating system. It provides a large body of pre-coded solutions to common program requirements, and manages the execution of programs written specifically for the framework. The .NET Framework is a key Microsoft offering, and is intended to be used by most new applications created for the Windows platform. (wikipedia)

This is the first release of the .NET Framework and is also part of the first release of Microsoft Visual Studio .NET. The Microsoft .NET Framework is a software component that can be added to or is included with Microsoft Windows operating system. It provides a large body of pre-coded solutions to common program requirements, and manages the execution of programs written specifically for the framework.

The pre-coded solutions that form the framework's class library cover a large range of programming needs in areas including: user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. The functions of the class library are used by programmers who combine them with their own code to produce applications.

2.2.1. Web Service Enhancements

Web Service Enhancements (WSE) is a Visual Studio.Net and .Net Framework add-on created by Microsoft to support .Net development teams in the development of web services implementing protocols pertaining to WS-Specifications.

WSE version releases are triggered by changes in WS-Specifications and are independent of .Net Framework releases. The first two versions are compatible with .Net Framework 1.0/1.1 while the latest version can be deployed with .Net Framework 2.0 and Visual Studio 2005.

WSE 3.0 supports specifications such WS-Security 1.0/1.1, WS-Trust, WS-Secure Conversation, WS-Policy, WS-Addressing, WS-Reliable Messaging and MTOM, SOAP 1.1 and 1.2. It also includes capabilities to host web service outside IIS including console applications, windows services and forms and COM+ applications.

WSE Configuration Editor (WSE Settings 3.0 Tool) is a graphical user interface supported in Visual Studio 2005. It enables user to specify WSE settings such as security, routing, policy, token issuing, diagnostics and messaging rather than editing configuration file directly.

2.3. Java

Java is an extremely prolific language and there are a number of web services frameworks to choose from. Each framework fits a particular need for implementing web services. The wide variety of frameworks is both an advantage and a disadvantage for those trying to choose what framework to use. The advantage is that the developer has a large selection of frameworks and features from which to choose. The disadvantage is that it can be difficult to determine which framework fits a project's particular needs. This section will briefly describe the most popular frameworks, and where one can find more detailed information regarding each.

2.3.1. Java Web Services

The Java Web Services Developer's Pack (JWSDP) is a framework stack maintained at the Sun Developer's Network (Sun). Its purpose is to provide the necessary tools and frameworks for implementing secure and reliable web services using the java runtime. The frameworks included are:

- JAX-WS – Implements the base Web Services specifications, such as, SOAP, WS-I Basic Profile 1.1, WS-I Attachments Profile 1.0, WS-I Simple SOAP Binding Profile 1.0, and WS-Addressing 1.0

- JAXB – This is a data binding framework for XML. It creates Java class objects for reading and writing XML files.

- WSIT – Extends the JAX-WS framework, so that it web services created with JAX-WS can interoperability with the Windows Communication Framework. It uses both JAXB and JAX-WS.

- JAXP – A standardized API implemented by many XML parsers used for reading and writing XML files using either SAX or W3C DOM.

- XWSS - XML and Web Services Security is a framework for implementing the various WS-Security specifications. This includes SAML, X509a, and Username and Password.

Unlike the Apache Axis 2 framework which allows for plugging in your own data banding framework, the Web Services Developer pack only directly supports the JAXB framework. It may be possible to get the other frameworks to work with it but that is beyond the scope of this document.

2.3.2. Apache Axis 1.4

Apache Axis 1.4 is the redesign and re-implementation of the Apache SOAP framework. It is widely used and provides support for both a wide variety of SOAP design patterns. Its sole goal is to implement support for the World Wide Web Consortium's SOAP specification. It is still widely used but is also limited in its support for more current Web Services specifications.

2.3.3. Apache Axis 2

Apache Axis 2 is a redesign of the Apache Axis 1.4 framework. It has been redesigned to allow a plug-in architecture to support many of the new web services specifications. Apache Axis 2 also allows you to choose and implement a variety of XML Data Binding frameworks instead of being locked into one particular framework. The following are some of the plug-ins that extend Apache Axis 2:

- Apache Sandesha2 – Implements the WS-ReliableMessaging specification.
- Apache Kandula2 – Implements the WS-Coordination specification.
- Apache Rampart – Implements the WS-Security specification including the WS-SecurityPolicy specification. Like the XWSS framework, this implements support for XML Security, Username and Password, and XML Signature.
- WS- Addressing support is built into the Apache Axis 2.
- Apache WSS4J – Implements the OASIS Web Services Security Specification.

Apache Axis 2 is where the new development is heading at the Apache Foundation for web services. Its plug-in architecture allows it to easily evolve and adapt to the changing web services specifications (apache).

2.3.4. XML RPC

XML RPC was one of the first specifications implemented. Both Sun and Apache have implementations of the specification, and XML RPC is supported on a wide variety of platforms. However, even though XML RPC is still widely used, it is not recommended for STAR implementations as it doesn't have support for many of the security related features STAR Web Services require. The specification can be found at the XML RPC homepage (xmlrpc).

Chapter 3. Web Services Development Tools

Table of Contents

3.1. .NET Tools

This is the first release of the .NET Framework and is also part of the first release of Microsoft Visual Studio .NET. The Microsoft .NET Framework is a software component that can be added to or is included with Microsoft Windows operating system. It provides a large body of pre-coded solutions to common program requirements, and manages the execution of programs written specifically for the framework.

The pre-coded solutions that form the framework's class library cover a large range of programming needs in areas including: user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. The functions of the class library are used by programmers who combine them with their own code to produce applications.

Probably the most widely used Microsoft.Net tool is the Microsoft Visual Studio.Net development software (IDE). The IDE provides specific goal oriented project types and visual functionality, which allow the developer to take full advantage of the .Net Framework. Example project types include ASP.Net Web Service, ASP.Net Web Application, and Windows Application. The ASP.Net Web Service project type provides basic skeleton code of a web service, so the developer need only fill in the details. The Windows Application and ASP.Net Web Application project types provide graphical functionality for the developer to (for instance) set a “web reference” to a web service, which automatically creates a proxy of the target web service. This allows the developer to quickly write code to access and communicate with the target web service.

3.2. Java Tools

In addition to the various frameworks for Java Web Service development, Java also has many development environments implementing these frameworks. These tools range from open source to commercially available tools. Each has its advantages and disadvantages, the tool one chooses depends on many factors and in particular the environment in which one is targeting their web service implementation.

Eclipse – Web Tools Platform. The Eclipse Foundation's Web Tools Project includes a wide variety of added functionality for working with Web Services implementations. It includes built in support for working with Apache Axis 1.4 and Apache Axis 2 frameworks. It also includes a basic set of tools for working with XML, XSD, and WSDL files. In addition, it includes a Web Services Explorer for testing and debugging Web Services either developed with the tool or hosted on another system. The Eclipse Web Tools Platform is a free set of plug-ins available from the eclipse website (eclipse). Additional functionality such as UML, BPEL and other tools can be downloaded from the Eclipse website.

Netbeans – Enterprise Pack. Netbeans is one of the first Java IDEs on the market is developed with Sun Microsystems as an open source project. The Enterprise Pack add-in includes support for working with the Java Web Services Developer's Pack. It also includes several base XML, XSD, and WSDL editing tools. It also includes a BPEL designer and UML modeling tools.

Rational Software Architect. IBM has built Rational Software Architect on top of the Eclipse Platform and it includes the Eclipse Web Tools Platform as well. In addition, IBM has included tools specific for working with its IBM Websphere application server, and UML Modeling tools.

Weblogic Workshop. BEA has migrated their Weblogic Worshop IDE to the Eclipse platform and the Eclipse Web Tools Platform plug-ins. In addition, they have extended this base platform to include tools for working with XML Beans, enhanced support for a variety of web services frameworks, and the BEA Weblogic application server.

JBuilder 2007. J-Builder 2007 is CodeGear's, formerly Borland, Java IDE. It has been migrated to using the Eclipse Platform and Eclipse Web Tools Platform. It also has extended the functionality by including both an EJB and Web Services visual designer.

3.3. Cross-platform

XML Spy Professional. XMLSpy™ 2007 by Altova supports development in a wide variety of Web services platforms including Microsoft .Net, J2EE, and Eclipse.

