IBM WebSphere Business Integration Adapters

Adapter for Web Services User Guide

V 3.3.x

IBM WebSphere Business Integration Adapters

Adapter for Web Services User Guide

V 3.3.x

Note!
Before using this information and the product it supports, read the information in "Notices" on page 225.

25June2004

This edition of this document applies to IBM WebSphere Business Integration Adapter for Web Services (5724-H09), version 3.3.x.

To send us your comments about IBM CrossWorlds documentation, email doc-comments@us.ibm.com. We look forward to hearing from you.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

About This Document

IBM(R) WebSphere(R) Business Integration Adapter portfolio supplies integration connectivity for leading e-business technologies and enterprise applications. This document describes the installation, configuration, and business object development for the adapter for web services.

Audience

This document is for IBM WebSphere customers, consultants, developers, and anyone who is implementing the WebSphere Business Integration Adapter for web services.

Prerequisites for This Document

A variety of prerequisites are cited throughout this book. Many of these consist of references to Web sites that contain information about, or resources for, web services. You should also be familiar with implementing the WebSphere business integration system. A good place to start is the *Technical Introduction to IBM WebSphere InterChange Server*, which contains cross-references to more detailed documentation.

Related Documents

The complete set of documentation available with this product describes the features and components common to all WebSphere adapter installations, and includes reference material on specific components.

You can install related documentation from the following sites:

- For general adapter information:
  - http://www.ibm.com/websphere/integration/wbiadapters/infocenter
- For using adapters with InterChange Server:
  - http://www.ibm.com/websphere/integration/wicserver/infocenter
  - http://www.ibm.com/websphere/integration/wbicollaborations/infocenter

These sites contain simple directions for downloading, installing, and viewing the documentation.

Note: Important information about this product may be available in Technical Support Technotes and Flashes issued after this document was published. These can be found on the WebSphere Business Integration Support Web site, http://www.ibm.com/software/integration/websphere/support/. Select the component area of interest and browse the Technotes and Flashes sections. Additional information might also be available in IBM Redbooks at http://www.redbooks.ibm.com/.

Typographic Conventions

This document uses the following conventions :

`courier font`	Indicates a literal value, such as a command name, filename, information that you type, or information that the system prints on the screen.
bold	Indicates a new term the first time that it appears.
italic, *`italic`*	Indicates a variable name or a cross-reference.
blue outline	A blue outline, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click inside the outline to jump to the object of the reference.
{ }	In a syntax line, curly braces surround a set of options from which you must choose one and only one.
`[ ]`	In a syntax line, square brackets surround an optional parameter.
`...`	In a syntax line, ellipses indicate a repetition of the previous parameter. For example, `option[,...]` means that you can enter multiple, comma-separated options.
`< >`	In a naming convention, angle brackets surround individual elements of a name to distinguish them from each other, as in `<server_name><`*`connector_name`*`>tmp.log`.
/, \	In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All IBM product pathnames are relative to the directory where the product is installed on your system.
`%text%` and *`$text`*	Text within percent (%) signs indicates the value of the Windows `text` system variable or user variable. The equivalent notation in a UNIX environment is *`$text`*, indicating the value of the *`text`* UNIX environment variable.
ProductDir	Represents the directory where the IBM WebSphere Business Integration Adapters product is installed. The CROSSWORLDS environment variable contains the *ProductDir* directory path, which is IBM\WebSphereAdapters by default.
″	Indicates a choice from a menu such as: Choose File ″ Update ″ SGML References

New in this release

New in release 3.3.x

This release includes the following enchancements:

- Compliance of the Adapter for Web Services (the connector, the WSDL ODA, and the SOAP data handler) with the WS-I Basic Profile 1.0 specifications released in August, 2003.
- Support for SOAP version 1.2; a new SOAP data handler meta-object property (see "MO_DataHandler_DefaultSOAPConfig" on page 108) and a new WSDL ODA property (see Table 47 on page 157) allow you to specify SOAP 1.2 or 1.1.
- The WSDL ODA allows you to reuse business object names based on the same schema to reduce the total number of artifacts needed for deployment. For further information, see Table 47 on page 157.
- The WSDL ODA and SOAP data handler support all possible values for maxOccurs on sequence, choice, and group. For further information, see "maxOccurs indicator on sequence, choice, group and all" on page 132.
- If you have not specified a value for java.protocol.handler.pkgs, the connector uses the default value during initialization. For further information, see "JSSE" on page 82.
- The HTTP protocol listener supports requests with any Accept header values; if necessary, the validation of the header can be delegated to the collaboration.
- The minimum value has changed for the connector-specific property WorkerThreadCount. For further information, see "WorkerThreadCount" on page 89.
- The adapter supports TextMessage and BytesMessage payload types for inbound and outbound messages in SOAP/JMS listeners and handlers.
- In the case of synchronous event processing by SOAP/HTTP(S) listeners, when a response is not populated by a collaboration, the ContentType portion of the Content-Type HTTP header of the response will be set to the ContentType of the request.

As of version 3.3.x, the adapter is not supported on Solaris 7, so references to that platform version have been deleted from this guide.

New in release 3.2.x

This release includes the following enchancements:

- Support for custom properties in all listeners and protocol handlers
- Support for propagating and reading standard headers in the SOAP/HTTP(S) listeners and handlers
- Support for propagating and reading major JMS standard headers in the SOAP/JMS listeners and handlers
- Support for customizing success return code in asynchronous HTTP processing in the SOAP/HTTP and SOAP/HTTPS listeners
- Support for multiple acceptable success codes in asynchronous HTTP processing in the SOAP/HTTP-HTTPS handler
- Support for Bytes Messages in the SOAP/JMS listener

- Support for mime type and message charset customization in SOAP/HTTP and SOAP/HTTPS listeners based on the request ContentType and Request URL
- Support for runtime mime type and message charset customization in the SOAP/HTTP-HTTPS handler based on the response ContentType
- Ability to send and receive any ContentType messages in SOAP/HTTP and SOAP/HTTPS listeners as well as the SOAP/HTTP-HTTPS handler
- Ability to specify at runtime the mime type and charset of outgoing messages in the SOAP/HTTP and SOAP/HTTPS listeners and the SOAP/HTTP-HTTPS handler

Beginning with the 3.2.x version, the adapter for web services does not run on Microsoft Windows NT.

The WSDL ODA has been enhanced: it can now read interface files to generate event processing TLOs.

Adapter installation information has been moved from this guide. For further information, see "Installing the connector and related files" on page 15.

New in release 3.1.x

The web services connector has been enhanced. Among the new features and components are:

- Support for HTTP credential propagation. For further information, see "HTTP credential propagation for event processing" on page 34, "HTTP credential propagation for request processing" on page 50, and "ProxyServer" on page 96.
- The WSDL ODA and SOAP data handler support the following Style/Use properties: rpc/literal, rpc/encoded, and document/literal. For further information, see "Style and Use impact on SOAP messages" on page 111.
- The WSDL Configuration Wizard supports the Use attribute in SOAP Config MOs. For more information on the Use attribute, see "SOAP configuration meta-object: child of every SOAP business object" on page 109. The WSDL Configuration Wizard also support internationalized characters for some elements. For more information see "Processing requirements and exceptions" on page 151.
- The JMS protocol listener has a new configuration property, SessionPoolSize. For further information, see Table 41 on page 93.
- The connector runs on the following platforms:
  - Microsoft Windows NT 4.0 Service Pack 6A or Windows 2000
  - Solaris 7, 8 or AIX 5.1, 5.2 or HP UX 11.i

New in release 3.0.x

The web services connector has been redesigned. Among the new features and components are:

- The web services connector (replacing the SOAP connector and proxy class of prior releases) with bi-directional support for SOAP/HTTP, SOAP/HTTPS, and SOAP/JMS web services
- An enhanced WebSphere Business Integration Toolset capability to expose a collaboration as a web service
- A Web Services Description Language (WSDL) Object Discovery Agent (ODA)

- A new business object structure that utilizes application-specific information (ASI) capabilities
- A new SOAP data handler

Chapter 1 contains an overview of the new adapter. For information on backwards compatibility with prior releases of this product, see Appendix D, "Migrating to 3.0.x," on page 217.

The "CrossWorlds" name is no longer used to describe an entire system or to modify the names of components or tools, which are otherwise mostly the same as before. For example "CrossWorlds System Manager" is now "System Manager," and "CrossWorlds InterChange Server" is now "WebSphere InterChange Server."

The connector has been globalized. For more information, see "Locale-dependent data" on page 3 and Appendix A, "Standard configuration properties for connectors," on page 169.

This guide provides information about using this adapter with WebSphere InterChange Server (ICS).

Chapter 1. Overview of the connector

The connector is a runtime component of the WebSphere Business Integration Adapter for Web Services. The connector allows businesses to aggregate, publish, and consume web services for use either within their organization or by trading partners. The connector and other components described in this document provide the functionality needed to exchange business object information in the body of a Simple Object Access Protocol (SOAP) message.

This chapter describes the scope, components, design tools, and architecture used to implement the WebSphere Business Integration Adapter for Web Services. It also provides an overview of tasks you must complete to install and configure the web services components described in this document. For information about installing and configuring the components, see "Install, configure, and design checklist" on page 12.

Note: The adapter for Web Services implements the standard Adapter Framework API. For this reason, the adapter can operate with any integration broker that the Framework supports. However, the functionality provided by the adapter has been designed specifically to support the IBM WebSphere InterChange Server (ICS) integration broker. Accordingly, when you select the Expose as Web Service option in System Manager, this refers to ICS, and not to any other integration broker.

Adapter for Web Services environment

Before installing, configuring, and using the adapter, you must understand its environmental requirements:

Broker compatibility

The adapter framework that an adapter uses must be compatible with the version of the integration broker (or brokers) with which the adapter is communicating. The 3.3 version of the adapter for Web Services is supported on the following adapter framework and integration broker:

- Adapter framework: WebSphere Business Integration Adapter Framework, versions:
  - 2.2.0
  - 2.3.0

  - 2.3.1
  - 2.4.0
- WebSphere InterChange Server only, versions:
  - 4.2
  - 4.2.1
  - 4.2.2

See the Release Notes for any exceptions.

Note: To utilize all the features documented for the 3.2 version of the adapter, you must install version 4.2.2 of the WebSphere InterChange Server.

Software prerequisites

Review the following assumptions and software requirements before you install the connector for web services:

- The design of the connector and other components is based on the specifications published for SOAP 1.1 and 1.2.
- If you are using SOAP/JMS web services, you must install your own- JMS and JNDI implementation.
- If you are using HTTPS/SSL, you need your own third-party software for creating keystore and truststore.

Adapter platforms

The adapter runs on the following platforms (operating systems):

- Microsoft Windows 2000
- Solaris 8
- AIX 5.1, 5.2
- HP-UX 11i

Standards and APIs

The Adapter for Web Services (the connector, the WSDL ODA, and the SOAP data handler) is in compliance with the WS-I Basic Profile 1.0 specifications released in August, 2003.

A variety of standards and technologies give web services access to their functionality over a network.

The standards used by the adapter are as follows:

- SOAP versions 1.2 and 1.1
- WSDL 1.1 SOAP bindings
- HTTP 1.0
- JMS 1.0.2

The APIs used by the adapter are as follows:

- Apache SOAP 2.3.1 APIs: The connector incorporates the SOAP APIs from Apache Foundation. Apache SOAP APIs are an open source implementation of the SOAP version 1.1. Apache SOAP APIs have the following requirements:
  - Java Activation Framework 1.0.1 (activation.jar)
  - JavaMail(TM) API 1.2 (mail.jar)

- Xerces Java parser 1.4.3 and higher Xerces2 is a fully conforming XML schema processor
- JMS API version 1.0.2
- WSDL4J 1.2.1 - The Web Service Description Language for Java API (WSDL4J) provides an object model for WSDL documents
- UDDI4J-WSDL 2.1.0 - The UDDI4J-WSDL API encapsulate classes present in the UDDI4J API, as well as some defined by the WSDL4J API
- JNDI 1.2.1
- WSDL4J 1.0
- IBM JSSE 1.0.2

Depending on your configuration, you may need to install additional software. The sections below discuss these contingencies.

JMS protocol

If you are using JMS protocol, you must install a JMS provider and create queues. The queue creation really depends on your requirements. You may use JMS Protocol for both exposing a collaboration as a web service and also for invoking external web services. For further information, see "Connector and JMS" on page 80.

JNDI: You must configure the JNDI and then enter appropriate parameters in the JNDI configuration properties for the connector. You also must ensure that the Connection factory and JMS destination (queue) object are made available in the JNDI. If you want to use JNDI and do not have JNDI implementation, you can download the reference implementation of File System JNDI from Sun Microsystems. For further information, see "Connector and JMS" on page 80.

SSL

If you plan to use SSL, you must use third-party software for managing your keystores, certificates, and key generation. No tooling is provided to set up keystores, certificates, or for key generation. You may choose to use keytool (shipped with IBM JRE) to create self-signed certificates and to manage keystores. For further information, see "SSL" on page 82.

Locale-dependent data

The connector has been globalized so that it can support double-byte character sets. When the connector transfers data from a location that uses one character code to a location that uses a different code set, it performs character conversion to preserves the meaning of the data.

The Java runtime environment within the Java Virtual Machine (JVM) represents data in the Unicode character code set. Unicode contains encodings for characters in most known character code sets (both single-byte and multibyte). Most components in the WebSphere business integration system are written in Java. Therefore, when data is transferred between most integration components, there is no need for character conversion.

Note: The connector has not been internationalized. This means that the trace and log messages are not translated.

Web services connector

This section discusses localization and the connector.

Event notification: The connector uses pluggable protocol listeners for event notification. The protocol listeners extract the SOAP message from the transport and invoke the SOAP data handler. This section describes how each of the listeners encode SOAP messages over the transport:

- **SOAP/HTTP and SOAP/HTTPS Listeners** These listeners read the body of the HTTP request message as bytes. The encoding of the body is given by the charset parameter of the HTTP Content-Type header. If the charset parameter is missing, ISO-8859-1(ISO Latin 1) is assumed. The listener uses this encoding to convert the body of the request message into a Java String. This Java String is used to invoke the SOAP data handler. For synchronous (request-response) web services, the SOAP data handler is invoked using the business object returned by the collaboration. The Java String returned by the SOAP data handler is converted into bytes using the encoding from the HTTP request message.
- **SOAP/JMS Listener** This listener supports JMS text messages as well as JMS byte messages.

Request processing: The connector uses pluggable protocol handlers for request processing. The protocol handlers invoke the SOAP data handler. This section describes how each of the handlers encodes SOAP message over the transport:

- **SOAP/HTTP-HTTPS handlers** These handlers invoke the SOAP data handler. To compose the web services request, the string returned by the data handler is converted into bytes using UTF 8 encoding. For synchronous (request-response) web services, the protocol handler reads the body of the HTTP response message. The encoding of the body is given by the charset parameter of HTTP Content-Type header. If the charset parameter is missing, ISO-8859-1 is assumed. The handler uses this encoding to convert the body of the response message into Java String. The SOAP data handler is invoked using this String.
- **SOAP/JMS handler** This listener supports JMS text messages.

SOAP data handler

This section discusses localization and the SOAP data handler.

SOAP character limitations: XML element names and attributes names must be legal ascii characters that are allowed by either business object names, business object attribute names or business object application-specific information.

Internationalized characters are not supported in business object names or business object attribute names. Only attribute values can be internationalized.

SOAP data handler processing: When transforming a SOAP message into a business object, the data handler can receive a string only. The data handler simply populates the business object with string values and returns the business object. Java strings are UCS2, and therefore double-byte enabled characters are transferred without problem. Only XML element and attribute values can be non-ascii characters (see character limitations). When transforming a business object to a SOAP message, the data handler uses the Xerces parser to convert a business object to a string. Java strings are UCS2, so double-byte enabled characters are transferred without problem. Only XML element and attribute values can be non-ascii characters (see character limitations).

WSDL ODA

This section discusses localization and the WSDL ODA.

The WSDL ODA does not support characters other than legal ASCII in the WSDL file. The WSDL ODA can support file names and URLs in other character sets. But the contents of these files must be in legal ASCII.

Properties in the Configuring Agent table of the WSDL ODA are globalized as follows:

- **WSDL_URL** URL can be in native language
- **UDDI_InquiryAPI_URL** Check UDDI registry support
- **WebServiceProvider** Legal ASCII characters only
- **WebService** Legal ASCII characters only
- **MimeType** Legal ASCII characters only
- **BOPrefix** Legal ASCII characters only
- **BOVerb** Legal ASCII characters only
- **Collaboration** Legal ASCII characters only

Terminology

The following terms are used in this Guide:

- **ASI (Application-Specific Information)** is code tailored to a particular application or technology. ASI exists at both the attribute level and business object level of a business object definition.
- **ASBO (Application-Specific Business Object)** A business object that can have ASI.
- **BO (Business Object)** A set of attributes that represent a business entity (such as Customer) and an action on the data (such as a create or update operation). Components of the IBM WebSphere system use business objects to exchange information and trigger actions.
- **Content-Type** The HTTP protocol header that includes the *type/subtype* and optional parameters. For example, in the Content-Type value `text/xml;charset=ISO-8859-1`, `text/xml` is the type/subtype and `charset=ISO-8859-1` is the optional Charset parameter.
- **ContentType** refers to the *type/subtype* portion of the Content-Type header value only. For example, in the Content-Type value`text/xml;charset=ISO-8859-1`, `text/xml` is referred to in this document as the ContentType.
- **MO_DataHandler_DefaultSOAPConfig** Child data handler meta-object specifically for the SOAP data handler.
- **GBO (Generic Business Object)** A business object with no ASI and not tied to any application.
- **MO_DataHandler_Default** Data handler meta-object used by the connector agent to determine which data handler to instantiate. This is specified in the DataHandlerMetaObjectName configuration property of the connector.
- **Non-Top Level Business Object (Non-TLO)**A non-TLO is any business object that does not adhere to the web services TLO structure.
- **Protocol Config MO** During request processing, the SOAP/JMS, SOAP/HTTP-HTTPS protocol handlers use a Protocol Config MO to determine the destination of the target web service. If during event processing you are exposing collaborations as SOAP/JMS web services, the connector uses the Protocol Config MO to convey the JMS message header information from the SOAP/JMS protocol listener to the collaboration.
- **SOAP (Simple Object Access Protocol)** defines a model of using simple request and response messages, written in XML, as the basic protocol for electronic

communication. SOAP messaging is a platform-neutral remote procedure call (RPC) mechanism, but it can be used for the exchange of any kind of XML information (document exchange).

- **SOAP Business Object** A SOAP business object is a child of a TLO and can be a SOAP Request, a SOAP Response or a SOAP Fault business object. SOAP business objects contain information necessary for processing by the SOAP data handler, including SOAP ConfigMOs, which are children of SOAP business objects, and also contain SOAP header container business objects.
- **SOAP Config MO (Configuration Meta Object)** The data handler requires an object that contains configuration information about a single transformation, for example, from a SOAP message to a SOAP business object. This information is stored as meta-data in the child of a SOAP business object. This child object is the SOAP Config MO
- **SOAP Header Child Business Object** A business object that represents a single header element in a SOAP message. The header element is an immediate child of the SOAP-Env:Header element of the SOAP message. All attributes of a header container business object must be of this type. These business objects may have an actor and a mustUnderstand attribute. These attributes correspond to the actor and mustUnderstand attributes of the SOAP header element.
- **SOAP Header Container Business Object** A business object that contains information about the headers in a SOAP message. This business object contains one or more child business objects. Each child business object represents a header entry in the SOAP message. The SOAP data handler business object may have an attribute, which is of type SOAP header container business object. This attribute is also referred to as the SOAP header attribute. Such an attribute has special application-specific information requirements as described in Chapter 5, "SOAP data handler," on page 107. This attribute must be an immediate child of a SOAP business object.
- **Top-Level Business Object** A web services top-level business object contains a SOAP Request, a SOAP Response (optional) and one or more SOAP Fault (optional) business objects. A TLO is used by the connector for both event processing and request processing.
- **Web services** are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. They can be local, distributed, or Web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML. Web services use new standard technologies such as SOAP (Simple Object Access Protocol) for messaging, and UDDI (Universal Description, Discovery and Integration) and WSDL (Web Service Description Language) for publishing.
- **UDDI (Universal Description, Discovery and Integration)** is a specification that defines a way to publish and discover information about web services. UDDI specification provides for XML-based interfaces (APIs) that allow programmatic access to the UDDI registry information. SOAP is the underlying RPC mechanism for these APIs.
- **WSDL (Web Services Description Language)** is an XML vocabulary that defines the software interfaces for web services. It organizes all of the web service technical details required for automatic integration at the programming level, and is used to publish IBM WebSphere collaborations as web services. WSDL is to web services as IDL is to CORBA objects.

  For more information on WSDL, go to:

  http://www.w3.org/TR/wsdl

Components of connector for web services

Figure 1 illustrates the connector for web services, including its protocol handler and listener frameworks and the SOAP data handler.

Note: The Web Services Adapter comes with a limited use license of the XML data handler. The adapter, however, does not require the XML data handler to function.

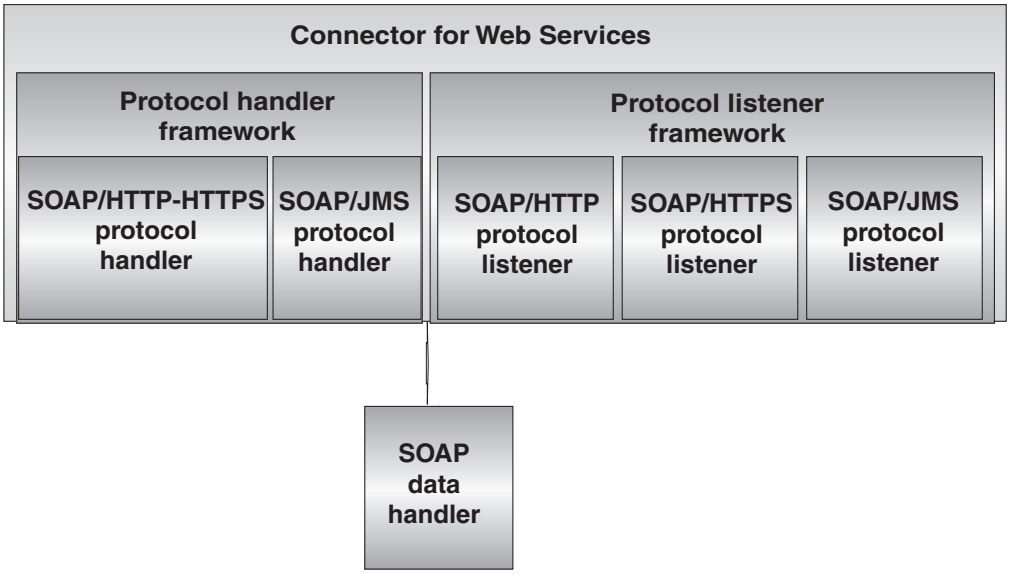


Figure 1. The connector for web services

The following components interact to enable data exchanges across the Internet:

- Web services connector, including the SOAP data handler and protocol listeners and handlers
- Web services-enabled collaborations
- Business objects and SOAP messages
- WebSphere Business Integration InterChange Server

Web services connector

During request processing, the web services connector responds to collaboration service calls by converting business objects to SOAP request messages and conveying them to destination web services. Optionally (for synchronous request processing) the connector converts SOAP response messages to SOAP Response business objects and returns these to the collaboration.

During event processing, the connector processes SOAP request messages from client web services by converting them into SOAP Request business objects and passing them on to collaborations (that have been exposed as web services) for processing. The connector optionally receives SOAP Response business objects from the collaboration, which are converted to SOAP response messages and then returned to client web services.

For further information, see Chapter 4, “Web services connector,” on page 57

Note: In this document, any mention of a connector is a reference to the web services connector, unless specified otherwise.

Protocol listeners and handlers

The connector includes the following protocol listeners and handlers:

- SOAP/HTTP protocol listener
- SOAP/HTTPS protocol listener
- SOAP/JMS protocol listener
- SOAP/HTTP-HTTPS protocol handler
- SOAP/JMS protocol handler

Protocol listeners detect events from internal or external web service clients in SOAP/HTTP, SOAP/HTTPS, or SOAP/JMS formats. They notify the connector of events that require processing by a collaboration that has been exposed as a web service. Protocol listeners then read the business-object-level and attribute-level ASI, connector properties, and transformation rules embedded in protocol configuration objects to determine the collaboration, data handler, processing mode (synchronous/asynchronous) and transport-specific aspects of the web services transaction. For a detailed account of protocol listener processing, see "Protocol listeners" on page 61.

Protocol handlers invoke web services in SOAP/HTTP, SOAP/HTTPS, or SOAP/JMS formats on behalf of a collaboration. Protocol handlers read TLO ASI and transformation rules embedded in protocol configuration objects to determine how to process the request (synchronously or asynchronously), which data handler to use to convert SOAP messages to SOAP business objects and vice versa, and to determine the target address of the web service (from the Destination attribute of the SOAP Request business object Protocol Config MO). For synchronous transactions, the protocol handler processes SOAP response messages, converting them into SOAP Response business objects and passing them back to the collaboration.

For further information on protocol handlers, see "Protocol handlers" on page 73.

SOAP data handler

The SOAP data handler converts SOAP business objects to SOAP messages and vice versa. For further information on the SOAP data handler, see Chapter 5, "SOAP data handler," on page 107.

For further details, see Chapter 5, "SOAP data handler," on page 107.

Web services configuration tools

You can deploy web service solutions with collaborations that invoke, or are exposed as, web services.

When you enable a collaboration for request processing, you use the WSDL Object Discovery Agent (ODA) to generate web service TLOs. For further information on request processing and the WSDL ODA, see Chapter 6, "Enabling collaborations for request processing," on page 141.

When you expose a collaboration as a web service, you use the WSDL Configuration Wizard, which helps you generate a WSDL document for the collaboration that you then publish, for example, via a UDDI registry. The

connector provides no tools for publishing this information. For information on exposing collaborations as web services, see Chapter 7, "Exposing collaborations as web services," on page 143.

Deploying the connector

There are two ways to deploy the web services connector:

- Behind the firewall as an intranet-based solution (see Figure 2) within an enterprise whose business processes communicate in SOAP/HTTP, SOAP/HTTPS, or SOAP/JMS web service formats.

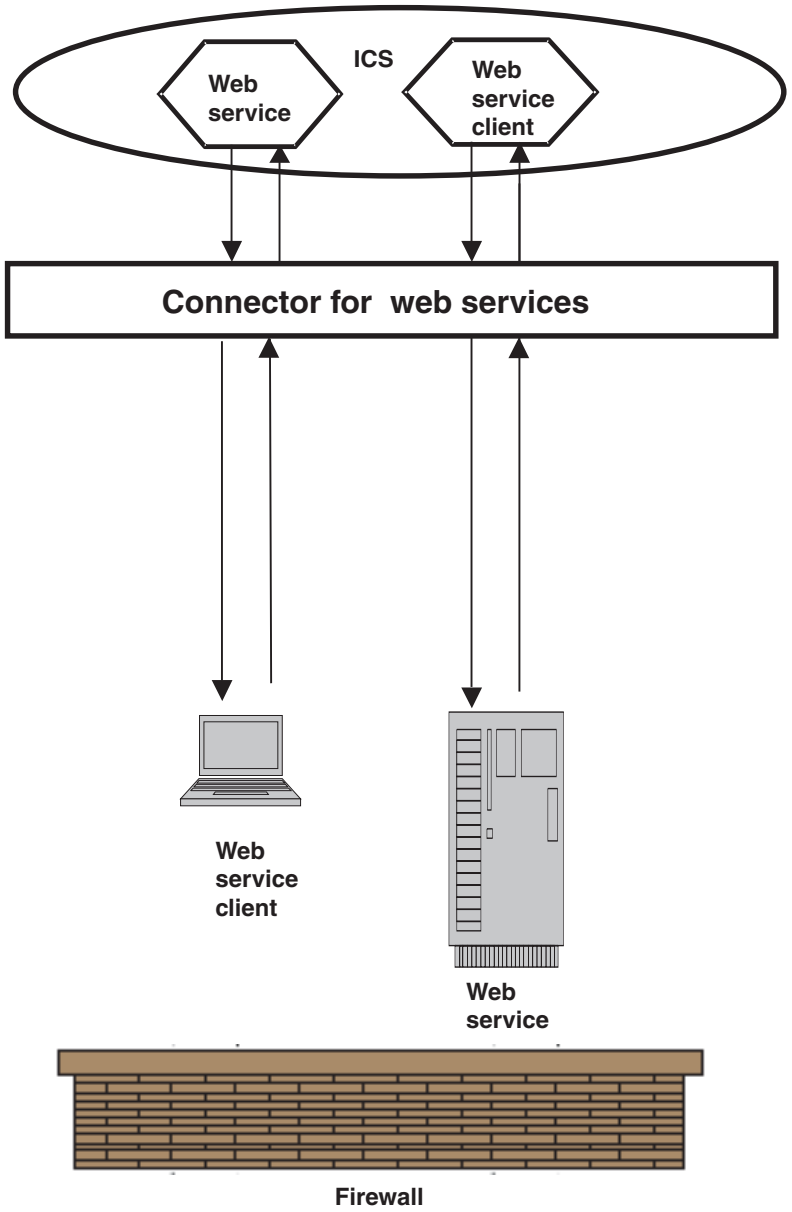


Figure 2. Web services adapter as an intranet solution

- Behind the firewall with a front-end or gateway server to process, filter, and otherwise manage communications with web services that are external to the enterprise.

Note: The web services connector does not include a gateway or front-end for managing incoming or outgoing messages from or to external web services. You must configure and deploy your own gateway. ***The connector must be deployed within the enterprise only, not in the DMZ or outside of the firewall.***

Architecture of connector for web services

To illustrate the architecture of the components at a high level, this section describes two data flows. Figure 3 illustrates the two scenarios. These two scenarios are described below.

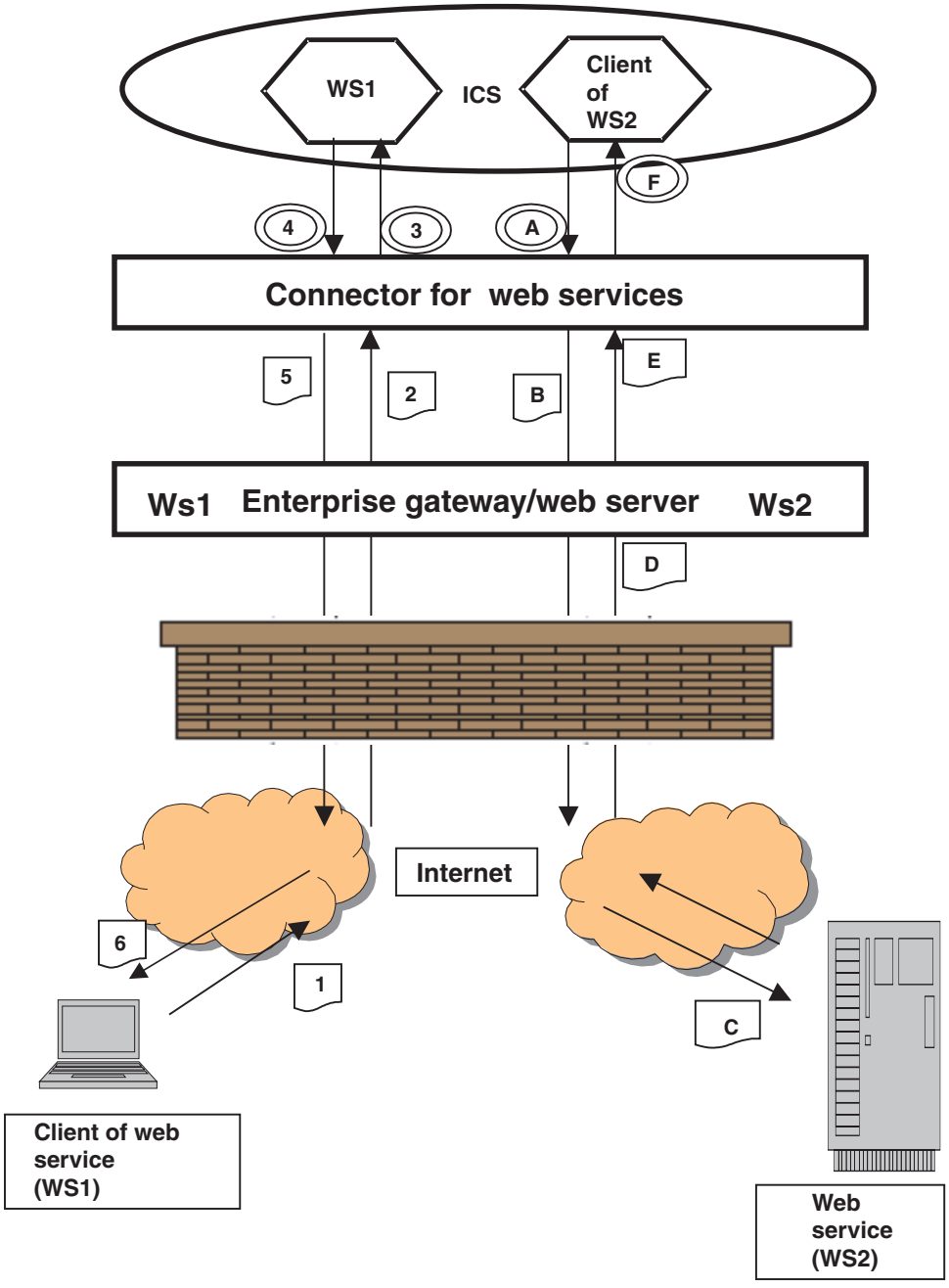


Figure 3. Flow of a web services message

Request processing illustrates the sequence of events that occurs when a collaboration makes a service call request to the connector to invoke a web service. In this scenario, the collaboration plays the role of a client, sending a request to a server.

A The collaboration sends a service call request to the connector, which calls the SOAP data handler to convert the business object to a SOAP request message.

B The connector invokes the web service WS2 by sending the SOAP message. If the destination is an external web service, the connector sends the SOAP message to a gateway. The gateway sends the SOAP message to the endpoint corresponding to the destination web service. This invokes the web service.

C The invoked web service receives the SOAP request message and performs the requested processing.

D The invoked web service sends a SOAP response (or fault) message. If the web service is external to the enterprise, a gateway receives and routes the SOAP response message.