Stylus Studio. Stylus Studio™ 2007 XML Enterprise Suite, Release 2 by DataDirect Technologies utilizes an open Web service framework architecture that fully supports frameworks such as Microsoft SOAP, Microsoft .Net and the open source Web service framework, Apache Axis.

Liquid XML Data Binder. Liquid XML Data Binder™ is a multi-platform data binding framework for use with XML Schema. It will generate data objects for C#, VB.NET, JAVA, VB6, and C+. It supports both DTD and W3C XML Schemas. Source code is available for the libraries, allowing for recompiling the base application on almost any platform. Liquid XML Data Binder is a product of Liquid Technologies.

Chapter 4. Web Services Testing Tools

Table of Contents

4.1. Overview

Most toolsets and integrated development environments include some basic testing for the web services that are developed. Typically this includes a tool that can read an existing WSDL, and communicate to a server that has already been established. Other tools allow for the complete testing of a web service from both the server side and the client side. The tools that follow are only a few of the tools that are available to users, and are a few that STAR members are using to develop and test their Web Services.

4.2. SOAPUI

SOAPUI is a java based tool for testing and working with any type of Web Service. All that is needed is a WSDL file whether stored locally or remotely. The program will read the WSDL and generate some mock code that can emulate a server or act as a client. SOAPUI comes in two versions. An open source version that is free, and provides most of the features one will need for testing Web Services. It also comes in a Professional version which extends the testing suite ability and provides some further advanced features for running test suites.

4.3. OxygenXML

OxygenXML is a full featured XML Integrated Development Environment for working with a wide variety of XML related technologies. It includes a set of programs for working with and testing WSDL files. The WSDL Analyzer allows you to use an existing WSDL as a client application, and test against a Server instance. The basic soap envelope and payloads are generated, and available for editing. The corresponding results received back from the service are available for editing and viewing later. OxygenXML is a commercial application, but is available for a 30 day trial.

4.4. XML Spy Enterprise

XML Spy Enterprise Edition 2008 includes a WSDL Graphical Editor, as well as a SOAP Client and SOAP Debugger. The latter two allow for interactive debugging of the SOAP messages sent and received from a web service. It has the ability to step through the service, set break points, and set conditional break points based on an XPATH expression.

Chapter 5. .NET Examples

Table of Contents

5.1. Obtaining the STAR WSDLs

This section will provide the reader with STAR web service implementation examples for a number of the development frameworks. The implementation examples have been contributed by the STAR members that are currently implementing the web services. Currently there are only a few implementation examples available. As STAR receives examples for additional development frameworks, they will be added to future releases of the STAR Web Services Quick-Start guide.

5.2. Scope and Limitation

The scope of this section is the construction of a basic asp.net web service generated from the STAR WSDL. A windows client application will be created to simulate a user calling and receiving response from the web service using the ProcessMessage web method.

Microsoft WSE related implementations of WS-Security and WS-Addressing and other STAR requirements and/or recommendations are not implemented in this guide.

5.3. Objectives

- Create an ASP.Net Web Service based on a class generated from the STAR WSDL.
- Create a windows form application that calls the web service.
- Client application will call the Web Service’s ProcessMessage function.
- Web service will return the payload to the client with a different ContentID to simulate processing.

5.4. Prerequisites

- A copy of the WSDL from the Star Schema Repository (Schema Version 4.2.4 is used for this example).

- Installation of Microsoft Visual Studio.Net 2003.
- .Net Framework 1.1
- 7Zip or similar program able to handle zip files.

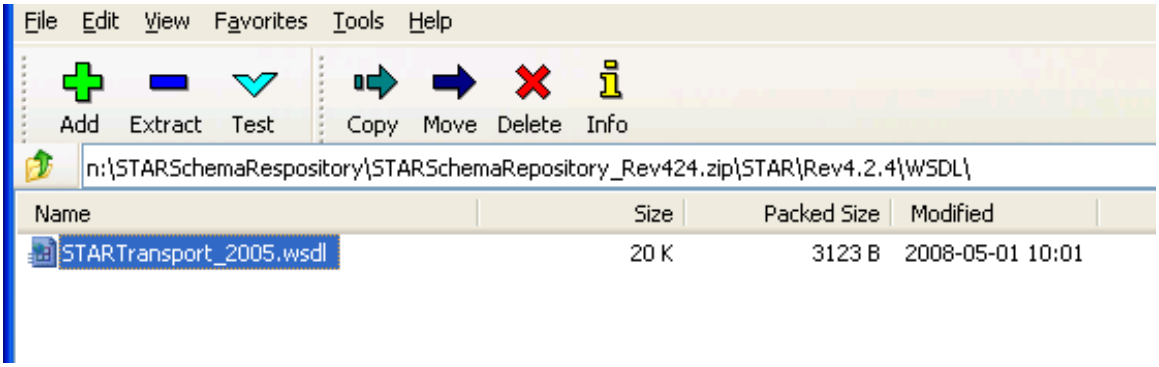
5.5. .NET Walkthrough

5.5.1. .NET Service

Procedure 5.1. Generating Web Service Class code from STAR WSDL

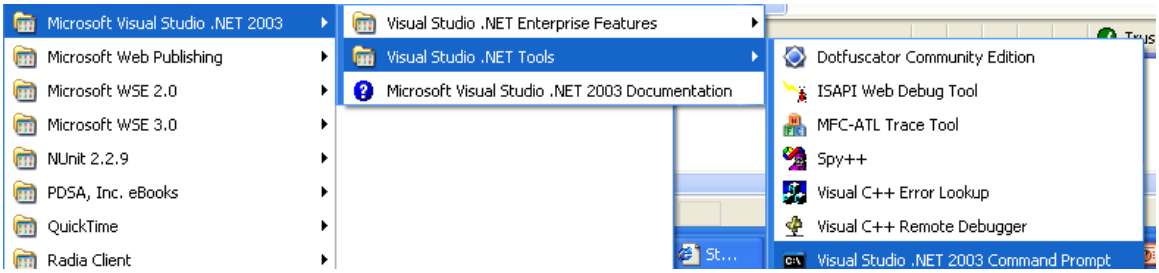
1. Retrieve WSDL from Star Schema Repository and extract the StarTransport_2005.wsdl file.

   Figure 5.1. WSDL Extract

2. Open Visual Studio 2003 command prompt.

   Figure 5.2. Visual Studio 2003 Command Prompt

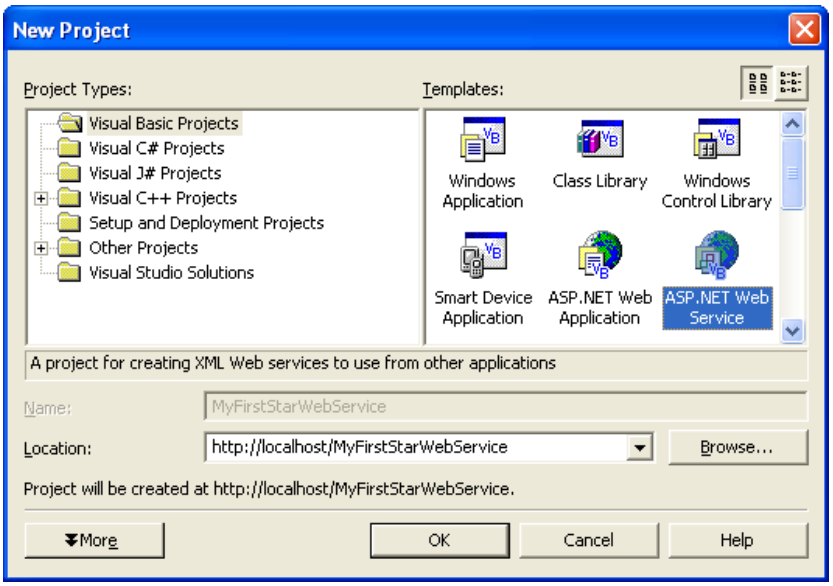
3. Run wsdl.exe against the extracted WSDL file. By default WSDL generated codes are generated in C# language. You can optionally specify the target language to 'VB' (Visual Basic) by using the /language parameter. Make sure to execute this command where you have extracted the STAR WSDL.

```
wsdl startransport_2005.wsdl /language:VB
```