E The SOAP response (or fault) message is routed back to the connector, which calls the data handler to convert it to a response or fault business object.

F The connector returns the SOAP response or fault business object to the collaboration.

Event processing illustrates the sequence of events that occurs when a collaboration is called as a web service. In this scenario, the collaboration, which is exposed as a web service, plays the role of the server, accepting a request from a client, external or internal, and responding as required.

1 The client web service (WS1) sends a SOAP request message to the destination specified in the WSDL document generated for the collaboration.

2 If the client web service is external, the gateway receives and routes the message to the connector.

3 The connector sends the SOAP message to the SOAP data handler to convert the SOAP message to a business object. The connector invokes the collaboration exposed as a web service.

4 The collaboration returns a SOAP Response (or Fault) business object.

5 The connector calls the SOAP data handler to convert the SOAP Response (or Fault) business object to a SOAP response message. The connector returns the response to the gateway.

6 If the client web service is external, the gateway routes the SOAP response message to the client web service (WS1).

Install, configure, and design checklist

This section summarizes the tasks you must perform to install, configure, and design your web services solution. Each section briefly describes the tasks and then provides links to sections in this document (and cross references to related documents) that describe how to perform the task or provide background information.

Installing the adapter

See Chapter 2, “Installation and startup,” on page 15 for a description of what and where you must install.

Configuring connector properties

Connectors have two types of configuration properties: standard configuration properties and connector-specific configuration properties. Some of these properties have default values that you do not need to change. You may need to set the values of some of these properties before running the connector. For more information, see Chapter 4, "Web services connector," on page 57.

Configuring protocol handlers and listeners

You configure protocol handlers and listeners when you assign values to connector configuration properties that govern the behavior of these components. For more information, see Chapter 4, "Web services connector," on page 57.

Enabling collaborations for web services

When you enable collaborations for web services, you create collaborations that can invoke, or be exposed as, web services. You also create or adapt business objects. For an overview of the tasks involved, see "Web services configuration tools" on page 8.

Exposing collaborations as web services

For a step-by-step description see Chapter 7, "Exposing collaborations as web services," on page 143.

Enabling collaborations to invoke web services

For a step-by-step description, see Chapter 6, "Enabling collaborations for request processing," on page 141.

Configuring the SOAP data handler

You configure information in data handler meta-objects after you install the product files, but before startup. Unless you are adding a custom name handler, you can use the default SOAP data handler configuration to save time. You must, however, configure specific meta-object information for each data handler transformation. This information is contained in SOAP Config MOs. You specify SOAP Config MOs when you create business objects. Much of this work is automated when you are developing collaborations that invoke web services (request processing): when you use the WSDL ODA to generate business objects for SOAP messages, the SOAP Config MOs are automatically generated for you.

For further information on configuring the data handler, see Chapter 5, "SOAP data handler," on page 107.

Limitations

- The WSDL ODA automatically generates business objects. If the results do not meet your requirements, you must manually create business objects using Business Object Designer.

  See describes WSDL ODA support for various combinations of attributes style, use, and part definitions using type and element.
- For XML limitations on style (rpc, document) use (literal, encoded), and how parts are defined, see Chapter 5, "SOAP data handler," on page 107 and Chapter 6, "Enabling collaborations for request processing," on page 141.
- The connector supports SOAP/HTTP and SOAP/JMS bindings only.
- The connector's SOAP/JMS protocol listener supports queue destinations only; topics are not supported. JMS text and byte messages are supported.

- HTTP POST Request and Response are supported. No other HTTP method is supported. HTTP 1.1 persistent connection is not supported.

Chapter 2. Installation and startup

This chapter describes how to install components for implementing the connector for web services. For information regarding installation of an ICS system generally, see the *System Installation Guide* appropriate for your platform.

Overview of Installation Tasks

For information on broker compatibility, adapter framework, software prerequisites, dependencies, and standards and APIs, see "Adapter for Web Services environment" on page 1.

To install the connector for web services, you must perform the following tasks:

Install ICS

This task, which includes installing the system and starting ICS, is described in the System Installation Guide. You must install ICS, version 4.2.

To load files into the repository, consult the *Implementation Guide for WebSphere InterChange Server*.

Install the connector and related files

This task includes installing the files for the connector (and related components) from the software package onto your system. See "Installing the connector and related files."

Installing the connector and related files

For information on installing WebSphere Business Integration adapter products, refer to the *Installation Guide for WebSphere Business Integration Adapters*, located in the WebSphere Business Integration Adapters Infocenter at the following site:

http://www.ibm.com/websphere/integration/wbiadapters/infocenter

Installed file structure

The tables in this section show the installed file structure.

Windows connector file structure

The Installer copies the standard files associated with the connector into your system.

The utility installs the connector and adds a shortcut for the connector agent to the Start menu.

Table 1 describes the Windows file structure used by the connector, and shows the files that are automatically installed when you choose to install the connector through Installer.

Table 1. Installed Windows file structure for the adapter

Subdirectory of *ProductDir*	Description
\lib\WBIA.jar	WebSphere Business Integration Adapter jar file
\bin\CWConnEnv.bat	Generic connector startup file
\bin\ODAEnv.bat	Generic ODA startup file
connectors\WebServices\CWWebServices.jar	The web services connector
connectors\WebServices\start_WebServices.bat	The startup file for the connector
DataHandlers\CwSOAPDataHandler.jar	The SOAP data handler
DataHandlers\CwSOAPNameHandler.jar	The SOAP name handlers
repository\DataHandlers\MO_DataHandler_SOAP.xsd	SOAP data handler-related files
bin\Data\App\WebServicesConnectorTemplate	Web services connector template
ODA\WSDL\WSDLODA.jar	The WSDL ODA
ODA\WSDL\start_WSDLODA.bat	The WSDL ODA startup file
connectors\WebServices\dependencies\soap.jar	Apache SOAP API required by the SOAP connector, SOAP data handler, WSDL Configuration Wizard, and WSDL ODA.
connectors\WebServices\dependencies\LICENSE	Apache license file
connectors\WebServices\dependencies\mail.jar	The JavaMail API
connectors\WebServices\dependencies\activation.jar	The Java Activation Framework
connectors\WebServices\dependencies\ibmjsse.jar	JSSE (Java Secure Socket Extension) API from IBM
connectors\WebServices\dependencies\jms.jar	The Java Messaging Service
connectors\WebServices\dependencies\uddi4j-wsdl.jar	Required by WSDL ODA
connectors\WebServices\dependencies\uddi4jv2.jar	Required by WSDL ODA
connectors\WebServices\dependencies\IPL10.txt	License file required by WSDL ODA
connectors\WebServices\dependencies\wsdl4j.jar	Required by WSDL ODA
connectors\WebServices\dependencies\CPL10.txt	License file required by WSDL ODA
connectors\WebServices\dependencies\qname.jar	Required by WSDL ODA
connectors\WebServices\dependencies\j2ee.jar	Required by WSDL ODA
connectors\WebServices\dependencies\wswb2.1.1\common.jar	Required by WSDL ODA
connectors\WebServices\dependencies\wswb2.1.1\ecore.jar	Required by WSDL ODA
connectors\WebServices\dependencies\wswb2.1.1\xercesImpl.jar	Required by WSDL ODA
connectors\WebServices\dependencies\wswb2.1.1\xmlParserAPIs.jar	Required by WSDL ODA
connectors\WebServices\dependencies\wswb2.1.1\xsd.jar	Required by WSDL ODA
connectors\WebServices\dependencies\wswb2.1.1\xsd.resources.jar	Required by WSDL ODA
connectors\WebServices\dependencies\IBMReadme.txt	License
connectors\WebServices\samples\WebSphereICS\WebServicesSample.jar	Repository file for samples
connectors\WebServices\samples\WebSphereICS\CLIENT_SYNCH_TLO_OrderStatus.bo	Sample (synchronous) business object for test connector
connectors\WebServices\samples\WebSphereICS\CLIENT_ASYNCH_TLO_Order.bo	Sample (asynchronous) business object for test connector
connectors\messages\WebServicesConnector.txt	Connector message file
ODA\messages\WSDLODAAgent.txt	Message file for WSDL ODA

Note: All product pathnames are relative to the directory where the product is installed on your system.

UNIX connector file structure

The Installer copies the standard files associated with the connector into your system.

Table 2 describes the UNIX file structure used by the connector, and shows the files that are automatically installed when you choose to install the connector through Installer.

Table 2. Installed UNIX file structure for the adapter

Subdirectory of *ProductDir*	Description
/lib/WBIA.jar	WebSphere Business Integration Adapter jar file
/bin/CWConnEnv.bat	Generic connector startup file
/bin/ODAEnv.bat	Generic ODA startup file
connectors/WebServices/CWWebServices.jar	The web services connector
connectors/WebServices/start_WebServices.sh	The startup file for the connector
DataHandlers/CwSOAPDataHandler.jar	The SOAP data handler
DataHandlers/CwSOAPNameHandler.jar	The SOAP name handlers
repository/DataHandlers/MO_DataHandler_SOAP.xsd	SOAP data handler-related files
bin/Data/App/WebServicesConnectorTemplate	Web services connector template
ODA/WSDL/WSDLODA.jar	The WSDL ODA
ODA/WSDL/start_WSDLODA.sh	The WSDL ODA startup file
connectors/WebServices/dependencies/soap.jar	Apache SOAP API required by the SOAP connector, SOAP data handler, WSDL Configuration Wizard, and WSDL ODA.
connectors/WebServices/dependencies/LICENSE	Apache license file
connectors/WebServices/dependencies/mail.jar	The JavaMail API
connectors/WebServices/dependencies/activation.jar	The Java Activation Framework
connectors/WebServices/dependencies/ibmjsse.jar	JSSE (Java Secure Socket Extension) API from IBM
connectors/WebServices/dependencies/jms.jar	The Java Messaging Service
connectors/WebServices/dependencies/uddi4j-wsdl.jar	Required by WSDL ODA
connectors/WebServices/dependencies/uddi4jv2.jar	Required by WSDL ODA
connectors/WebServices/dependencies/IPL10.txt	License file required by WSDL ODA
connectors/WebServices/dependencies/wsdl4j.jar	Required by WSDL ODA
connectors/WebServices/dependencies/CPL10.txt	License file required by WSDL ODA
connectors/WebServices/dependencies/qname.jar	Required by WSDL ODA
connectors/WebServices/dependencies/j2ee.jar	Required by WSDL ODA
connectors/WebServices/dependencies/wswb2.1.1/common.jar	Required by WSDL ODA
connectors/WebServices/dependencies/wswb2.1.1/ecore.jar	Required by WSDL ODA
connectors/WebServices/dependencies/wswb2.1.1/xercesImpl.jar	Required by WSDL ODA
connectors/WebServices/dependencies/wswb2.1.1/xmlParserAPIs.jar	Required by WSDL ODA
connectors/WebServices/dependencies/wswb2.1.1/xsd.jar	Required by WSDL ODA
connectors/WebServices/dependencies/wswb2.1.1/xsd.resources.jar	Required by WSDL ODA
connectors/WebServices/dependencies/IBMReadme.txt	License
connectors/WebServices/samples/WebSphereICS/WebServicesSample.jar	Repository file for samples
connectors/WebServices/samples/WebSphereICS/CLIENT_SYNCH_TLO_OrderStatus.bo	Sample (synchronous) business object for test connector
connectors/WebServices/samples/WebSphereICS/CLIENT_ASYNCH_TLO_Order.bo	Sample (asynchronous) business object for test connector
connectors/messages/WebServicesConnector.txt	Connector message file
ODA/messages/WSDLODAAgent.txt	Message file for WSDL ODA

Note: All product pathnames are relative to the directory where the product is installed on your system.

Overview of configuration tasks

After installation and before startup, you must configure components as follows:

Configure the connector

This task includes setting up and configuring the connector. See "Configuring the connector" on page 84.

Configure business objects

The steps for configuring business objects depend on how you elect to implement the product suite:

- **Request Processing** You must create the business objects that correspond to:

– The request messages to be sent to each web service
– Each possible response, including faults

For further information, review Chapter 3, "Business object requirements," on page 21 and then see Chapter 6, "Enabling collaborations for request processing," on page 141.

- **Event Processing** You can use TLO or non-TLO business objects.
  For further information, review Chapter 3, "Business object requirements," on page 21 and then see Chapter 7, "Exposing collaborations as web services," on page 143.

Configure the data handler

The SOAP data handler meta-object must be configured after installation. In addition, SOAP Config MOs must be configured for each SOAP business object. To configure the data handler, see Chapter 5, "SOAP data handler," on page 107

Configure collaborations

- **Request processing** For collaborations that invoke web services as part of their processing, you generate business objects using the WSDL ODA and then bind collaboration object ports to the connector. For further information including a step-by-step procedure, see Chapter 6, "Enabling collaborations for request processing," on page 141.
- **Event processing** For a collaboration that is exposed as a web service, you must generate a WSDL document using the WSDL Configuration Wizard, make the document available to potential clients, and then configure the ports of the collaboration object so that clients can invoke the collaboration. For further information including a step-by-step procedure, see Chapter 7, "Exposing collaborations as web services," on page 143.

Running multiple instances of the adapter

Creating multiple instances of a connector is in many ways the same as creating a custom connector. You can set your system up to create and run multiple instances of a connector by following the steps below. You must:

- Create a new directory for the connector instance
- Make sure you have the requisite business object definitions
- Create a new connector definition file
- Create a new start-up script

Create a new directory

You must create a connector directory for each connector instance. This connector directory should be named:

```
ProductDir\connectors\connectorInstance
```

where `connectorInstance` uniquely identifies the connector instance.

If the connector has any connector-specific meta-objects, you must create a meta-object for the connector instance. If you save the meta-object as a file, create this directory and store the file here:

```
ProductDir\repository\connectorInstance
```

Create business object definitions

If the business object definitions for each connector instance do not already exist within the project, you must create them.

1. If you need to modify business object definitions that are associated with the initial connector, copy the appropriate files and use Business Object Designer to import them. You can copy any of the files for the initial connector. Just rename them if you make changes to them.
2. Files for the initial connector should reside in the following directory:

   `ProductDir\repository\initialConnectorInstance`

   Any additional files you create should be in the appropriate `connectorInstance` subdirectory of `ProductDir\repository`.

Create a connector definition

You create a configuration file (connector definition) for the connector instance in Connector Configurator. To do so:

1. Copy the initial connector's configuration file (connector definition) and rename it.
2. Make sure each connector instance correctly lists its supported business objects (and any associated meta-objects).
3. Customize any connector properties as appropriate.

Create a start-up script

To create a startup script:

1. Copy the initial connector's startup script and name it to include the name of the connector directory:

   dirname
2. Put this startup script in the connector directory you created in "Create a new directory" on page 18.
3. Create a startup script shortcut (Windows only).
4. Copy the initial connector's shortcut text and change the name of the initial connector (in the command line) to match the name of the new connector instance.

You can now run both instances of the connector on your integration server at the same time.

For more information on creating custom connectors, refer to the *Connector Development Guide for C++ or for Java*.

Starting and stopping the connector

Important: As noted earlier in this chapter, the connector, business objects, the SOAP data handler meta-objects, and collaborations must be configured after installation and before starting the connector to assure proper operation. For a summary of these tasks, see "Overview of configuration tasks" on page 17. In addition, connector polling should not be disabled (connector polling is enabled by default).

A connector must be explicitly started using its **connector start-up script**. The startup script should reside in the connector's runtime directory:

ProductDir`\connectors\`*connName*

where *connName* identifies the connector. The name of the startup script depends on the operating-system platform, as Table 3 shows.

Table 3. Startup scripts for a connector

Operating system	Startup script
UNIX-based systems	`connector_manager_`*`connName`*
Windows	`start_`*`connName`*`.bat`

You can invoke the connector startup script in any of the following ways:

- On Windows systems, from the **Start** menu

  Select **Programs>IBM WebSphere Business Integration Adapters>Adapters>Connectors**. By default, the program name is "IBM WebSphere Business Integration Adapters". However, it can be customized. Alternatively, you can create a desktop shortcut to your connector.
- From the command line
  - On Windows systems:

    `start_`*`connName connName brokerName`* `[-c`*`configFile`* `]`
  - On UNIX-based systems:

    `connector_manager_`*`connName`* `-start`

  where *connName* is the name of the connector and *brokerName* identifies your integration broker, as follows:
  - For WebSphere InterChange Server, specify for *brokerName* the name of the ICS instance.
  - For WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, or WebSphere Business Integration Message Broker) or WebSphere Application Server, specify for *brokerName* a string that identifies the broker.

  Note: For a WebSphere message broker or WebSphere Application Server on a Windows system, you *must* include the `-c` option followed by the name of the connector configuration file. For ICS, the `-c` is optional.
- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector starts when the Windows system boots (for an Auto service) or when you start the service through the Windows Services window (for a Manual service).

For more information on how to start a connector, including the command-line startup options, refer to one of the following documents:

- For WebSphere InterChange Server, refer to the *System Administration Guide*.
- For WebSphere message brokers, refer to *Implementing Adapters with WebSphere Message Brokers*.
- For WebSphere Application Server, refer to *Implementing Adapters with WebSphere Application Server*.

Chapter 3. Business object requirements

This chapter describes the structure, requirements, and attributes of connector business objects.

Business object meta-data

The connector for web services is a meta-data-driven connector. In business objects, meta-data is data about the application, which is stored in a business object definition and which helps the connector interact with an application. A meta-data-driven connector handles each business object that it supports based on meta-data encoded in the business object definition rather than on instructions hard-coded in the connector.

Business object meta-data includes the structure of a business object, the settings of its attribute properties, and the content of its application-specific information. Because the connector is meta-data-driven, it can handle new or modified business objects without requiring modifications to the connector code. However, the connector's configured data handler makes assumptions about the structure of its business objects, object cardinality, the format of the application-specific text, and the database representation of the business object. Therefore, when you create or modify a business object for web services, your modifications must conform to the rules the connector is designed to follow, or the connector cannot process new or modified business objects correctly.

For more information on meta-data, meta-objects, and their configuration and interaction with business objects and SOAP messages, see Chapter 5, "SOAP data handler," on page 107.

Connector business object structure

The connector processes two kinds of business objects:

- **TLOs** A web services top-level business object (TLO) contains a Request business object and, optionally, Response and Fault business objects. These child objects contain content data as well as SOAP Config MOs, and, optionally, Protocol Config MOs. The TLO, Request, Response, and Fault objects as well as application-specific information, attributes, and requirements with regard to request versus event processing are described and illustrated in the sections below.

  Note: *TLOs are used for request processing and event processing*.

- **Non-TLOs** These are generic business objects (GBOs) and application-specific business objects (ASBOs) that are not TLOs, but which have been used by the WSDL Configuration Wizard in WSDL generation. The connector can process non-TLOs during event processing. These objects are discussed below in "Event processing non-TLOs" on page 39. For further information, see "WSDL Configuration Wizard" on page 146.

  Note: *Non-TLOs are used for event processing only*.

Note: SOAP header container and header business objects, which are included in Request, Response, and Fault business objects, are not discussed in this chapter. For information on SOAP header container and header business objects, see Chapter 5, "SOAP data handler," on page 107.

Synchronous event processing TLOs

For event processing the connector allows two kinds of TLOs—synchronous and asynchronous. This section discusses synchronous event processing TLOs.

Figure 4 on page 23 shows the business object hierarchy for synchronous event processing. Request and Response objects are required, Fault objects are optional.

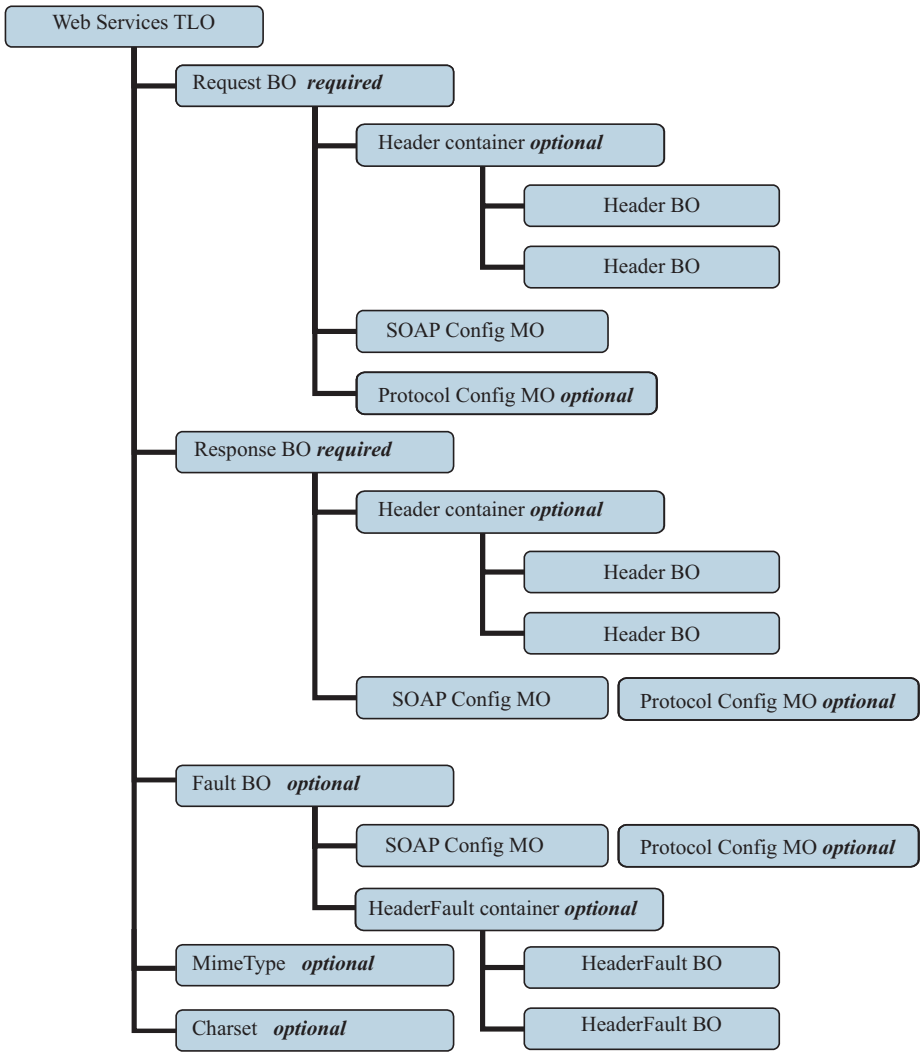


Figure 4. Business object hierarchy for synchronous event processing

The TLO contains object-level ASI as well as attributes with attribute-level ASI. Both kinds of ASI are discussed below.

Object-level ASI for synchronous event processing TLOs

Object-level ASI provides fundamental information about the nature of a TLO and the objects it contains. Figure 5 shows the object-level ASI for SERVICE_SYNCH_OrderStatus, a sample TLO for synchronous event processing.

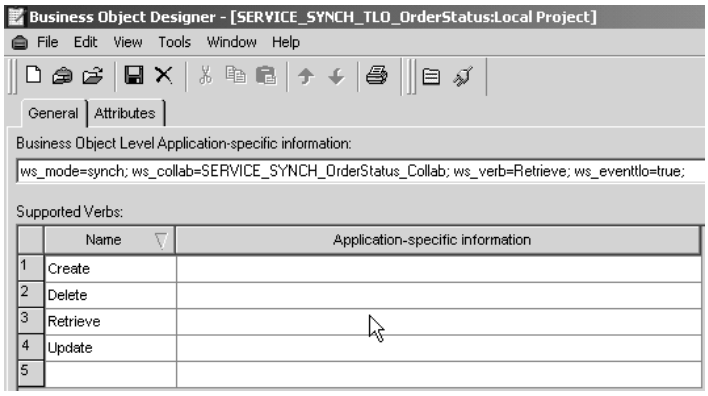


Figure 5. Top-level business object for synchronous event processing

Table 4 below describes the object-level ASI for a synchronous event processing TLO.

Table 4. Synchronous event processing TLO object ASI

Object-level ASI	Description
`ws_eventtlo=true`	If this ASI property is set to `true`, the connector treats this object as a TLO for event processing only. Note that the WSDL Configuration Wizard uses this ASI to determine whether a business object is a TLO. For more on this see "WSDL Configuration Wizard" on page 146.
`ws_collab=`*`collabname`*	This ASI tells the connector which collaboration to invoke. Its value is the name of the collaboration. (This ASI is also used during WSDL generation to determine the TLO for a collaboration. For more on this see "WSDL Configuration Wizard" on page 146.) In the sample shown inFigure 5, the collaboration name is SERVICE_SYNCH_OrderStatus_Collab)
`ws_verb=`*`verb`*	Before delivering the TLO to the collaboration, the connector uses this ASI to set the verb on the TLO. In the sample shown inFigure 5, the verb is Retrieve.
`ws_mode=synch`	During event notification, the connector uses this ASI property to determine whether to invoke the collaboration synchronously (`synch`) or asynchronously (`asynch`). For synchronous processing, this ASI must be set to `synch`. The default is `asynch`.

Attribute-level ASI for synchronous event processing TLOs

Each synchronous event processing TLO has attributes and attribute-level ASI. Figure 6 shows the attributes of SERVICE_SYNCH_OrderStatus, a sample TLO. It also shows the attribute-level ASI in the App Spec Info column.

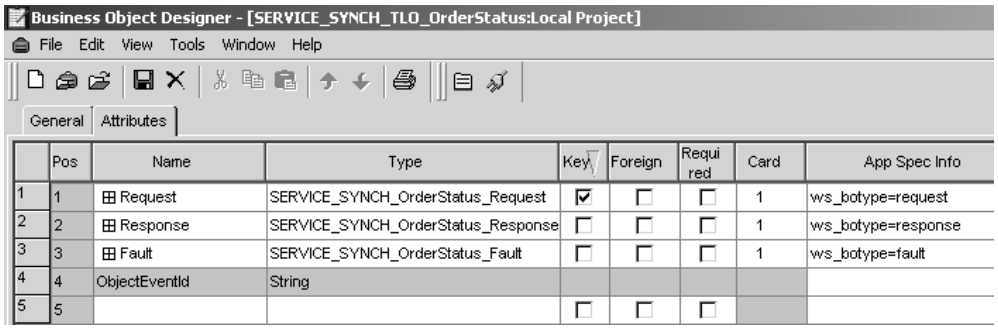


Figure 6. TLO attributes for synchronous event processing

Table 5 summarizes the attribute-level ASI for the Request, Response, Fault, MimeType, and Charset attributes of an synchronous event processing TLO.

Table 5. Synchronous event processing TLO attribute ASI

TLO attribute	Attribute-level ASI	Description
MimeType		Optional attribute; if specified, its value is used as the mime type of the data handler to invoke for the synchronous response. The type is String and the default is `xml/soap`.
Charset		This optional parameter of type String specifies the charset to be set on the data handler when transforming an outgoing business object to the message. NOTE: the charset value specified in this attribute will not be propagated in the Content-Type protocol header of the response message.
Request	`ws_botype=request`	This attribute corresponds to a web service request. The connector uses its ASI to determine whether this TLO attribute is of type SOAP Request BO. This ASI, not the attribute name, determines the attribute type. If there is more than one request attribute, the connector uses the ASI of the first one. This attribute is required for synchronous event processing TLOs.

Table 5. Synchronous event processing TLO attribute ASI (continued)

TLO attribute	Attribute-level ASI	Description
Response	`ws_botype=response`	This attribute corresponds to the response returned by a web service. The connector uses this ASI to determine whether this TLO attribute is of type SOAP Response BO. This ASI, not the attribute name, determines the attribute type. If there is more than one response attribute, the connector uses the ASI of the first one. This attribute is required for synchronous event processing TLOs.
Fault	`ws_botype=fault` `ws_botype=defaultfault`	This attribute, optional for synchronous event processing, corresponds to a fault message returned by a collaboration when it cannot successfully populate a response. The connector uses this ASI, not the attribute name, to determine if the attribute is of type SOAP Fault BO.If `ws_botype=defaultfault`,then the WSDL Configuration Wizard uses this Fault business object for header processing. For further information, see “Header fault processing” on page 118.

Request business object for synchronous event processing

A Request business object is a child of a TLO and is required for synchronous event processing. A Request business object has object-level ASI. For example, if you open SERVICE_SYNCH_OrderStatus_Request in Business Object Designer and click the General tab, the object level ASI is displayed as shown in Figure 7 on page 27.

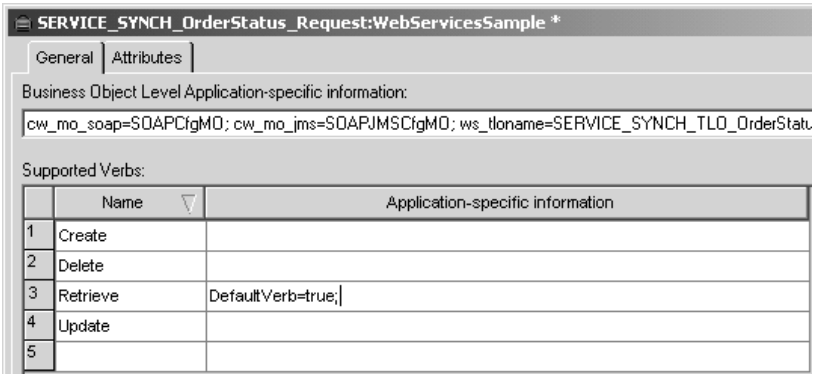


Figure 7. Object-level ASI for synchronous event processing request object

The object-level ASI for a Request business object for synchronous event processing is described in Table 6. As shown in Figure 7, you can specify a default verb for the Request business object. You do so by specifying:

```
DefaultVerb=true;
```

in the ASI field for the verb in the Supported Verbs list at the top-level of the Request business object. If `DefaultVerb` ASI is not specified and the data handler processes a business object with no verb set, the business object is returned without a verb.

Table 6. Synchronous event processing: object-level ASI for Request business objects

Object-level ASI	Description
`cw_mo_soap=`*`SOAPCfgMO`*	The value of this ASI must match the name of the attribute that corresponds to the SOAP Config MO. This is the meta-object that defines the data handler transformation for the Request business object. For further information, see "SOAP Config MO" on page 28.
`cw_mo_jms=`*`SOAPJMSCfgMO`* or `cw_mo_http=`*`SOAPHTTPCfgMO`*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. The first ASI designates the SOAP/JMS protocol listener; the second designates the SOAP/HTTP or SOAP/HTTPS protocol listener. Both the ASI and the Protocol Config MO are optional. For further information, see "Protocol Config MO" on page 29.
`ws_tloname=`*`tloname`*	This ASI specifies the name of the web services TLO that this object belongs to. During event processing, the connector uses this ASI to determine whether the Request business object delivered by the data handler is a child of the TLO. If so, the connector creates the specified TLO, sets the Request business object as its child, and uses the TLOs object-level ASI to deliver it to the subscribing collaboration.

Response business object for synchronous event processing

A Response business object is a child of a TLO and is required for synchronous event processing. The object-level ASI for a Response business object for synchronous event processing is described in Table 7.

Table 7. Synchronous event processing: object-level ASI for Response business objects

Object-level ASI	Description
cw_mo_soap=*SOAPCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the SOAP Config MO. This is the SOAP Config MO that defines the data handler transformation for the Response business object. For further information, see "SOAP Config MO."

Note: You can optionally include a Protocol Config MO object-level ASI for the Response BO.

Fault business object for synchronous event processing

A Fault business object is a child of a TLO and is optional for synchronous event processing. The object-level ASI for a Fault business object for synchronous event processing is described in Table 8.

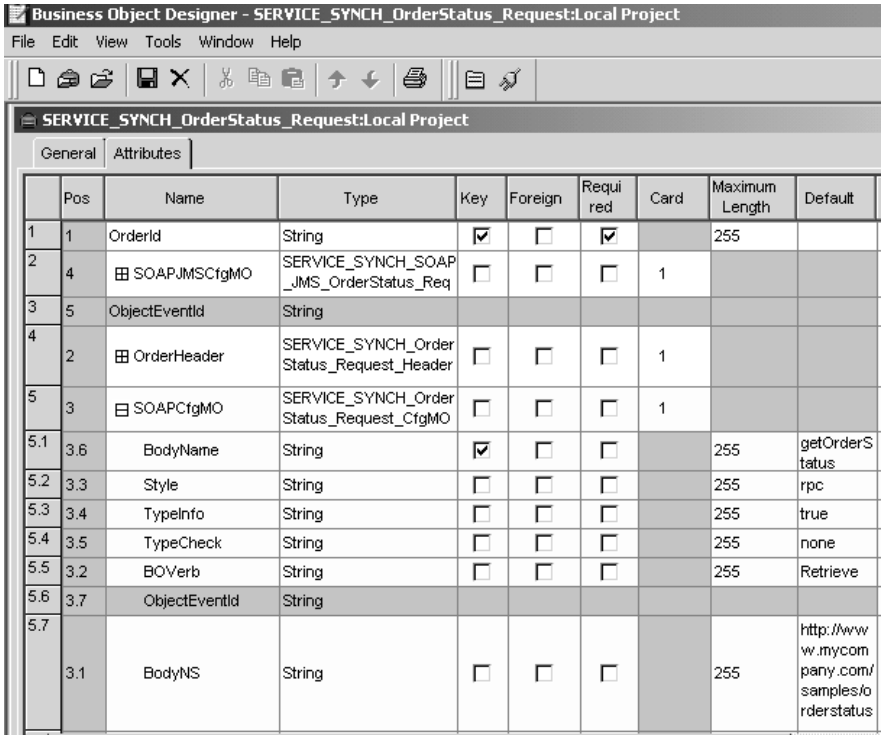
Table 8. Synchronous event processing: object-level ASI for Fault business objects

Object-level ASI	Description
cw_mo_soap=*SOAPCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the SOAP Config MO. This is the SOAP Config MO that defines the data handler transformation for the Fault business object. For further information, see "SOAP Config MO."

Note: You can optionally include a Protocol Config MO object-level ASI for the Fault BO.

SOAP Config MO

Figure 8 shows a sample SOAP Config MO, expanded in Business Object Designer.