Procedure 5.2. Creating the ASP.Net Web Service

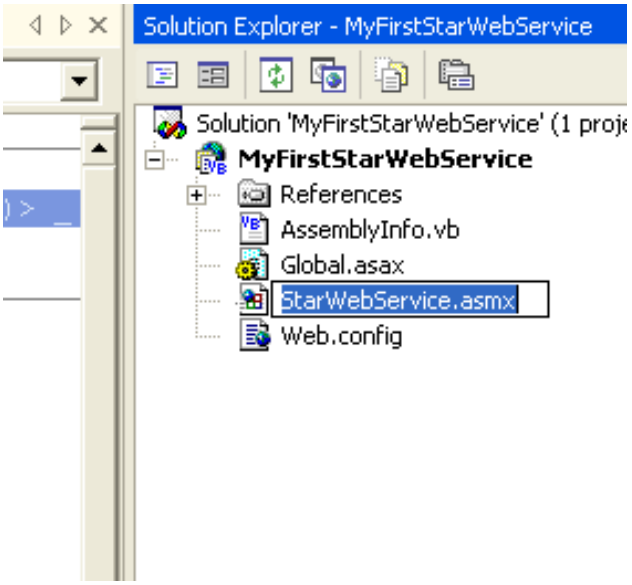
1. Open up the Visual Studio .Net 2003 and create a new ASP.Net Web Service Project.

Figure 5.3. Create ASP.NET Web Service



2. Rename the Service1.asmx file to StarWebService.asmx.

Figure 5.4. Rename Service1.asmx



3. Copy contents of generated GenericWebService.vb to the code behind of your new web service page.

4. Rename the Class name to StarWebService

Figure 5.5. Rename StarWebService

```
Public Class StarWebService 'was previously Public Class GenericService
    Inherits System.Web.Services.Protocols.SoapHttpClientProtocol
```

5. Set the URL in the class constructor to the actual URL of the web service.

Figure 5.6. Set URL for Web Service

```
'<remarks/>
Public Sub New()
    MyBase.New()
    'Me.Url = "http://127.0.0.1:8088/mockstarTransportPortTypes" 'this was the original
    Me.Url = "http://localhost/MyFirstStarWebService.asmx/StarWebService.asmx" 'this is your adjusted URL
End Sub
```

6. Add a WebMethod attribute for each of the STAR WebMethods.

Figure 5.7. Add Web Method to ProcessMessage

```
    '<remarks/>
    <System.Web.Services.Protocols.SoapHeaderAttribute("payloadManifest", Direction:=
    System.Web.Services.WebMethodAttribute(), _
     System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://www.starstanda
"age", RequestNamespace:="http://www.starstandards.org/webservices/2005/10/transport"
    Public Sub ProcessMessage(ByRef payload As Payload)
        Dim results() As Object = Me.Invoke("ProcessMessage", New Object() {payload})
        payload = CType(results(0), Payload)
    End Sub
```

Figure 5.8. Add Web Method to Pull Message

```
    '<remarks/>
    <System.Web.Services.Protocols.SoapHeaderAttribute("payloadManifest", Directio
    System.Web.Services.WebMethodAttribute(), _
     System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://www.starsta
"", RequestNamespace:="http://www.starstandards.org/webservices/2005/10/transport"
    Public Function PullMessage() As <System.Xml.Serialization.XmlElementAttribute

        Dim results() As Object = Me.Invoke("PullMessage", New Object(-1) {})
        Return CType(results(0), Payload)
    End Function
```

Figure 5.9. Add Web Method to Put Message

```
<System.Web.Services.Protocols.SoapHeaderAttribute("payloadManifest"),
System.Web.Services.WebMethodAttribute(), _
 System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://www
Public Sub PutMessage(ByVal payload As Payload)
    Me.Invoke("PutMessage", New Object() {payload})
End Sub
```

Public Dim payload As MyFirstStarWebServi

7. Replace the ProcessMessage function's code with the following line:

Figure 5.10. Update Process Message

```
'<remarks/>
<System.Web.Services.Protocols.SoapHeaderAttribute("
System.Web.Services.WebMethodAttribute(), _
 System.Web.Services.Protocols.SoapDocumentMethodAtt
ge", RequestNamespace:="http://www.starstandards.org/w
Public Sub ProcessMessage(ByRef payload As Payload)
    payload = CreateResponsePayload(payload)
End Sub
```

8. Insert the following function anywhere inside the class code.

Figure 5.11. Add Playload Response

```
Public Function CreateResponsePayload(ByVal inputPayload As Payload) As Payload
    'Inside a code like this, the implementer should do the following:
    'a) Parse the contents of the payload.
    'b) Process the transactions to their backend as applicable.
    'c) Respond by creating a response payload to be returned to caller.
    inputPayload.content(0).id = Guid.NewGuid.ToString
    Return inputPayload
End Function
```

9. Modify the ASMX file to refence the StarWebService Class.

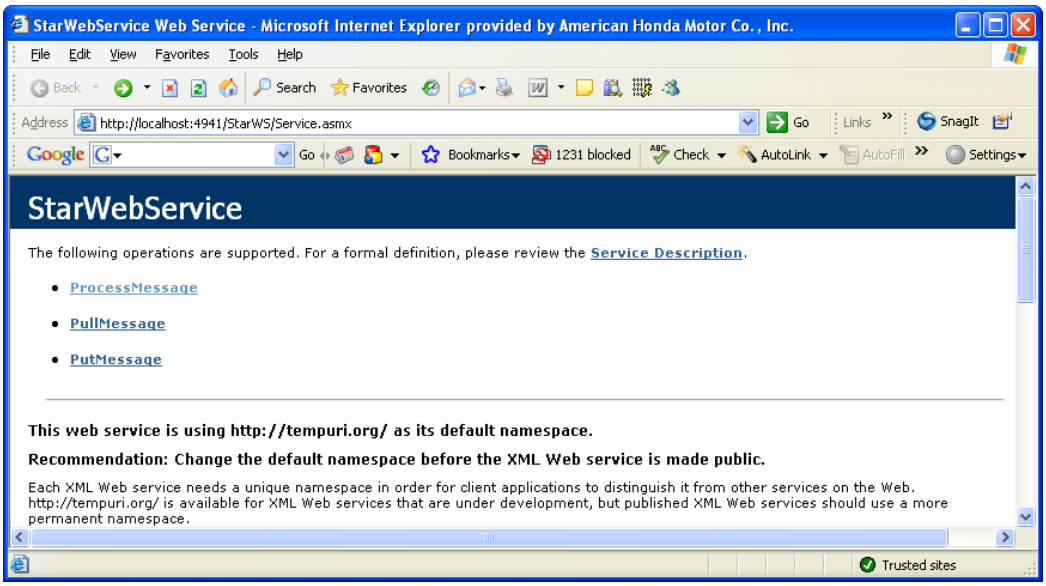
Figure 5.12. Modify ASMX Web Service

ct Browser | Start Page | StarWebService.asmx.vb [Design]* | StarWebService.asmx.vb* **StarWebService.asmx*** |

```
<%@ WebService Language="vb" Codebehind="StarWebService.asmx.vb" Class="MyFirstStarWebService.StarWebService"
```

10. Compile the code and run. You should now have a web service that has all three STAR Methods.

Figure 5.13. Deployed Web Service



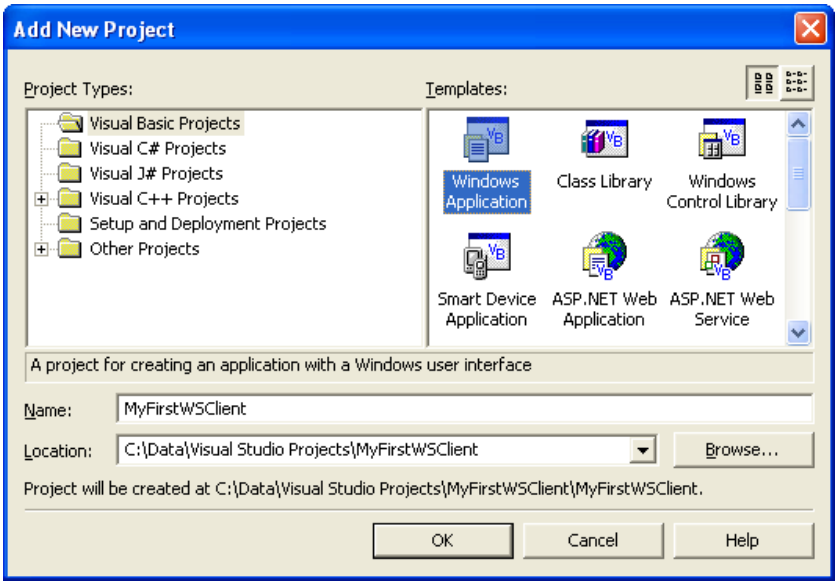
5.5.2. .NET Client

In this section, we will be creating a windows form application that calls the Web Service's ProcessMessage method.

Procedure 5.3. Creating Web Service Client

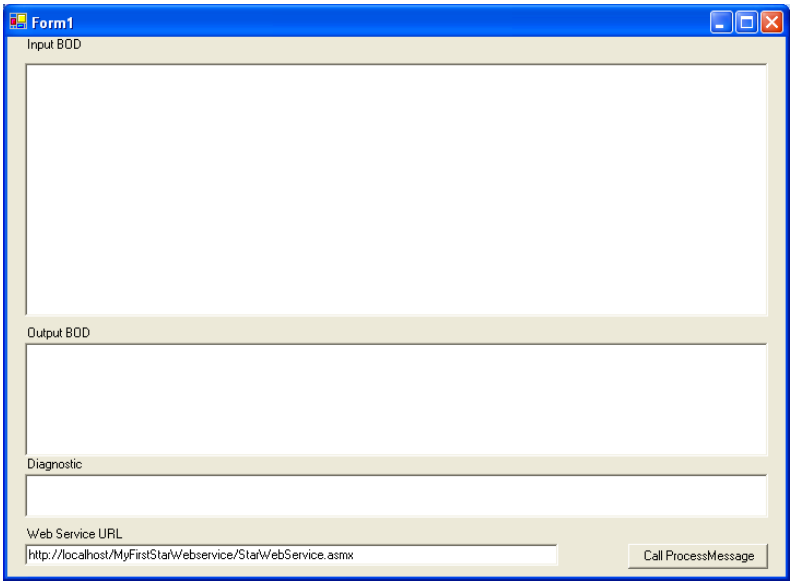
1. Create a new windows application project.

Figure 5.14. Create .NET Client



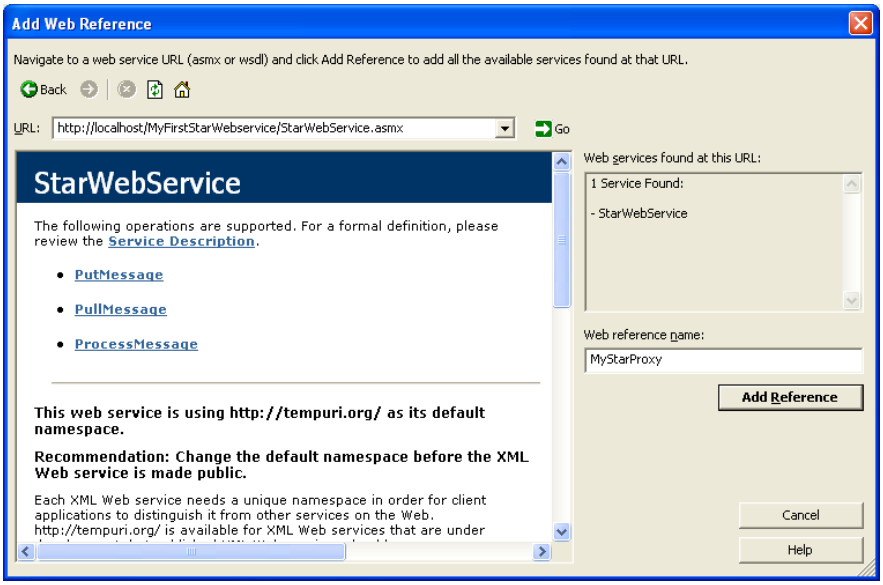
2. Create a Windows form that has the following layout:

Figure 5.15. Create Windows Form



3. Add a web reference to the star web service you've previously created.

Figure 5.16. Add Web Reference



4. Create a function that would process the request BOD.

Figure 5.17. Create Function Process Request

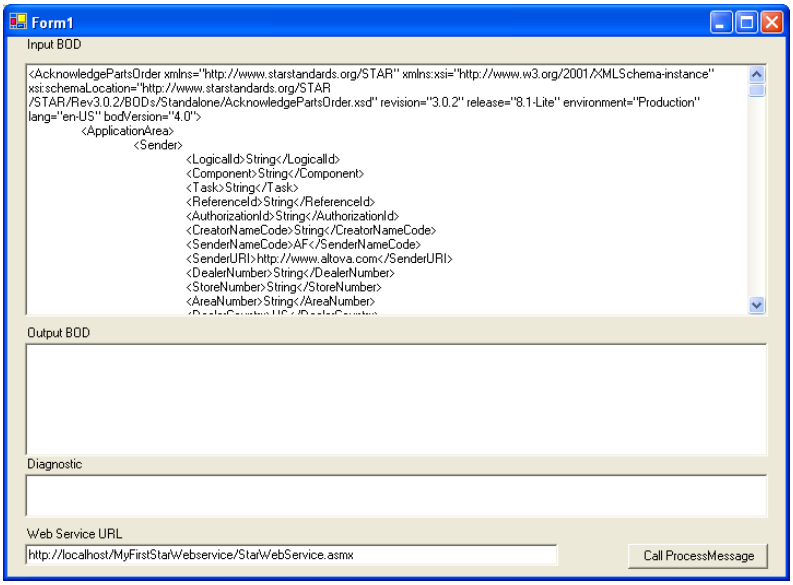
```
Public Function ProcessRequest()
    Dim host As New MyStarProxy.StarWebService
    Dim inputDoc As New Xml.XmlDocument
    inputDoc.LoadXml(txtInputBOD.Text)
    Dim xmlBod As Xml.XmlElement = inputDoc.DocumentElement

    Dim content As MyStarProxy.Content = New MyStarProxy.Content
    Dim inputGuid As String = Guid.NewGuid.ToString

    content.Any = xmlBod
    content.id = inputGuid

    Dim payload As MyStarProxy.Payload = New MyStarProxy.Payload
    payload.content = New MyStarProxy.Content() {content}

    Dim proxy As MyStarProxy.StarWebService = New MyStarProxy.StarWebService
    proxy.Url = txtURL.Text

    ' Set the Manifest SOAP header
    proxy.payloadManifest = New MyStarProxy.PayloadManifest
    proxy.payloadManifest.manifest = New MyStarProxy.Manifest() {New MyStarProxy.Manifest}
    proxy.payloadManifest.manifest(0).element = xmlBod.LocalName
    proxy.payloadManifest.manifest(0).namespaceURI = xmlBod.NamespaceURI
    proxy.payloadManifest.manifest(0).contentID = inputGuid

    proxy.ProcessMessage(payload)

    ' Read the return payload back to the client
    Dim elementBod As XmlElement = DirectCast(payload.content(0).Any, XmlElement)
    txtOutputBOD.Text = elementBod.OuterXml                                    Class XmlElement

    ' Read the Input and Output ContentID
    txtDiags.Lines(0) = "Input BOD Content ID:" + inputGuid
    txtDiags.Lines(1) = "Output BOD Content ID:" + payload.content(0).id

End Function
```

5. Recompile the Client Code.

Procedure 5.4. Running the Client and Web Service Code

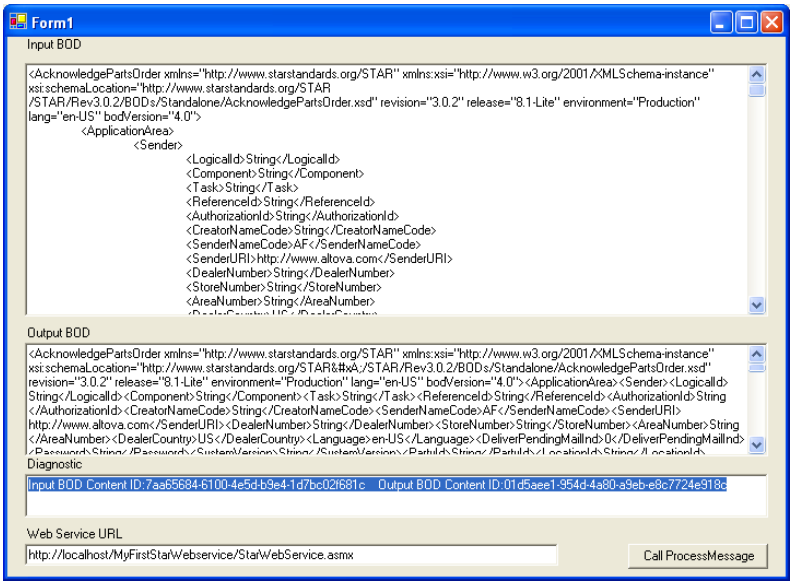
1. Run the windows application code and copy any STAR BOD examples from the Repository's example folder to the Input BOD text box.