	Pos	Name	Type	Key	Foreign	Required	Card	Maximum Length	Default
1	1	OrderId	String	☑	☐	☑		255	
2	4	⊞ SOAPJMSCfgMO	SERVICE_SYNCH_SOAP_JMS_OrderStatus_Req	☐	☐	☐	1		
3	5	ObjectEventId	String						
4	2	⊞ OrderHeader	SERVICE_SYNCH_OrderStatus_Request_Header	☐	☐	☐	1		
5	3	⊟ SOAPCfgMO	SERVICE_SYNCH_OrderStatus_Request_CfgMO	☐	☐	☐	1		
5.1	3.6	BodyName	String	☑	☐	☐		255	getOrderStatus
5.2	3.3	Style	String	☐	☐	☐		255	rpc
5.3	3.4	TypeInfo	String	☐	☐	☐		255	true
5.4	3.5	TypeCheck	String	☐	☐	☐		255	none
5.5	3.2	BOVerb	String	☐	☐	☐		255	Retrieve
5.6	3.7	ObjectEventId	String						
5.7	3.1	BodyNS	String	☐	☐	☐		255	http://www.mycompany.com/samples/orderstatus

Figure 8. SOAP Config MO attributes for synchronous event processing

The SOAP Config MO defines the formatting behavior for one data handler transformation — either a SOAP-message-to-business-object or business-object-to-SOAP-message transformation. Each Request, Response, and Fault attribute has a SOAP Config MO. Its attributes, BodyName, BodyNS, Style, Use, TypeInfo, TypeCheck and BOVerb, are always of type `String`. They correspond to SOAP message elements and their values determine how messages and objects are read and validated by the SOAP data handler. For more information on SOAP Config MOs and attributes, see “SOAP configuration meta-object: child of every SOAP business object” on page 109.. All SOAP Config MOs, whether for a request, response, or fault object, must have unique entries for default values of BodyName and BodyNS.

Protocol Config MO

Figure 9 shows a JMS Protocol Config MO, whose attributes correspond to headers in the inbound SOAP message.

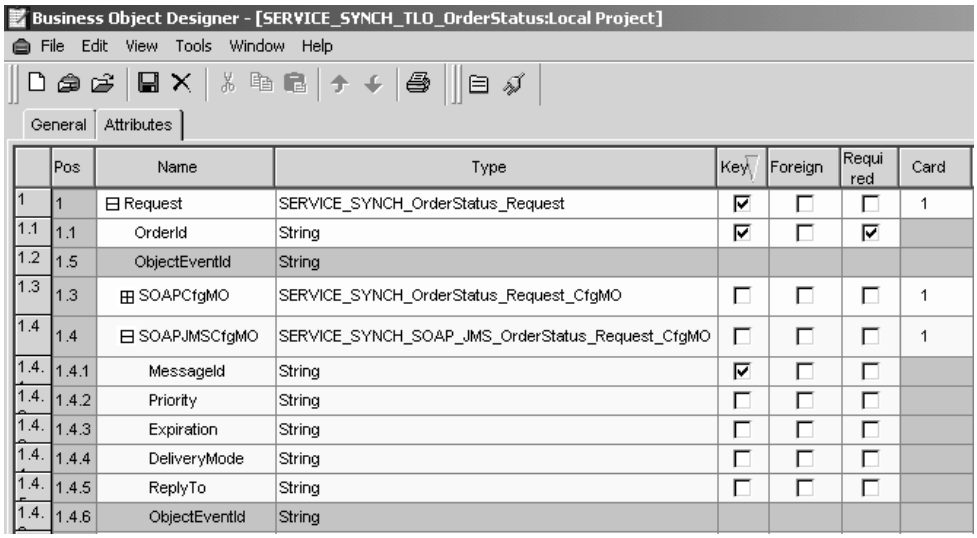


Figure 9. JMS Protocol Config MO attributes for synchronous event processing

This MO is optionally included as a child of the request, response, or fault business objects for event processing. Typically you specify it when you need to read (from request messages) or propagate (to response or fault messages) the protocol headers and custom properties. As noted above, the request business object optionally declares the name of the Protocol Config MO as business-object-level ASI:

- `cw_mo_jms=`*`JMSProtocolListenerConfigMOAttribute`*
- `cw_mo_http=`*`HTTPProtocolListenerConfigMOAttribute`*

During event processing, the connector uses protocol listeners (SOAP/HTTP, SOAP/HTTPS or SOAP/JMS) to retrieve events from the transport. These events are messages from internal or external web service clients requesting service from collaborations that have been exposed as web services. Each transport has its own header requirements. The connector uses the Protocol Config MO to convey the protocol-specific header information from the protocol listener to the collaboration. The Protocol Config MO attributes correspond to headers in the inbound SOAP/JMS message. The connector sets the value of these attributes in the business object using inbound SOAP message content. For SOAP/JMS protocol, the Protocol Config MO attributes for event and request processing are as follows:

Table 9. SOAP JMS Protocol Config MO attributes:event and request processing

SOAP/JMS Protocol Config MO attribute	**JMSHeaderName**	**Description**
CorrelationID	JMSCorrelationID	Inbound messages: this atrribute will be populated with the value from JMSCorrelationID header. Outbound messages: : the value from this attribute will be set as the JMSCorrelationID header of outgoing message.

Table 9. SOAP JMS Protocol Config MO attributes:event and request processing (continued)

SOAP/JMS Protocol Config MO attribute	JMSHeaderName	Description
MessageId	`JMSMessageId`	Inbound messages: this atrribute will be populated with the value from the JMSMessageId header. Outbound messages: this attribute is not used for outbound messages.
Priority	`JMSPriority`	Inbound messages: this atrribute will be populated with the value from the JMSPriority header. Outbound messages: the value from this attribute will be set in the JMSPriority header of outgoing message.
Expiration	`JMSExpiration`	Inbound messages: this atrribute will be populated with the value from the JMSExpiration header. Outbound messages: the value from this attribute will be set in the JMSExpiration header of outgoing message.
DeliveryMode	`JMSDeliveryMode`	Inbound messages: : this atrribute will be populated with the value from the JMSDeliveryMode header. Outbound messages: the value from this attribute will be set in the JMSDeliveryMode header of outgoing message.
Destination	`JMSDestination`	Inbound messages: this atrribute will be populated with the value from the JMSDestination header. Outbound messages: **Request processing** the value from this attribute will be used as the destination queue name and will indirectly be set in the JMSDestination header of outgoing messages to the derived destination path. **Synchronous response** in event notification: this attribute is not used.

Table 9. SOAP JMS Protocol Config MO attributes:event and request processing (continued)

SOAP/JMS Protocol Config MO attribute	JMSHeaderName	Description
Redelivered	`JMSRedelivered`	Inbound messages: this atrribute will be populated with the value from the JMSRedelivered header. Outbound messages: the value from this attribute will be set in the JMSRedelivered header of outgoing message..
ReplyTo	`JMSReplyTo`	Inbound messages: this atrribute will be populated with the value from the JMSReplyTo header. Outbound messages: the value from this attribute will be set in the JMSReplyTo header of outgoing message
TimeStamp	`JMSTimeStamp`	Inbound messages: this atrribute will be populated with the value from the JMSTimeStamp header. Outbound messages: the value from this attribute will be set in the JMSTimeStamp header of outgoing message..
Type	`JMSType`	Inbound messages: this atrribute will be populated with the value from the JMSType header. Outbound messages: the value from this attribute will be set in the JMSType header of outgoing message.
UserDefinedProperties	See "User-defined properties for event processing" on page 33.	This optional read/write attribute will hold the user-defined protocol properties business object. For further information, see"User-defined properties for event processing" on page 33.

Note: It is the responsibility of the collaboration to ensure that the header values passed to the JMS Protocol Config MO are logically correct in the context of a request-response event.

For SOAP/HTTP(S) protocol, the Protocol Config MO attributes are as follows:

Table 10. HTTP/HTTPS Protocol Config MO Attributes for Event Processing

Attribute	Required	Type	Description
Content-Type	No	String	The value of this attribute defines the Content-Type header of the outgoing message (which includes message ContentType and 0 or more parameters --the charset-- for the outgoing message). The syntax is the same as that for the Content-Type header in the HTTP Protocol, for example: `text/html; charset=ISO-8859-4`. If there is no Content-Type attribute defined, the connector uses the ContentType of the request as the ContentType of the response/fault message.
UserDefinedProperties	No	Business object	This attribute holds the user-defined protocol properties business object.
One or more HTTP headers	No	String	This attribute allows the handler to pass or retrieve the value for the specified HTTP header.
Authorization_UserID	No	`String`	This attribute corresponds to the userID of the HTTP basic authentication.
Authorization_Password	No	`String`	This attribute corresponds to the password of the HTTP basic authentication

These attributes are described in:

- "User-defined properties for event processing"
- "HTTP credential propagation for event processing" on page 34

For further information on protocol listeners, see "Protocol listeners" on page 61.(For information describing the Protocol Config MO for request processing, see "Synchronous request processing TLOs" on page 40).

User-defined properties for event processing: You can optionally specify custom properties in the HTTP(S) Protocol Config MO. You do so by including the UserDefinedProperties attribute. This attribute corresponds to a business object that has one or more child attributes with property values. Every attribute in this business object must define a single property to be read (or, for synchronous responses, written) in the variable portion of the message header as follows:

- The type of the attribute should always be String regardless of the protocol property type. The application-specific information of the attribute can contain two name-value pairs defining the name and format of the protocol message property to which the attribute maps.

Table 11summarizes the application-specific information for these attributes.

Table 11. Application-specific information for user-defined protocol property attributes: name=value pair content

Name	Value	Description
ws_prop_name (case-insensitive; if not specified the attribute name will be used as the property name	Any valid protocol property name	This is the name of the protocol property. Some vendors reserve certain properties to provide extended functionality. In general, you should not define custom properties that begin with JMS (for JMS protocol) unless you are seeking access to these vendor-specific features.
ws_prop_type (case insensitive, optional for JMS - if not specified String is assumed; irrelevant for HTTP(S) since only String types make sense)	String, Integer, Boolean, Float, Double, Long, Short	The type of the protocol property. For JMS protocol, the JMS API provides a number of methods for setting property values in the JMS Message: setIntProperty, setLongProperty, setStringProperty, etc. The type of the JMS property specified here dictates which of these methods will be used for setting the property value in the message.

If the given custom property ASI (either the ws_prop_name or ws_prop_type) is invalid and there is no logical way to process this header (such as ignoring the property type for HTTP processing), the connector logs a warning and ignores this property. If the value of the custom property can neither be set nor retrieved after the necessary check against ws_prop_name or ws_prop_type has been performed, the connector logs the error and fails the event.

If the UserDefinedProperties attribute is specified, the connector will create an instance of a UserDefinedProperties business object. The connector then attempts to extract property values from the message and store them in the business object. If at least one property value is successfully retrieved, the connector will set a modified UserDefinedProperties attribute in the Protocol Config MO.

For synchronous event processing, if a UserDefinedProperties attribute is specified and its business object is instantiated, the connector will process each attribute of this child business object and set the message property value accordingly.

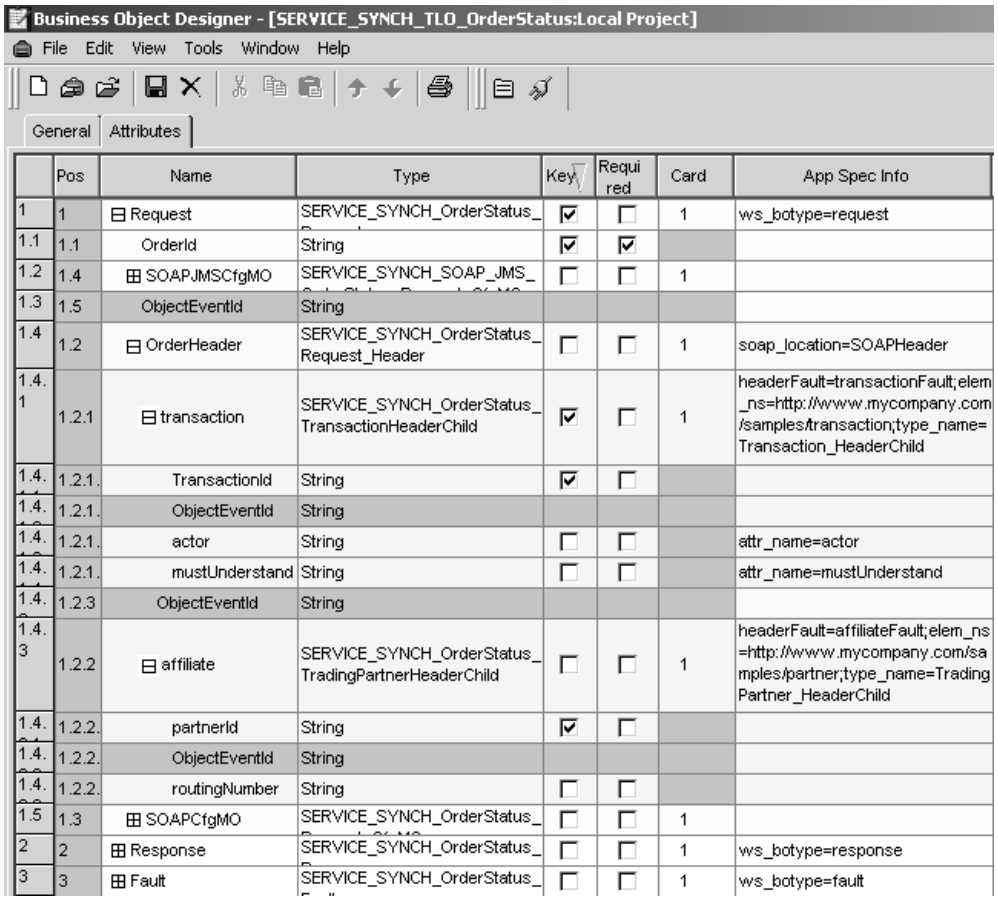
HTTP credential propagation for event processing: For the purpose of credential propagation, the connector supports the Authorization_UserID and Authorization_Password attributes in the HTTP Protocol Config MO. The support is limited to the propagation of these credentials as part of the HTTP Basic authentication scheme.

If a SOAP/HTTP or SOAP/HTTPS protocol listener processes a SOAP/HTTP web service request that includes an authorization header, the listener will parse the header to determine whether it conforms to HTTP Basic authentication. If so, the

listener extracts and decodes (using Base64) the username and password. This decoded string consists of a username and password separated by a colon. If the protocol listener finds the Authorization_UserID and Authorization_Password attributes in the Protocol Config MO, the listener sets these values with those extracted from the event authorization header.

Header container business objects

Figure 10 shows the expanded header container attribute, OrderHeader.



	Pos	Name	Type	Key	Required	Card	App Spec Info
1	1	⊟ Request	SERVICE_SYNCH_OrderStatus_	☑	☐	1	ws_botype=request
1.1	1.1	OrderId	String	☑	☑		
1.2	1.4	⊞ SOAPJMSCfgMO	SERVICE_SYNCH_SOAP_JMS_	☐	☐	1	
1.3	1.5	ObjectEventId	String				
1.4	1.2	⊟ OrderHeader	SERVICE_SYNCH_OrderStatus_ Request_Header	☐	☐	1	soap_location=SOAPHeader
1.4.1	1.2.1	⊟ transaction	SERVICE_SYNCH_OrderStatus_ TransactionHeaderChild	☑	☐	1	headerFault=transactionFault;elem_ns=http://www.mycompany.com/samples/transaction;type_name=Transaction_HeaderChild
1.4.	1.2.1.	TransactionId	String	☑	☐		
1.4.	1.2.1.	ObjectEventId	String				
1.4.	1.2.1.	actor	String	☐	☐		attr_name=actor
1.4.	1.2.1.	mustUnderstand	String	☐	☐		attr_name=mustUnderstand
1.4.	1.2.3	ObjectEventId	String				
1.4.3	1.2.2	⊟ affiliate	SERVICE_SYNCH_OrderStatus_ TradingPartnerHeaderChild	☐	☐	1	headerFault=affiliateFault;elem_ns=http://www.mycompany.com/samples/partner;type_name=TradingPartner_HeaderChild
1.4.	1.2.2.	partnerId	String	☑	☐		
1.4.	1.2.2.	ObjectEventId	String				
1.4.	1.2.2.	routingNumber	String	☐	☐		
1.5	1.3	⊞ SOAPCfgMO	SERVICE_SYNCH_OrderStatus_	☐	☐	1	
2	2	⊞ Response	SERVICE_SYNCH_OrderStatus_	☐	☐	1	ws_botype=response
3	3	⊞ Fault	SERVICE_SYNCH_OrderStatus_	☐	☐	1	ws_botype=fault

Figure 10. Header container and child business objects

The header container attribute, also known as the SOAP header attribute, corresponds to a business object that contains only child business objects. Each child represents a header entry in the SOAP message. In the example shown in Figure 10, the request header container is OrderHeader. SOAP header attributes have application-specific information (ASI) required by the SOAP data handler. For example, a header container business object is identified by its ASI: `soap_location=SOAPHeader`. For information on header processing, see "SOAP data handler processing" on page 113.

All SOAP business objects, whether a Request, Response, or Fault object, have one and only one header container.

Header child business objects

In the example shown Figure 10, the two child attributes of the request header container (OrderHeader) are 1) transaction of type SERVICE_SYNCH_OrderStatus_TransactionHeaderChild and 2) affiliate of type

SERVICE_SYNCH_OrderStatus_TradingPartnerHeaderChild. These attributes correspond to header child business objects. Each represents a single header element in a SOAP message. The header element is an immediate child of the SOAP-Env:Header element of the SOAP message. As shown Figure 10, the header child business objects may have an actor and a mustUnderstand attribute. These attributes correspond to the actor and mustUnderstand attributes of the SOAP header element. For information on header processing, see "SOAP data handler processing" on page 113.

There may be as many header child objects as are needed to represent the SOAP header message elements.

Asynchronous event processing TLOs

Figure 11 shows the business object hierarchy for asynchronous event processing. A request object only is required.

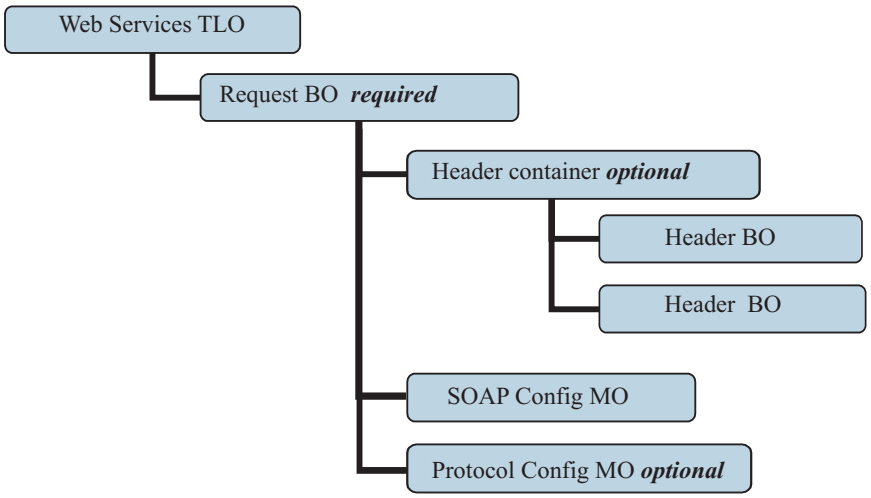


Figure 11. Business object hierarchy for asynchronous event processing

The TLO contains object-level ASI as well as attributes with attribute-level ASI. Both kinds of ASI are discussed below. For information on the header container and header child business objects, see "Header container business objects" on page 35.

Object-level ASI for asynchronous event processing TLOs

Object-level ASI provides fundamental information about the nature of a TLO and the objects it contains. Figure 12 shows the object-level ASI for SERVICE_ASYNCH_TLO_Order, a sample TLO for asynchronous event processing.

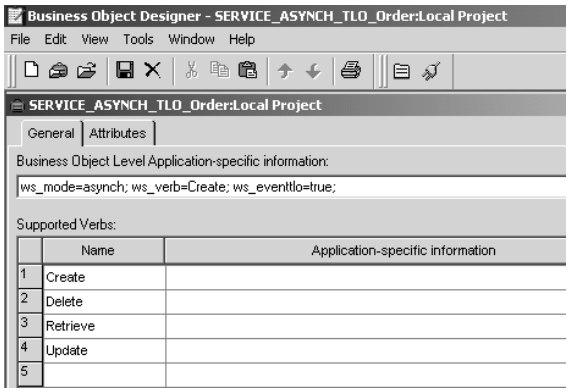


Figure 12. Top-level business object for asynchronous event processing

Table 4 below describes the object-level ASI for an asynchronous event processing TLO.

Table 12. Asynchronous event processing TLO object ASI

Object-level ASI	Description
`ws_eventtlo=true`	If this ASI property is set to `true`, the connector treats this object as a TLO for event processing. Note that the WSDL Configuration Wizard uses this ASI to determine whether a business object is a TLO. For more on this see "WSDL Configuration Wizard" on page 146.
`ws_verb=`*`verb`*	Before delivering the TLO to the collaboration, the connector uses this ASI to set the verb on the TLO. In the sample shown inFigure 12, the verb is `Create`.
`ws_mode=asynch`	During event notification, the connector uses this ASI property to determine whether to invoke the collaboration synchronously (`synch`) or asynchronously (`asynch`). For asynchronous processing, this ASI must be set to `asynch`. The default is `asynch`.

Note: Unlike synchronous event processing, no collaboration name ASI is required at the TLO level for asynchronous event processing. Instead the integration broker assures that application events reach all subscribing collaborations.

Attribute-level ASI for asynchronous event processing TLOs

Each asynchronous event processing TLO has a single attribute that corresponds to a Request business object. Figure 13 shows the request attribute of SERVICE_ASYNCH_TLO_Order, a sample TLO, and the attribute's ASI.

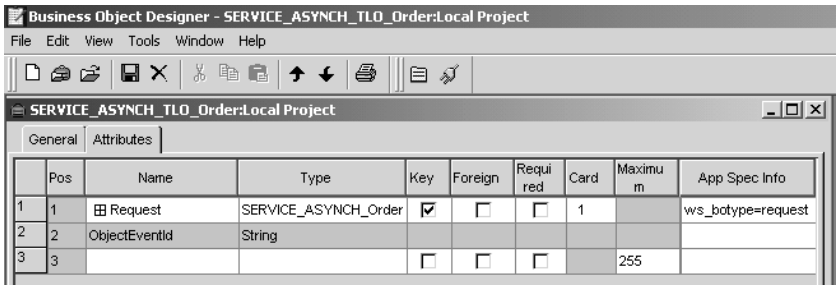


Figure 13. TLO attribute for asynchronous event processing

Table 13 summarizes the attribute-level ASI for the request attribute of an asynchronous event processing TLO.

Table 13. Asynchronous event processing TLO attribute ASI

TLO attribute	Attribute-level ASI	Description
Request	`ws_botype=request`	This attribute corresponds to a web service request. The connector uses its ASI to determine whether this TLO attribute is of type SOAP Request BO. This ASI, not the attribute name, determines the attribute type. If there is more than one request attribute, the connector uses the ASI of the first one. This attribute is required for synchronous event processing TLOs.

Request business object for asynchronous event processing

A Request business object is a child of a TLO and is required for asynchronous event processing. You can specify a default verb for the Request business object. You do so by specifying:

```
DefaultVerb=true;
```

in the ASI field for the verb in the Supported Verbs list at the top-level of the Request business object. If `DefaultVerb` ASI is not specified and the data handler processes a business object with no verb set, the business object is returned without a verb. The object-level ASI for a Request business object for asynchronous event processing is described in Table 14.

Table 14. Asynchronous event processing: object-level ASI for Request business objects

Object-level ASI	Description
`cw_mo_soap=`*`SOAPCfgMO`*	The value of this ASI must match the name of the attribute that corresponds to the SOAP Config MO. This is the SOAP Config MO that defines the data handler transformation for the Request business object. For further information, see "SOAP Config MO" on page 28.

Table 14. Asynchronous event processing: object-level ASI for Request business objects (continued)

Object-level ASI	Description
`cw_mo_jms=SOAPJMSCfgMO` or `cw_mo_http=`*`SOAPHTTPCfgMO`*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. The first ASI designates the SOAP/JMS protocol listener; the second designates the SOAP/HTTP or SOAP/HTTPS protocol listener. Both the ASI and the Protocol Config MO are optional. For further information, see "Protocol Config MO" on page 29.
`ws_tloname=`*`tloname`*	This ASI specifies the name of the web services TLO that this object belongs to. During event processing, the connector uses this ASI to determine whether the Request business object delivered by the data handler is a child of the TLO. If so, the connector creates the specified TLO, sets the Request business object as its child, and uses the TLOs object-level ASI to deliver it to the subscribing collaboration.

In the sample shown in Figure 14, the Request attribute contains a SOAP Config MO and header container (OrderHeader), as well as a content-related attribute (OrderLineItems). The requirements and characteristics of the SOAP Config MO, Protocol Config MO, SOAP header container, and header child business objects are the same for asynchronous event processing as they are for synchronous event processing. For further information, see these topics above in "Synchronous event processing TLOs" on page 22.

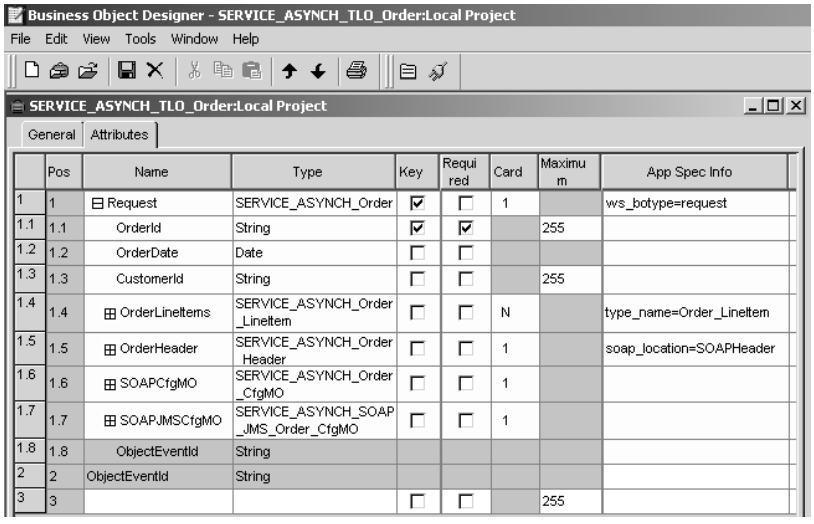


Figure 14. Request attributes for asynchronous event processing

Event processing non-TLOs

If the object-level ASI `ws_eventtlo=true` is not present in a business object, the connector concludes that the object is not a TLO. During event processing, the connector can process non-TLOs—generic business objects and application specific business objects. With non-TLOs, the same business object represents the Request and Response business object.

Non-TLOs do not have SOAP Config MOs. When you expose a collaboration as a web service, the WSDL Configuration Wizard configures the WSCollaborations property of the connector. The connector uses the WSCollaborations property to determine the BodyName and BodyNS of the request message. Note that for non-TLOs, the WSCollaborations property is used for business object resolution.

The advantage to using non-TLOs is that you need not develop new, TLO-structured business objects for use with your web services solution. TLOs, however, allow a more precise and economical exposure of data—customer, company, or otherwise. TLO business objects also lend themselves to more customization than do non-TLOs.

For further information on requirements when using non-TLOs as input to the WSDL Configuration Wizard, see "Identifying or Developing Business Objects" on page 144.

Synchronous request processing TLOs

For request processing the connector allows two kinds of TLOs—synchronous and asynchronous. This section discusses synchronous request processing TLOs.

Figure 15 shows the TLO business object hierarchy for synchronous request processing. Request and Response objects are required, Fault objects are optional. Unlike event processing, a Protocol Config MO is required for the Request objects, and optional for the Response and Fault objects. For information on the header container and header child business objects, see "Header container business objects" on page 35.

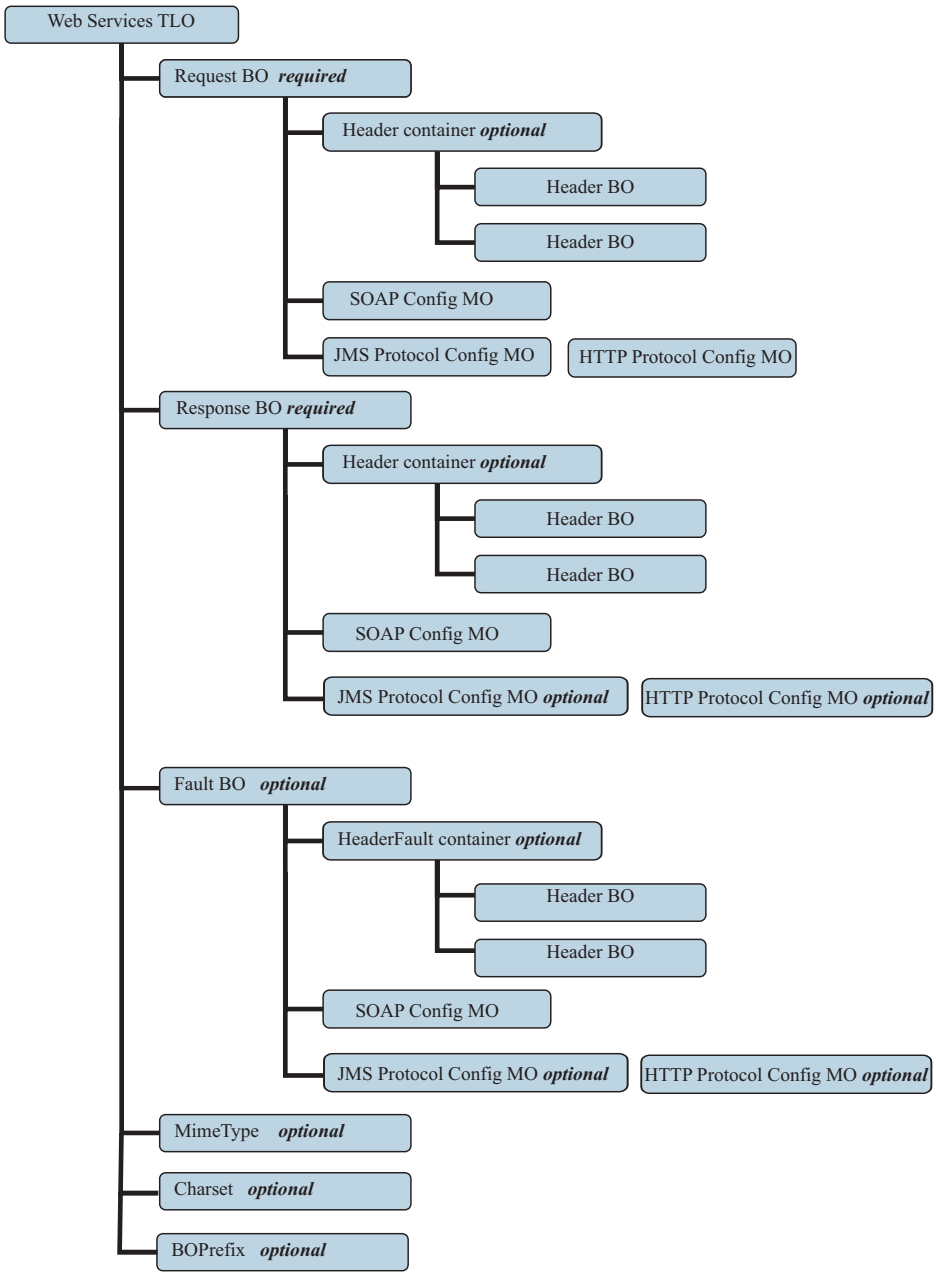


Figure 15. Business object hierarchy for synchronous request processing

Object-level ASI for synchronous request processing TLOs

Object-level ASI provides important information about the nature of a TLO and the objects it contains. Figure 16 shows CLIENT_SYNCH_TLO_OrderStatus, a sample TLO for synchronous request processing.

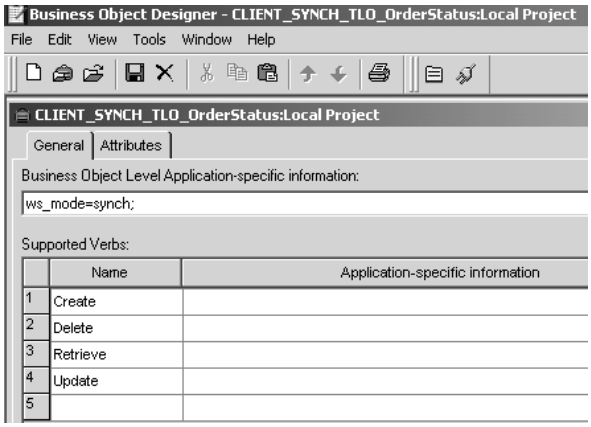


Figure 16. Top-level business object for synchronous request processing

Table 15 describes the object-level ASI for a synchronous request processing TLO. Unlike the ASI for synchronous event processing TLOs, no `ws_collab`, `ws_verb` or `ws_eventtlo` ASI is required at this level for request processing.

Table 15. Synchronous request processing TLO object ASI

Object-level ASI	Description
`ws_mode=synch`	During request processing, the connector uses this ASI property to determine whether to invoke the web service synchronously (`synch`) or asynchronously (`asynch`). If `synch` is indicated, then the connector expects a response, and the TLO must include request and response business objects and, optionally, one or more fault objects. The default is `asynch`.

Attribute-level ASI for synchronous request processing TLOs

Figure 17 shows the attributes of the CLIENT_SYNCH_TLO_OrderStatus TLO as well as attribute-level ASI.

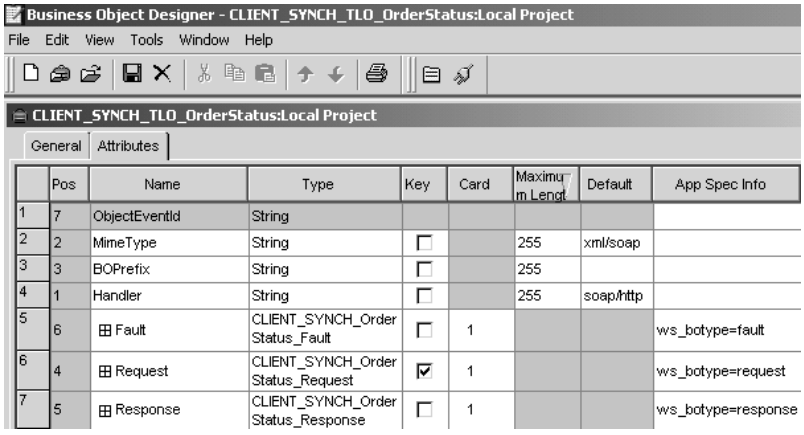


Figure 17. TLO attributes for synchronous request processing

Table 16 describes the attributes and ASI shown in Figure 17.