Figure 5.18. Copy sample BOD



2. Click on the Submit button and examine the output BOD and Content IDs in the Diagnostic Textbox.

Figure 5.19. Client Results



Chapter 6. Java Examples

Table of Contents

6.1. JAX-WS 2.x

This example uses Sun's JAX-WS implementation to create a Web Service and client. The JAX-WS implementation is part of the Metro Web Service framework, a subset of Sun's Project GlassFish and a continuation of the work Sun packaged as the Java Web Services Developer Pack (JWSDP). The homepage for Metro is https://metro.dev.java.net/ . The target runtime for the service is the Apache Tomcat Server. The client will be a standalone application.

This example was created using the JDK 5.0 Update 14, JAX-WS 2.1.2, and Tomcat 5.5. While some of the screen shots will show Eclipse 3.3, Eclipse is not required for this example. Eclipse was primarily used to give a visual representation of the WAR, but it was used in some instances for source editing.

6.1.1. Prerequisites

1. JAX-WS can be downloaded from the JAX-WS homepage at https://jax-ws.dev.java.net/

2. Since our target runtime is Tomcat, refer to the Tomcat homepage (http://tomcat.apache.org/) for download and installation instructions.

3. Both service and client will be generated using the wsimport tool included with JAX-WS. This tool can be found in <JAX-WS Install Directory>\jaxws-ri\bin\wsimport.bat. A document describing the tool is available at <JAX-WS Install Directory>\jaxws-ri\docs\wsimport.html.

6.1.2. JAX-WS Walkthrough

Procedure 6.1. Add Address Information to the WSDL

1. If the WSDL was retrieved from STAR repository, the service information in the WSDL is incomplete. It must be changed to reflect your Web Service implementation. An unmodified WS-

DL will have an XML comment beginning with “Note to implementors…” along with an example of what the address information should look like. The example below is based off of the ProcessRetailDeliveryReporting.wsdl in Rev5.2.1. Assuming you are running a local instance of Tomcat, with 8080 as your HTTP port, the service information will look like the following:

Example 6.1. Sample Service

```
<wsdl:service name="ProcessRetailDeliveryReportingWebService">
  <wsdl:port name="ProcessRetailDeliveryReportingStarTransport" binding="starbindings:starTransport">
    <soap:address location="http://localhost:8080/StarQuickStartService/StarTransport/"/>
  </wsdl:port>
</wsdl:service>
```

Once the WSDL has been edited with the address information, it can be used by wsimport to produce source for the service and client, as well as classes for JAXB marshalling and unmarshalling.

2. At a windows command prompt, move to the <JAX-WS Install Directory>\jaxws-ri\bin directory. Execute the wsimport script. Be sure to set the output location for the source (-s) since the default behavior is to produce compiled code. This location must already exist since the script will not create the directory for you. It will give an error, “directory not found…” if the directory does not exist. Also, if you run into memory errors, edit the wsimport.bat script to increase the Java heap size. For this example the initial and maximum values were increased to 512MB, so the launch section of the wsimport.bat script looked like this (edited section in bold):

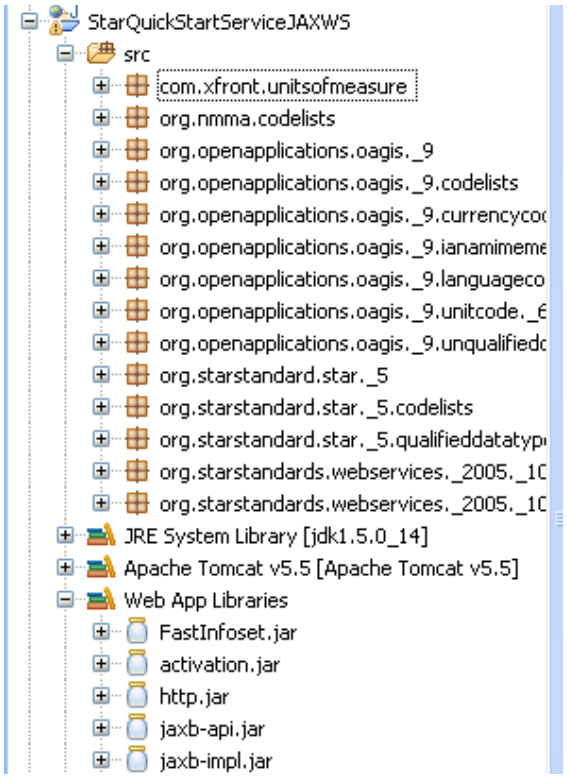
```
%JAVA% %WSIMPORT_OPTS% -Xms512M -Xmx512M -jar "%JAXWS_HOME%\lib\jaxws-tools.jar" %*
```

Example 6.2. Example WSImport command

```
wsimport -d classes -s src -verbose C:\JAX-WS\jaxws-ri\bin\STAR\Rev5.1.2\WSDL\ProcessRetailDeliveryReporting.wsdl > results.txt
```

3. Import the generated source into a WAR, adding the necessary jar files from <JAX-WS Install Directory>\jaxws-ri\lib

Figure 6.1. Import Source



4. Create the Web Service implementation class. This class can implement the StarTransportPortTypes interface that was created by wsimport. It is not required that it implement the interface, though it may help if using an IDE since the IDE can generate the inherited abstract methods. For this example, I called my class ProcessRetailDeliveryReportingWebServiceImpl. Once this class is created, add an annotation above the class declaration to refer to the endpointInterface:

```
@javax.jws.WebService (endpointInterface="org.starstandards.webservices._2005._10.transport.bindings.StarTransportPortTypes")
```

5. For now, only add print statements to the implementation class so you can tell when you have called the method via the client that we will soon create. Below is how your ProcessRetailDeliverWebServiceImpl class should currently look:

```
package org.starstandards.webservices._2005._10.transport.bindings;

import org.starstandards.webservices._2005._10.transport.AcknowledgeRetailDeliveryReportingPayload;
import org.starstandards.webservices._2005._10.transport.ProcessRetailDeliveryReportingPayload;

@javax.jws.WebService (endpointInterface="org.starstandards.webservices._2005._10.transport.bindings.StarTransportPortTypes")
public class ProcessRetailDeliveryReportingWebServiceImpl implements
        StarTransportPortTypes {

    public AcknowledgeRetailDeliveryReportingPayload processMessage(
            ProcessRetailDeliveryReportingPayload payload) {
        System.out.println("Called processMessage");
        return null;
    }

    @SuppressWarnings("unchecked")
    public AcknowledgeRetailDeliveryReportingPayload pullMessage()
    {
       System.out.println("Called pullMessage!");
       return null;
    }

    public void putMessage(ProcessRetailDeliveryReportingPayload payload) {
        System.out.println("Called putMessage");
    }
}
```

Before our service is complete, we need to configure the application so that a Web Service call will be routed to the implementation class we just created. This is done by editing the web.xml and creating a JAX-WS configuration file, sun-jaxws.xml.