Table 16. Request processing TLO attributes

TLO attribute	Attribute-level ASI	Description
MimeType	None	This attribute specifies the mime type of the data handler that the connector invokes for transforming a Request business object into a request message. This value may be used for transforming synchronous response/fault messages into business objects, depending on the Message Transformation Rules configuration.
BOPrefix	None	This attribute of type `String` is passed to the data handler.
Handler	None	This attribute specifies the protocol handler to use to process the web service request and is for request processing only. It takes one of the following values: • `soap/jms` The connector uses the SOAP/JMS protocol handler to process the request • `soap/http` The connector uses the SOAP/HTTP, SOAP/HTTPS protocol handler to process this web service request. The default is `soap/http`
Charset		This optional parameter of type String specifies the charset to be set on the data handler when transforming the Request business object to a message. NOTE: the charset value specified in this attribute will not be propagated in the Content-Type protocol header of the request message.
Request	`ws_botype=request`	This attribute corresponds to a web service request business object. The connector uses this attribute ASI to determine whether this TLO attribute is of type SOAP Request BO. This ASI, not the attribute name, determines the attribute type. If there is more than one request attribute, the connector uses the ASI of the first populated attribute.
Response	`ws_botype=response`	This attribute corresponds to the response returned to a collaboration and is required for synchronous request processing. The connector uses this attribute ASI to determine whether this TLO attribute is of type SOAP Response BO. This ASI, not the attribute name, determines the attribute type.

Table 16. Request processing TLO attributes (continued)

TLO attribute	Attribute-level ASI	Description

Fault	`ws_botype=fault`
	or
	`ws_botype=defaultfault`
		This attribute, optional for synchronous request processing, corresponds to a fault message returned by a web service when it cannot successfully populate a response.
		
		The connector uses this ASI to determine if the attribute of TLO is of type SOAP Fault BO. This ASI, not the attribute name, determines the attribute type. A `defaultfault` business object is returned if the fault message is a detail element. `defaultfault` is used in default business object resolution. For further information, see Chapter 5, “SOAP data handler,” on page 107.

Request business object for synchronous request processing

A Request business object is a child of a TLO and is required for synchronous request processing. A Request business object has object-level ASI.

For example, if you open CLIENT_SYNCH_OrderStatus_Request and click the General tab, the object-level ASI is displayed as shown in Figure 18.

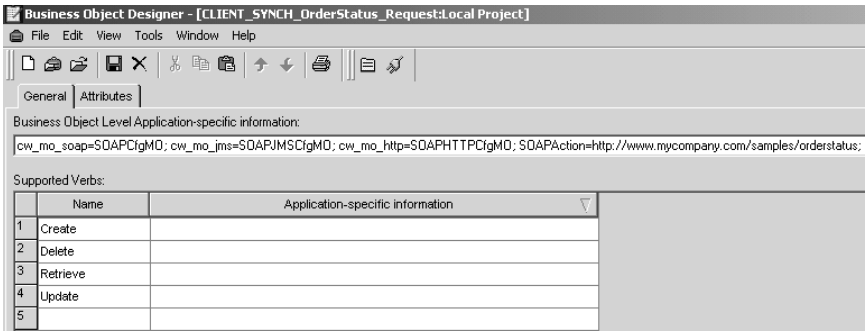


Figure 18. Request object ASI for synchronous request processing

Table 17 describes the object-level ASI for a Request business object for synchronous request processing.

Table 17. Synchronous request processing: object-level ASI for Request business objects

Object-level ASI	Description
`cw_mo_soap=`*`SOAPCfgMO`*	The value of this ASI must match the name of the attribute that corresponds to the SOAP Config MO. This is the SOAP Config MO that defines the data handler transformation for the Request business object. For further information, see “SOAP Config MO” on page 28.

Table 17. Synchronous request processing: object-level ASI for Request business objects (continued)

Object-level ASI	Description
cw_mo_jms=*SOAPJMSCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. This is the Protocol Config MO that specifies the destination web service for the JMS protocol handler. For further information, see "JMS Protocol Config MO of request business object for request processing" on page 46.
cw_mo_http=*SOAPHTTPCfgMO*	The value of this optional ASI must match the name of the attribute that corresponds to the Protocol Config MO. This is a separate Protocol Config MO that specifies the destination for the SOAP/HTTP-HTTPS protocol handler. This ASI is used by the SOAP/HTTP and SOAP/HTTPS Protocol Handler. Note that the TLO request attribute must have either a JMS or an HTTP Protocol Config MO for request processing, depending on the type of web service protocol you are using. For further information, see "HTTP Protocol Config MO for request processing" on page 47.
SOAPAction=*SOAPActionURI*	The connector uses this ASI to determine whether to set a SOAPAction header on the request message. Specify this ASI only if the target web service requires a SOAPAction header. Note that this ASI is used for request processing but not for event notification.

Response business object for synchronous request processing

A Response business object is a child of a TLO and is required for synchronous request processing. The object-level ASI for a Response business object for synchronous request processing is described in Table 18.

Table 18. Synchronous request processing: object-level ASI for response business objects

Object-level ASI	Description
cw_mo_soap=*SOAPCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the SOAP Protocol Config MO. This is the SOAP Config MO that defines the data handler transformation for the Response business object. For further information, see "SOAP Config MO" on page 28.
cw_mo_jms=*SOAPJMSCfg MO* or cw_mo_http=*SOAPHTTPCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. This is the Protocol Config MO, optional for a Response business object, that specifies the headers in the response SOAP message for the JMS or HTTP(s) protocol handler. For further information, see "Protocol Config MO" on page 29

You can specify a default verb for the Response business object. You do so by specifying:

```
DefaultVerb=true;
```

in the ASI field for the verb in the Supported Verbs list at the top-level of the Response business object. If `DefaultVerb` ASI is not specified and the data handler processes a business object with no verb set, the Response business object is returned without a verb.

Fault business object for synchronous request processing

A Fault business object is a child of a TLO and is optional for synchronous request processing. The object-level ASI for a Fault business object for synchronous request processing is described in Table 8.

Table 19. Synchronous request processing: object-level ASI for Fault business objects

Object-level ASI	Description
`cw_mo_soap=SOAPCfgMO`	The value of this ASI must match the name of the attribute that corresponds to the SOAP Protocol Config MO. This is the SOAP Config MO that defines the data handler transformation for the Fault business object. For further information, see "SOAP Config MO" on page 28.
`cw_mo_jms=`*`SOAPJMSCfg MO`* or `cw_mo_http=`*`SOAPHTTPCfgMO`*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. This is the Protocol Config MO, optional for a Fault business object, that specifies the headers in the response SOAP message for the JMS protocol handler. For further information, see "Protocol Config MO" on page 29

SOAP Config MO

The SOAP Config MO (SOAPCfgMO) has the same attributes as those for the event processing SOAP Config MO. For further information, see "SOAP Config MO" on page 28.as well as "SOAP configuration meta-object: child of every SOAP business object" on page 109.

JMS Protocol Config MO of request business object for request processing

The JMS Protocol Config MO is required in a Request business object when you are using JMS web services, and optional for Response and Fault objects. Table 20 on page 47 describes the request processing JMS Protocol Config MO—Destination is the most important and only required attribute. The JMS protocol handler uses this attribute to locate the requested web service. In addition, all the attributes described for the JMS Config MO in "Protocol Config MO" on page 29 are optional.

Table 20. JMS Protocol Config MO Attributes for Request Processing

Attribute	Required	Type	Description
Destination	Yes	`String`	The destination queue name of the target web service. The JMS Protocol Handler uses this attribute to determine the destination of the web service. If the connector-specific JNDI property LookupQueuesUsingJNDI is set to true, the JMS Protocol Handler looks up this queue using JNDI. Make sure that this attribute gives the JNDI name of the destination queue.

HTTP Protocol Config MO for request processing

During request processing, the SOAP/HTTP-HTTPS protocol handlers use the HTTP Protocol Config MO to determine the destination of the target web service. This Protocol Config MO is required for a Request business object. The SOAP/HTTP-HTTPS protocol handlers support HTTP 1.0 POST request only. As shown in Table 21 the sole required attribute (Destination) is the full URL of the target web service. The optional authorization attributes are described in the sections below.

Table 21. HTTP Protocol Config MO Attributes for Request Processing

Attribute	Required	Type	Description
Destination	Yes	`String`	The destination URL of the target web service. The SOAP/HTTP-HTTPS protocol handler uses this attribute to determine the destination of the web service.
Content-Type	Required for the Request business object, otherwise optional.	`String`	The value of this attribute defines the Content-Type header of the outgoing message (which includes message ContentType and optionally charset for the outgoing message). The syntax is the same as that for the Content-Type header in the HTTP Protocol, for example: `text/html; charset=ISO-8859-4`. If there is no Content-Type attribute defined, the connector uses `text/xml` as the ContentType of the message.
Authorization_UserID	No	`String`	This attribute corresponds to the userID of the HTTP basic authentication. For further information, see “HTTP credential propagation for request processing” on page 50
Authorization_Password	No	`String`	This attribute corresponds to the password of the HTTP basic authentication. For further information, see “HTTP credential propagation for request processing” on page 50
One or more HTTP headers	No	String	This attribute allows the handler to pass or retrieve the value for the specified HTTP header.

Table 21. HTTP Protocol Config MO Attributes for Request Processing (continued)

Attribute	Required	Type	Description
UserDefinedProperties	No	Business object	This attribute holds the user-defined protocol properties business object. For further information, see "User-defined properties for request processing."
MessageTransformationMap	No	Single cardinality business object	This is the attribute that points to business object holding 0 or more message transformation rules. The rules hold information regarding the mime type and charset to apply to the incoming message that is specified in the rule. For further information, see "Message transformation maps" on page 49.

Figure 19 shows some of the HTTP Protocol Config MO attributes in Business Object Designer.

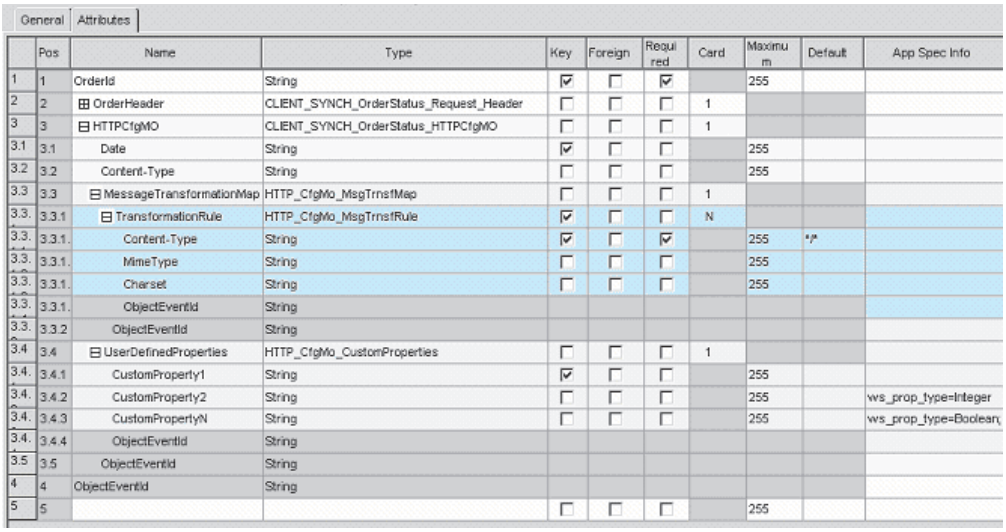


Figure 19. HTTP Protocol Config MO attributes for request processing

The HTTP Protocol Config MO attributes are described in:

- "User-defined properties for request processing"
- "Message transformation maps" on page 49
- "HTTP credential propagation for request processing" on page 50

User-defined properties for request processing: You can optionally specify custom properties in the HTTP Protocol Config MO. You do so by including the UserDefinedProperties attribute. This attribute corresponds to a business object that has one or more child attributes with property values. Every attribute in this business object must define a single property to be read (or, for synchronous responses, written) in the variable portion of the message header as follows:

- The type of the attribute should always be `String` regardless of the protocol property type. The application-specific information of the attribute can contain two name-value pairs defining the name and format of the protocol message property to which the attribute maps.

Table 22summarizes the application-specific information for these attributes.

Table 22. Application-specific information for user-defined protocol property attributes: name=value pair content

Name	Value	Description
ws_prop_name (case-insensitive; if not specified the attribute name will be used as the property name	Any valid protocol property name	This is the name of the protocol property. Some vendors reserve certain properties to provide extended functionality. In general, you should not define custom properties that begin with JMS (for JMS protocol) unless you are seeking access to these vendor-specific features.
ws_prop_type (case insensitive, optional for JMS - if not specified String is assumed; irrelevant for HTTP(S) since only String types make sense)	String, Integer, Boolean, Float, Double, Long, Short	The type of the protocol property. For JMS protocol, the JMS API provides a number of methods for setting property values in the JMS Message: setIntProperty, setLongProperty, setStringProperty, etc. The type of the JMS property specified here dictates which of these methods will be used for setting the property value in the message.

If the given custom property ASI (either the ws_prop_name or ws_prop_type) is invalid and there is no logical way to process this header (such as ignoring the property type for HTTP processing), the connector logs a warning and ignores this property. If the value of the custom property can neither be set nor retrieved after the necessary check against ws_prop_name or ws_prop_type has been performed, the connector logs the error and fails the event.

If the UserDefinedProperties attribute is specified and its business object is instantiated, the connector processes each attribute of this child business object and sets the message properties values accordingly.

For synchronous request processing, upon receipt of a response message from the web service/url, if the UserDefinedProperties attribute is specified, the connector creates an instance of a UserDefinedProperties business object and attempts to extract property values from the message and then stores them in the new business object. If at least one property value was successfully retrieved, the connector will set modified UserDefinedProperties business object in the Protocol Config MO.

Message transformation maps: The Message Transformation Map (MTM) feature is supported for request processing HTTP(S) protocol handlers only. MessageTransformationMap is an optional attribute in the Protocol Config MO that points to a business object. The business object contains rules for transforming messages with mime types and charsets that are specified in the rules. If it finds the (case-sensitive) attribute name MessageTransformationMap and this attribute is of the business object type (see Figure 19), the connector uses the rules in that object to transform a message.

As shown inFigure 19, the MTM attribute must have one cardinality N child business object attribute that is named TransformationRule. When trying to find TransformationRule for a message, the SOAP/HTTP(s) Protocol Handler first attempts to match the message exactly by the ContentType specified in all TransFormationRules. If unsuccessful, the connector attempts to find the rule that applies to multiple types of messages. For further information on protocol handler processing, see "SOAP/HTTP-HTTPS protocol handler processing" on page 74.

Each instance of a TransformationRule business object must have attributes specified as shown in Table 23.

Table 23. TransformationRule attributes for MessageTransformationMaps in HTTP Protocol Config MO

Attribute name	Required	Type	Default value	Description
TransformationRule	No	Business object, cardinality N		This is the attribute that holds 1 rule for message transformation. There can be 0 or more instances of this attribute under the MessageTranformationMap attribute.
+ContentType	Yes	`String`	`*/*`	The value of this property specifies the HTTP ContentType of the message for which this transformation rule applies. The default value `*/*` for this attribute enables the connector to apply this rule to any ContentType. For further information on protocol handler processing, see "SOAP/HTTP-HTTPS protocol handler processing" on page 74.Note that if Protocol Handler finds more than one rule that has the same ContentType as the other rule, Protocol Handler will log the warning and ignore all duplicate rules, but will use unique rules
+MimeType	No			The mime type to use when calling a data handler while processing messages of the ContentType specified in this business object.
+Charset	No			The charset to use when transforming a request of the ContentType specified in this business object.

HTTP credential propagation for request processing: For the purpose of credential propagation, the connector supports the Authorization_UserID and Authorization_Password attributes in the HTTP Protocol Config MO. The support is limited to the propagation of these credentials as part of the HTTP Basic authentication scheme.

If credential propagation is desired during request processing, you must manually add the Authorization_UserID and Authorization_Password attributes to the Protocol Config MO generated by the WSDL ODA. You do this in Business Object

Designer after generating the business object and meta-object definitions. (For further information on the WSDL ODA, see Chapter 6, "Enabling collaborations for request processing," on page 141.)

The collaboration sets the values of the Authorization_UserID and Authorization_Password attributes in the Protocol Config MO. If these attributes are neither null nor empty, the connector creates an authorization header on the request its sends to the to the target web service. The SOAP HTTP/HTTPS protocol handler follows *HTTP Authentication: Basic and Digest Access Authentication (RFC 2617)* when creating the authorization header.

Note: The digest authentication scheme is not be supported, nor is the optional challenge-response mechanism for HTTP authentication defined in Rfc2617. If the HTTP(s) protocol handler is invoking a server that requires a credential, the connector does not wait for the challenge response from the server. Instead, it sends the credentials continuously.

Asynchronous request processing TLOs

Figure 20 shows the business object hierarchy for asynchronous request processing. A request object only is required, and this object contains a SOAP Config MO for the SOAP data handler as well as two Protocol Config MOs, one each for the SOAP/JMS and SOAP/HTTP/HTTPS protocol handlers. These are described in the sections below.