Three items need to be added to the web.xml:

1. A listener reference to the JAX-WS WSServletContextListener
2. A servlet reference to the JAX-WS WSServlet servlet
3. A servlet-mapping reference for the above servlet

For this example, here are those following items:

```
<listener>
  <listener-class>com.sun.xml.ws.transport.http.servlet.WSServletContextListener</listener-class>
</listener>
<servlet>
  <servlet-name>StarTransportPort</servlet-name>
  <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>StarTransportPort</servlet-name>
  <url-pattern>/StarTransport</url-pattern>
</servlet-mapping>
```

Procedure 6.2. Create the sun-jaxws.xml

1. Now it’s time to create the sun-jaxws.xml. This file resides in the same location as the web.xml, the WEB-INF folder. Below is the content of the file for this example:

Example 6.3. sun-jaxws.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">
  <endpoint
      name="starTransportPortTypes"
      interface="org.starstandards.webservices._2005._10.transport.bindings.StarTransportPortTypes"
      implementation="org.starstandards.webservices._2005._10.transport.bindings.ProcessRetailDeliveryReportingWebServiceImpl"
      url-pattern="/StarTransport"
 />
</endpoints>
```

Note

Note that the endpoint name is the same as the name that you see on the Java annotation from the StarTransportPortTypes class. Also, the interface and implementation classes are provided. The url-pattern provided must match what is in the web.xml file.

2. The service is now ready to deploy. Deploy the WAR to Tomcat or the application server you are working with. You can confirm installation by accessing the service's URL. For this example, the service can be accessed at http://localhost:8080/StarQuickStartServiceJAXWS/StarTransport [http://localhost:8080/StarQuickStartServiceJAXWS/StarTransport] in a browser. The following should be returned:

Figure 6.2. JAX-WS Deploy Endpoint

Procedure 6.3. Create JAX-WS Client

1. Now it is time to create the client. Since the client uses the same source files generated by wsimport that were used for the service, the only item to create is a new class and method that will use the generated code to call the service. Below is an example of a test client that calls the PullMessage method. Since there is currently no implementation code in our service, we will at least see the print statement ("Called pullMessage") from the service printed to the standard output of the application server. Until implementation code is added to the service to return a valid response, this client will throw an exception.

```
package org.star.quickstart.client;

import org.openapplications.oagis._9.unqualifieddatatypes._1.IdentifierType;
import org.starstandard.star._5.AcknowledgeRetailDeliveryReportingType;
import org.starstandard.star._5.ApplicationAreaType;
import org.starstandards.webservices._2005._10.transport.AcknowledgeRetailDeliveryReportingContent;
import org.starstandards.webservices._2005._10.transport.AcknowledgeRetailDeliveryReportingPayload;
import org.starstandards.webservices._2005._10.transport.bindings.ProcessRetailDeliveryReportingWebService;
import org.starstandards.webservices._2005._10.transport.bindings.StarTransportPortTypes;

public class TestClient {
   public static void main(String[] args)
   {
      ProcessRetailDeliveryReportingWebService stub = new ProcessRetailDeliveryReportingWebService();
      StarTransportPortTypes port = stub.getProcessRetailDeliveryReportingStarTransport();
         ((javax.xml.ws.BindingProvider)port).getRequestContext().put(javax.xml.ws.BindingProvider.ENDPOINT_ADDRESS_PROPERTY, "http://localhost:8080/StarQuickStar
      AcknowledgeRetailDeliveryReportingPayload payload = port.pullMessage();
      AcknowledgeRetailDeliveryReportingContent content = payload.getContent().get(0);
      AcknowledgeRetailDeliveryReportingType bod = content.getAcknowledgeRetailDeliveryReporting();
      ApplicationAreaType applicationArea = bod.getApplicationArea();
      IdentifierType BODID = applicationArea.getBODID();
      System.out.println("BODID returned: " + BODID.getValue());
   }
}
```

2. Once you have created the TestClient class, execute the main method and verify that “pullMessage” has been written to the application server logs by the service.

3. Now that we have verified that the client can communicate to the service, we can add some JAXB unmarshalling code to service to simulate producing a valid response. Similarly, we can add marshalling code to the client to print out the AcknowledgeRetailDeliveryReporting BOD returned from the service.

   Below is the code added to the pullMessage method of the service implementation class. In this example, the BOD is being read straight from the “examples” directory of a STAR repository. Using the JAXB unmarshaller, the xml data from the file is placed into the Java objects reference by the AcknowledgeRetailDeliveryReportingType class.

```
@SuppressWarnings("unchecked")
   public AcknowledgeRetailDeliveryReportingPayload pullMessage()
   {
       System.out.println("Called pullMessage!");
       AcknowledgeRetailDeliveryReportingPayload response = new AcknowledgeRetailDeliveryReportingPayload();
       AcknowledgeRetailDeliveryReportingContent content = new AcknowledgeRetailDeliveryReportingContent();

       try
       {
         JAXBContext jaxbContext = JAXBContext.newInstance("org.starstandard.star._5");
         Unmarshaller unmarshaller = jaxbContext.createUnmarshaller();
         JAXBElement<AcknowledgeRetailDeliveryReportingType> bod =
            (JAXBElement<AcknowledgeRetailDeliveryReportingType>) unmarshaller.unmarshal(new File("C:\\jaxws-ri\\bin\\STAR\\Rev5.2.1\\BODExamples\\Ackn

         content.setAcknowledgeRetailDeliveryReporting(bod.getValue());
       }
        catch(Exception e)
        {
          e.printStackTrace();
        }

      response.getContent().add(content);

      return response;
}
```

4. Before adding the marshalling code to the client, deploy the service again with the updated implementation code from above. Invoke the service with the TestClient created earlier. If successful, the BOD Id will be printed.

   Below is the code added to the TestClient to take the results from the PullMessage call and print the resulting XML to the standard output stream. The JAXB marshaller is used to get the XML content from the Java object.

```
package org.star.quickstart.client;

import java.io.StringWriter;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.Marshaller;

import org.starstandard.star._5.AcknowledgeRetailDeliveryReportingType;
import org.starstandards.webservices._2005._10.transport.AcknowledgeRetailDeliveryReportingContent;
import org.starstandards.webservices._2005._10.transport.AcknowledgeRetailDeliveryReportingPayload;
import org.starstandards.webservices._2005._10.transport.bindings.ProcessRetailDeliveryReportingWebService;
import org.starstandards.webservices._2005._10.transport.bindings.StarTransportPortTypes;

public class TestClient {

   public static void main(String[] args)
   {
     ProcessRetailDeliveryReportingWebService stub = new ProcessRetailDeliveryReportingWebService();
     StarTransportPortTypes port = stub.getProcessRetailDeliveryReportingStarTransport();
        ((javax.xml.ws.BindingProvider)port).getRequestContext().put(javax.xml.ws.BindingProvider.ENDPOINT_ADDRESS_PROPERTY, "http://localhost:8080/StarQuickStar
     AcknowledgeRetailDeliveryReportingPayload payload = port.pullMessage();
     AcknowledgeRetailDeliveryReportingContent content = payload.getContent().get(0);
     AcknowledgeRetailDeliveryReportingType bod = content.getAcknowledgeRetailDeliveryReporting();

     try
     {
       JAXBContext jaxbContext = JAXBContext.newInstance("org.starstandard.star._5");
       Marshaller marshaller = jaxbContext.createMarshaller();

       JAXBElement<AcknowledgeRetailDeliveryReportingType> processRetailDeliveryReportingElement =
            (new org.starstandard.star._5.ObjectFactory()).createAcknowledgeRetailDeliveryReporting(bod);
       marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
       StringWriter writer = new StringWriter();
       marshaller.marshal( processRetailDeliveryReportingElement, writer );
       String xmlStr = writer.toString();

       System.out.println(xmlStr);
     }
     catch(Exception e)
     {
       e.printStackTrace();
     }

   }
}
```

5. After adding this code to the TestClient class, run the main method again. Now the entire AcknowledgeRetailDeliveryReporting BOD will be printed.

6.2. Apache Axis 2

In this example we will use the Apache Axis2 framework to generate a simple Web Service and client. The homepage for Axis2 is http://ws.apache.org/axis2/ . The target runtime for the service is the Apache Tomcat server. The client will be a standalone application.

This example was created using the JDK 5.0 Update 14, Apache Ant 1.6.5, Apache Axis2 release 1.3, and Tomcat 5.5.

6.2.1. Prerequisites

1. Prerequisites: Follow the instructions on downloading and installing the Axis2 binary distribution (http://ws.apache.org/axis2/1_3/installationguide.html#standalone1 [http://ws.apache.org/axis2/1_3/installationguide.html#standalone1]).

2. Since our target runtime is Tomcat, refer to the Tomcat homepage (http://tomcat.apache.org/) for download and installation instructions. Once Tomcat is installed, follow the instructions to install Axis2 (http://ws.apache.org/axis2/1_3/installationguide.html#servlet_container [http://ws.apache.org/axis2/1_3/installationguide.html#servlet_container]).

Both service and client will be generated using the “WSDL2Java” script included with the Axis2 binary distribution. This script takes a WSDL file as input and generates the appropriate Java classes as output. The “Axis2 Reference Guide” (http://ws.apache.org/axis2/1_3/reference.html) lists the parameters for the script and shows example usage.

6.2.2. Apache Axis 2 Walkthrough

6.2.2.1. Axis 2 Service

Procedure 6.4. Axis 2 Service

1. If the WSDL was retrieved from the “STAR Web Services Specifications” document, it is missing a crucial section in order for the script to properly generate the code. This is the service section, which contains information specific to your Web Service implementation, including the address of the Web Service. Assuming you will be running a local instance of Tomcat for this example, the following information can be added to the STAR WSDL:

```
<wsdl:service name="StarQuickStartServiceAxis2">
  <wsdl:port name="StarTransport" binding="starbindings:starTransport">
    <soap:address location="http://localhost:8080/StarQuickStartServiceAxis2/StarTransport"/>
  </wsdl:port>
</wsdl:service>
```

2. Generating server code: Using the aforementioned WSDL2Java script, generate the Java code for the service. At a windows command prompt, move to the bin directory of your Axis2 installation. Execute the WSDL2Java script with the -uri option to specify the location of the STAR WSDL, the -ss and -sd options to generate the server code and service descriptor, and the -o option to specify the output location. By default the script generates client code. This is why the options are needed for the service artifacts.

Figure 6.3. Axis 2 Generate Service

The script will create a src folder for the source code, a resources folder for the services.xml and a generated wsdl, and an ant build script. These are all the files needed for a simple service implementation. Note that the generated wsdl is identical to the one specified as input to the script, though formatting changes have been made.

3. Modify Service Skeleton with Implementation Code: One of the Java classes generated by WSDL2Java is a “skeleton” for the code you will eventually implement to handle Web Service calls. If you modified the WSDL with the service information from above, this class will be named “StarQuickStartServiceAxis2Skeleton.” Add code to this class to indicate the service is being invoked. If you do not modify this class, the methods will throw an UnsupportedOperationException when called since this is what has been generated by default.

Figure 6.4. Axis 2 Modify Skelton

```
 public org.starstandards.www.webservices._2005._10.transport.PutMessageResponse PutMessage(
     org.starstandards.www.webservices._2005._10.transport.PutMessage putMessage)
{
     System.out.println("PutMessage has been invoked");
     return null;
}
```

4. Generate Axis Archive (*.aar) for the Web Service

Once the skeleton class has been modified, the Web Service is ready to be packaged as an Axis Archive. Using the ant build file generated by WSDL2Java, execute the “jar.server” target.

Figure 6.5. Axis 2 Generate AAR

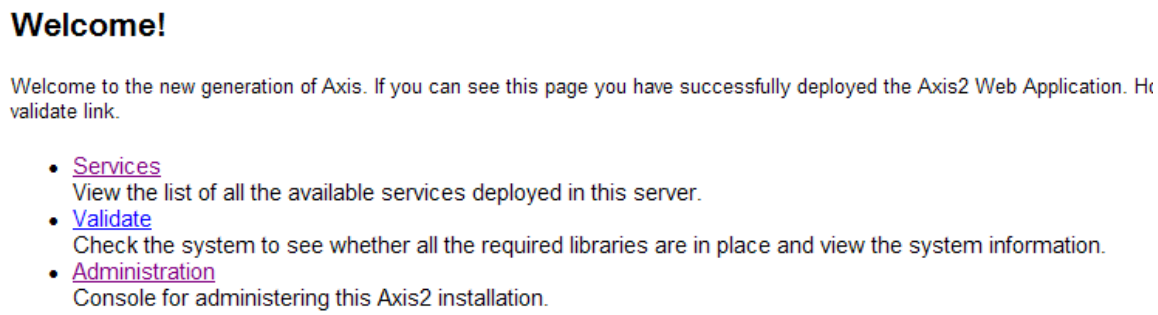
```
C:\axis2-1.3\bin\StarQuickStartServiceAxis2>ant jar.server
Buildfile: build.xml
```

Once the script is complete, StarQuickStartServiceAxis2.aar will be in the build/lib folder.

Procedure 6.5. Deploy Web Service to Tomcat

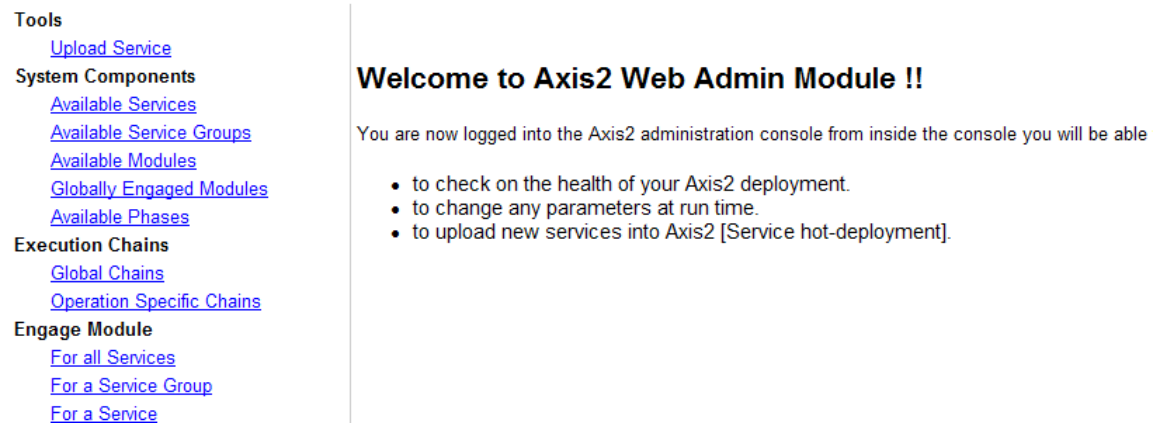
1. The service is ready to be deployed to the application server using the Axis2 Web application you installed as a part of the prerequisites. If you are using defaults, this application can be accessed at http://localhost:8080/axis2/ . The page will look similar to the screenshot below.

Figure 6.6. Axis 2 Deploy Service



2. Click on the “Administration” link. When prompted to login, the initial username/password is admin/axis2. Refer to the “Apache Axis2 Web Administrator’s Guide” (http://ws.apache.org/axis2/1_3/webadminguide.html) for instructions on how to change the default login as well as other information regarding the application. Below is a screenshot of the administration console.

Figure 6.7. Axis 2 Admin Page



3. Use the “Available Services” link to check the deployment of the Web Service. You should see the “StarQuickStartServiceAxis2” listed:

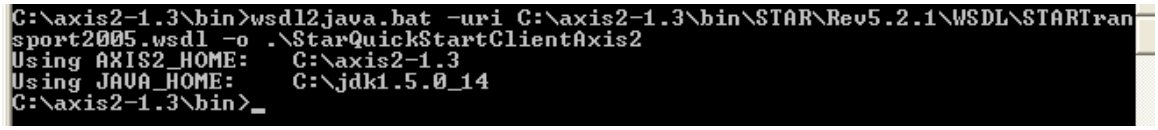
Figure 6.8. Axis 2 Available Services

6.2.2.2. Axis 2 Client

Procedure 6.6. Generating Client Code

- Using the WSDL2Java script again, generate the Java code for the client. Follow the same steps used to generate the service, but exclude the -ss and -sd options that were used to generate the service-specific code. You can also change the output location in order to separate your service and client code.

Figure 6.9. Axis 2 Generate Client Code



Procedure 6.7. Calling the Web Service

1. Use Generated Stub to call Web Service.

   The output of WSDL2Java generating a client will be a “Stub” class that will be used to call the Web Service. For this example, we will use the Stub class to transmit a ProcessPartsOrder BOD via a PutMessage call. Below is an example of a main method using the StarQuickStartServiceAxis2Stub generated by WSDL2Java. This code reads the BOD from a file.

```
package org.star.quickstart.client;

import java.io.*;
import javax.xml.stream.*;
import org.apache.axiom.om.*;
import org.apache.axiom.om.impl.builder.*;
import org.apache.axis2.*;
import org.apache.axis2.databinding.types.*;
import org.starstandards.www.webservices._2005._10.transport.bindings.*;
import org.starstandards.www.webservices._2005._10.transport.bindings.StarQuickStartServiceAxis2Stub.*;

public class TestClient {
  public static void main(String[] args) {
    try {
      String endpoint = "http://localhost:8080/axis2/services/StarQuickStartServiceAxis2";
      StarQuickStartServiceAxis2Stub stub = new StarQuickStartServiceAxis2Stub(endpoint);

      //Create Payload Manifest
      PayloadManifest0 payloadManifest0 = new PayloadManifest0();

      PayloadManifest payloadManifest = new PayloadManifest();
      Manifest manifest = new Manifest();

      manifest.setContentID(new IDRef("Content0"));
      manifest.setElement("ProcessPartsOrder");
      manifest.setNamespaceURI(new URI("http://www.starstandard.org/STAR/5"));
      manifest.setVersion("5.2.1");
      Manifest[] manifestAry = new Manifest[] { manifest };

      payloadManifest.setManifest(manifestAry);
      payloadManifest0.setPayloadManifest(payloadManifest);

      PutMessage putMessage = new PutMessage();
      Payload payload = new Payload();

      //Create Content
      Content content = new Content();
      content.setId(new Id("Content0"));
      //Read in Process Parts Order BOD
      XMLStreamReader parser = XMLInputFactory.newInstance()
        .createXMLStreamReader(
          new FileInputStream("C:\\axis2-1.3\\bin\\STAR\\Rev5.2.1\\BODExamples\\ProcessPartsOrder.xml"));

      StAXOMBuilder builder = new StAXOMBuilder(parser);
      OMElement documentElement = builder.getDocumentElement();

      content.setExtraElement(documentElement);
      payload.addContent(content);
      putMessage.setPayload(payload);

      //Call PutMessage operation
      stub.PutMessage(putMessage, payloadManifest0);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

2. Once this class is complete, it can be used to call the Web Service deployed to Tomcat. If you added a println statement to the PutMessage method of your skeleton, you should now see the message in the Tomcat stdout log. The client may receive an error on the response from the service; this is because additional work is necessary to complete the Web Service and client.

Appendix A. Development Resources

If you are a current STAR member you will have access to the various web services discussion groups available within the STAR online collaboration tool. The discussion groups contain valuable information contributed by STAR and its members regarding the development and implementation of the STAR web services.

If you are a current member and you do not yet have access to the online collaboration tool, please contact David Carver at dcarver@starstandard.org [mailto:dcarver@starstandard.org] to request a user account and login instructions.

- STAR Architecture SIG: [http://www.starstandard.org/SIGARCHITECTURE/Architecture] Contains the latest STAR Transport Package
- STAR XML Schema: [http://www.starstandard.org/SIGXML/XMLSchemas] Contains the latest STAR Schema Repository
- OASIS: [http://www.oasis-open.org/] Contains numerous web services specifications.
- WS-I.org: [http://www.ws-i.org/] Interoperability profiles for web services.
- Microsoft MSDN Developer Library [http://msdn2.microsoft.com/en-us/library/aa139615.aspx]
- Sun Java JAXB: [https://jaxb.dev.java.net/] A data binding framework for XML and java.
- Apache Axis 2: [http://ws.apache.org/axis2/] A data binding framework for XML, java, and C maintained by the Apache Software Foundation.

Works Cited

Apache Axis 2. March 2009. Apache Software Foundation. http://ws.apache.org/ .

Axis 2 Administration Guide. Dec. 2007. Apache Software Foundation. Dec. 2007. http://ws.apache.org/axis2/1_3/webadminguide.html .

Axis 2 Reference. Dec. 2007. Apache Software Foundation. Dec. 2007. http://ws.apache.org/axis2/1_3/reference.html .

Apache Tomcat. Dec. 2007. Apache Software Foundation. Dec. 2007. http://tomcat.apache.org/ .

Eclipse Web Tools Platform. March 2009. Eclipse Foundation. 20 Jan. 2008. http://www.eclipse.org/webtools/ .

Microsoft .NET. 15 Nov. 2007. Wikipedia. http://en.wikipedia.org/wiki/Microsoft.NET .

Java J2EE 1.4. Dec. 2007. java.sun.com. Dec. 2007. http://java.sun.com/j2ee/1.4/download.html .

Java's Web Developers Framework. Dec. 2007. Sun Microsystems. Dec. 2007. http://en.wikipedia.org/wiki/Microsoft.NET .

Stylus Studio XML Enterprise Suite. March 2009. DataDirect. Dec 2007. http://www.datadirect.com/index.ssp .

XML RPC. Oct 2007. XMLRPC.com. http://www.xmlrpc.com .

XML Spy Professional. March 2009. Altova. 20. Jan. 2008. http://www.altova.com/products/xmlspy/xml_editor.html .

Glossary

Asynchronous	In the context of STAR Transport, Asynchronous means that a specific thread of execution sending a message does NOT wait for an application level reply before continuing on to other processing or completing execution. For example, when a STAR message is sent asynchronously over HTTP, the thread of execution waits only for an HTTP reply or HTTP error before continuing, it does not wait for a transport level Acknowledgment or a business level Acknowledgment before continuing; the environment executing the thread may expect a Transport level and or Business level reply to be received later as in inbound message in a completely separate and unrelated HTTP session.
European Committee for Standardization / Information Society Standardization System (CEN/ISSS) (CEN/ISSS)	“CEN/ISSS provides market players with a comprehensive and integrated range of standardization services and products, in order to contribute to the success of the Information Society in Europe.” (CEN)
Digital Certificate	A digital certificate is an electronic attachment issued by a certification authority containing credentials such as “name, a serial number, expiration dates, a copy of the certificate holder's public key (used for encrypting messages and digital signatures), and the digital signature of the certificate-issuing authority so that a recipient can verify that the certificate is real.” (SearchSecurity.com [http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci211947,00.html])
Digital Signatures	A digital signature (not to be confused with a digital certificate) is an electronic signature that can be used to authenticate the identity of the signer of a document, and to ensure that the original content of the message is unchanged. The ability to prove that an original signed message arrived infers that the sender cannot easily repudiate it later.
Extensible Markup Language (XML)	Extensible Markup Language (XML) is a standard way of representing data in human-readable structures to be exchanged between business partners. XML was developed by the W3C.
Encryption	Processing and altering data so only the intended recipient can read or use it. The recipient of the encrypted data must have the proper decryption key and algorithms to decipher the data back to its original form.
Organization for the Advancement of Structured Information Standards (OASIS)	“Organization for the Advancement of Structured Information Standards (OASIS) is a not-for-profit consortium that drives the development, convergence and adoption of open standards for the global information society.” (OASIS)
Payload	Payload is the data that is being transmitted within a message.

Synchronous	In the context of STAR Transport, Synchronous means that the specific thread of execution that sent a message waits for an application level reply before continuing on to other processing or finishing execution.
Web Services	Web Services are a set of standards and protocols for exchanging xml-based data between Web-based applications or systems.
Web Service Description Language (WSDL)	The Web Services Description Language (WSDL) is an XML-based language used for describing Web services. (Wikipedia [http://en.wikipedia.org/wiki/WSDL] 2)
WS-I	“WS-I is an open industry organization chartered to promote Web services interoperability across platforms, operating systems, and programming languages.” (WS-I)