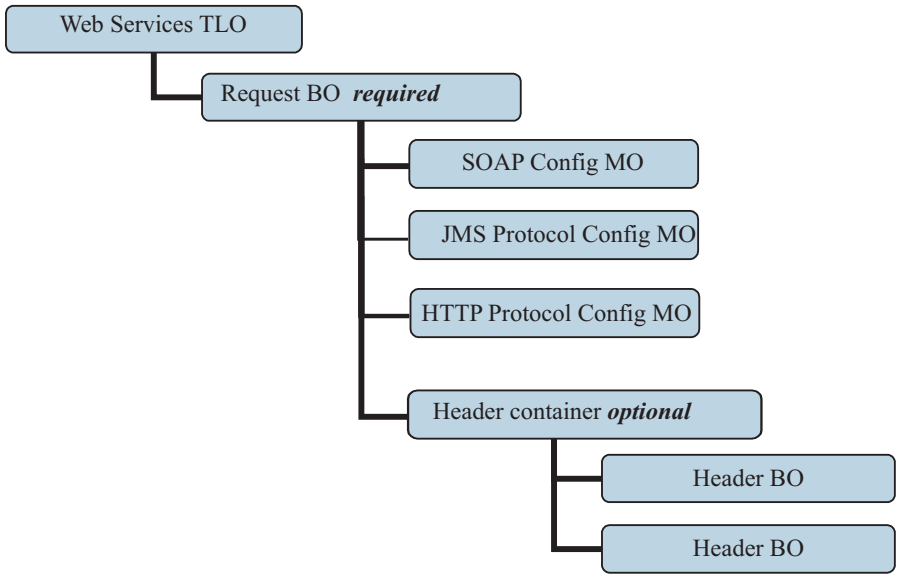


Figure 20. Business object hierarchy for asynchronous request processing

The TLO contains object-level ASI as well as attributes with attribute-level ASI. Both kinds of ASI are discussed below. For information on the header container and header child business objects, see "Header container business objects" on page 35.

Object-level ASI for asynchronous event processing TLOs

Figure 21 shows CLIENT_ASYNCH_Order_TLO, a sample TLO for asynchronous request processing.

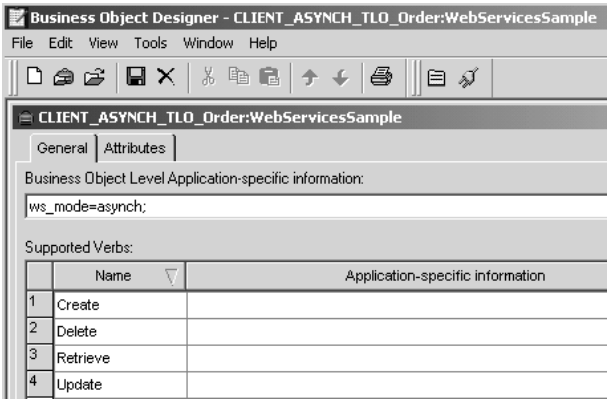


Figure 21. Top-level business object for asynchronous request processing

Table 24 below describes the object-level ASI for an asynchronous request processing TLO.

Table 24. Asynchronous request processing TLO object ASI

Object-level ASI	Description
`ws_mode=asynch`	During request processing, the connector uses this ASI property to determine whether to invoke the collaboration synchronously (`synch`) or asynchronously (`asynch`). For asynchronous request processing, this ASI must be set to `asynch`. The default is `asynch`.

Attribute-level ASI for asynchronous request processing TLOs

Figure 22 shows the attributes of the CLIENT_ASYNCH_TLO_Order, a sample request processing TLO.

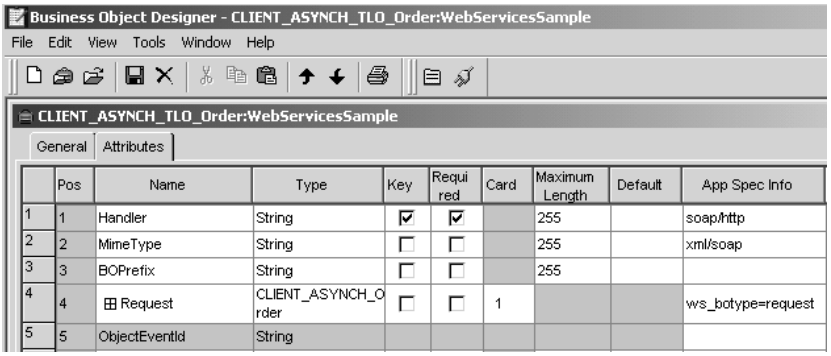


Figure 22. TLO attributes for asynchronous request processing

Table 25 summarizes the attribute-level ASI for the request attribute of an asynchronous request processing TLO.

Table 25. Asynchronous request processing TLO attributes

TLO attribute	Attribute-level ASI	Description
MimeType	None	This attribute specifies the mime type of the data handler that the connector invokes. Note that this attribute is used only for Request Processing. (For event processing, protocol listeners use the SOAPDHMimeType connector-specific configuration property.) The default is `xml/soap`.
BOPrefix	None	This attribute of type `String` is reserved for future development and not required.
Handler	None	This attribute specifies the protocol handler to use to process the web service request and is for request processing only. It takes one of the following values: • `soap/jms` The connector uses the SOAP/JMS protocol handler to process the request • `soap/http` The connector uses the SOAP/HTTP-HTTPS protocol handler to process this web service request. The default is `soap/http`
Request	`ws_botype=request`	This attribute corresponds to a web service request business object. The connector uses this attribute ASI to determine whether this TLO attribute is of type SOAP Request BO. This ASI, not the attribute name, determines the attribute type. If there is more than one request attribute, the connector uses the ASI of the first one.

Request business object for asynchronous request processing

A Request business object is a child of a TLO and is required for asynchronous request processing. The object-level ASI for a Request business object for asynchronous request processing is described inTable 26.

Table 26. Asynchronous request processing: object-level ASI for Request business objects

Object-level ASI	Description
`cw_mo_soap=`*`SOAPCfgMO`*	The value of this ASI must match the name of the attribute that corresponds to the SOAP Config MO. This is the SOAP Config MO that defines the data handler transformation for the Request business object. For further information, see "SOAP Config MO" on page 28.

Table 26. Asynchronous request processing: object-level ASI for Request business objects (continued)

Object-level ASI	Description
cw_mo_jms=*SOAPJMSCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. This is the Protocol Config MO that specifies the destination web service for the JMS protocol handler. For further information, see "JMS Protocol Config MO of request business object for request processing" on page 46.
cw_mo_http=*SOAPHTTPCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. This is a separate Protocol Config MO that specifies the destination for the SOAP/HTTP-HTTPS protocol handler. This ASI is used by the SOAP/HTTP-HTTPS Protocol Handler. Note that the TLO request attribute must have both JMS and HTTP Protocol Config MOs for request processing. For further information, see "HTTP Protocol Config MO for request processing" on page 47.
SOAPAction=*SOAPActionURI*	The connector uses this ASI to determine whether to set a SOAPAction header on the request message. Specify this ASI only if the target web service requires a SOAPAction header. Note that this ASI is used for request processing but not for event notification.

In the sample shown in Figure 14, the Request attribute contains a SOAP Config MO and header container (OrderHeader), as well as a content-related attribute (OrderLineItems). The requirements and characteristics of the SOAP Config MO, Protocol Config MO, SOAP header container, and header child business objects are the same for asynchronous request processing as they are for synchronous request processing. For further information, see these topics above in "Synchronous request processing TLOs" on page 40..

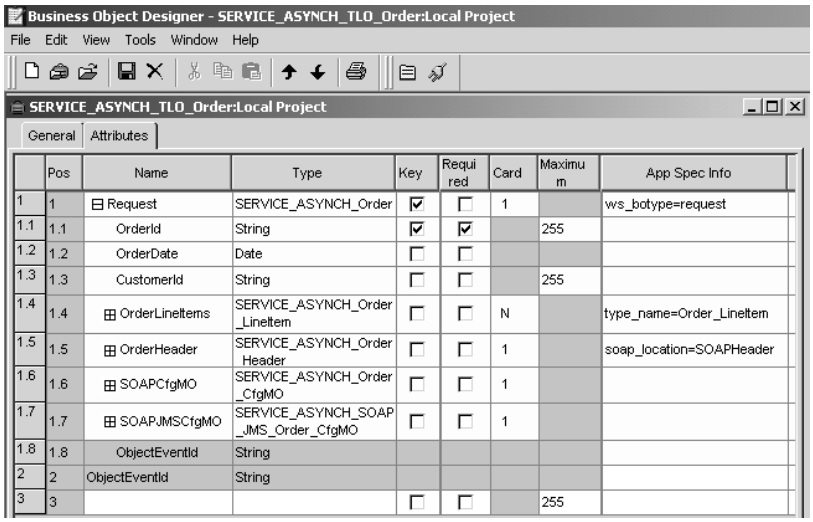


Figure 23. Request attributes for asynchronous event processing

Config MOs for asynchronous request processing

The SOAP Config MO (SOAPCfgMO) has the same attributes as those for the event processing SOAP Config MO. For further information, see “SOAP Config MO” on page 28. as well as “SOAP configuration meta-object: child of every SOAP business object” on page 109.

The JMS Protocol Config MO is required in a Request business object when you are using JMS web services. For further information, see “JMS Protocol Config MO of request business object for request processing” on page 46.

During request processing, the SOAP/HTTP-HTTPS protocol handlers use the HTTP Protocol Config MO to determine the destination of the target web service. This Protocol Config MO is required for a Request business object. For further information, see “HTTP Protocol Config MO for request processing” on page 47.

Developing business objects

You use Business Object Designer to create business objects and Connector Configurator to configure the connector to support them. For more information on the Business Object Designer tool, see the *Business Object Development Guide* and Chapter 7, “Exposing collaborations as web services,” on page 143. For further information on Connector Configurator, see Appendix B, “Connector Configurator,” on page 187.

Chapter 4. Web services connector

This chapter describes the web services connector and how to configure it.

All WebSphere business integration connectors operate with an integration broker. The web services connector operates with the IBM WebSphere InterChange Server integration broker, which is described in the *Technical Introduction to IBM WebSphere InterChange Server*.

A connector is a runtime component of an adapter. Connectors consist of an application-specific component and the connector framework. The application-specific component contains code tailored to a particular application. The connector framework, whose code is common to all connectors, acts as an intermediary between the integration broker and the application-specific component. The connector framework provides the following services between the integration broker and the application-specific component:

- Receives and sends business objects
- Manages the exchange of startup and administrative messages

This document contains information about the application-specific component and connector framework. It refers to both of these components as the connector.

For more information about the relationship of the integration broker to the connector, see the *System Administration Guid*e.

Connector processing

The connector includes a protocol listener framework for event processing and a protocol handler framework for request processing. This bi-directional functionality enables the connector framework to:

- Expose collaborations as web services and then process calls from web service clients
- Process a request by a collaboration that invokes a web service

For further information on the SOAP data handler, see Chapter 5, "SOAP data handler," on page 107.

Note: The connector supports SOAP/HTTP and SOAP/JMS bindings only.

Event processing overview

Connector event processing (or event notification) is used to handle requests from web service clients. This event processing capability encompasses a protocol listener framework, including the following components, which are discussed in greater detail later in this chapter:

- SOAP/HTTP protocol listener
- SOAP/HTTPS protocol listener
- SOAP/JMS protocol listener

The connector uses the listeners to expose collaborations as web services, and to listen on the transport for calls from web services clients to exposed collaborations.

The SOAP/HTTP and SOAP/HTTPS protocol listeners expose a collaboration as a SOAP/HTTP web service. The SOAP/JMS protocol listener exposes a collaboration as a SOAP/JMS web service.

When requests from web service clients arrive, the listener converts the SOAP request message into a business object and invokes the collaboration. If it is a synchronous request, the connector receives a Response business object of the same type as the Request business object. The listener converts the Response business object into a SOAP response message. The listener then transports the SOAP response message to the web service client. Note that event sequencing is not a requirement for this connector; the connector may deliver the events in any order.

The web services connector utilizes the SOAP data handler to convert incoming SOAP request messages into business objects. To aid the data handler in determining which business object to resolve for the incoming SOAP request message, the connector provides meta information regarding its supported business objects to the data handler. From its supported business objects, the connector first makes a list of all business objects that are potential candidates for the conversion. This list may be comprised of both TLOs and non-TLOs. Supported TLO business objects are those that have object-level ASI `ws_eventtlo=true`.

If TLOs are used, the protocol listener reads the object-level ASI of the TLO as follows:

- `ws_collab=` This determines which collaboration to invoke
- `ws_mode=` This determines how to invoke the collaboration, synchronously (`synch`) or asynchronously (`asynch`)

If non-TLOs, are used, then the protocol listener reads the collaboration and processing mode from the WSCollaborations configuration property values generated by the WSDL Configuration Wizard.

The connector compares and attempts to match the BodyName and BodyNamespace in the SOAP request to the names of potential business objects. In the case of TLOs, this BodyName/BodyNamespace pair is found using the SOAP Config MO properties of the SOAP Request business object. For non-TLOs, the BodyName/BodyNamespace pair is found using the WSCollaborations connector configuration property. (Note that the connector considers only those non-TLOs that have an entry in the WSCollaborations property.) The data handler uses the BodyName/BodyNamespace pair to determine the business object to use for the SOAP request to business object conversion.

The connector inspects the Request business object returned by the SOAP data handler. If this business object has `ws_tloname` ASI, the connector sets the Request business object in this TLO. This TLO is used to invoke the collaboration. However, if this ASI is not set, the connector invokes the collaboration using the Request business object returned by the SOAP data handler.

For synchronous collaboration execution, the connector utilizes the SOAP data handler to create a SOAP response or fault message to send back to the client. In this case, the connector simply passes either a SOAP business object (child of TLO), or a non-TLO to the data handler. The SOAP data handler returns a SOAP message based on the business object that it is passed to it.

Request processing overview

On behalf of a collaboration, the connector can invoke web services over SOAP/HTTP(S) and SOAP/JMS. This request processing functionality is supported by a WSDL Object Discovery Agent (ODA) and by a protocol handler framework. The WSDL ODA is a design-time tool you use to generate SOAP business objects that include information about the target web services. For further information, see Chapter 6, "Enabling collaborations for request processing," on page 141. The protocol handler framework is a configurable run-time module that consists of the following components, which are discussed in detail later in this chapter:

- SOAP/HTTP-HTTPS protocol handler
- SOAP/JMS protocol handler

Upon receipt of a collaboration Request business object, which is always (via the WSDL ODA) set in a TLO, the protocol handler framework loads the appropriate protocol handler. The protocol handlers manage transport-level details required for invoking the web service and (optionally) securing a response, performing three main tasks: converting a collaboration Request business object into a SOAP request message, invoking the endpoint web service with the request message, and, if in Request/Response (synchronous) mode, converting the SOAP response message into a business object and returning that object to the collaboration. The connector uses the SOAP/HTTP-HTTPS protocol handler to invoke SOAP/HTTP(S) web services, and the SOAP/JMS protocol handler to invoke SOAP/JMS web services.

The web services connector is always called from a collaboration using TLOs. The connector determines the SOAP Request business object from the TLO, and invokes the SOAP data handler with this business object. The data handler returns a request message which is sent on by the connector to the web service.

For synchronous web service execution, the connector utilizes the SOAP data handler to convert SOAP response and fault messages into SOAP Response and Fault business objects. To aid the data handler in determining which business object to resolve for these SOAP response/faults to business object conversions, the connector provides the data handler with specific meta information. Specifically, the connector makes a list of all Response and Fault business objects that are children of the invoking TLO. There should be only one response business object and, optionally, many Fault business objects. There may also be one and only one defaultfault business object. The connector attempts to match, and then map, the SOAP BodyName and BodyNamespace to a business object name that appears in the list of all Response business objects. In the case of SOAP Response business objects, this pair is found using the SOAP Config MO properties of the SOAP Response business object. In the case of SOAP Fault business objects, this pair is found using the elem_name and elem_ns attribute-level ASI properties for the first child of the detail element. For the defaultfault business object, the connector

simply notifies the data handler of the name of the defaultfault business object. The defaultfault business object should be resolved by the data handler as a last resort if no other fault business objects are resolved for this transformation.

SOAP/HTTP(S) web services

Web services support the HTTP transport protocol. HTTP embodies a client-server model in which an HTTP client opens a connection and sends a request message to an HTTP server. The client request message is to invoke a web service. The HTTP server dispatches the message containing the invocation and closes the connection.

The connector's SOAP/HTTP and SOAP/HTTPS protocol listeners make use of the HTTP client-server and the Request/Response models when handling client requests to a collaboration exposed as a web service. However, the SOAP/HTTP listener is not intended to function as an HTTP server— proxy, intermediary, or otherwise. Rather the SOAP/HTTP listener functions as an endpoint for use within an enterprise and behind a firewall. Accordingly, a separate web server or gateway must be deployed in the firewall to route client requests to the listener. For further information, see Chapter 1, "Overview of the connector," on page 1.

The SOAP/HTTP and SOAP/HTTPS protocol listeners expose a collaboration as a SOAP/HTTP(S) web service. The connector uses the SOAP/HTTP-HTTPS protocol handler to invoke SOAP/HTTP(S) web services.

Synchronous SOAP/HTTP(S) web service

From the perspective of connector processing, a synchronous HTTP web service is one that follows a Request/Response path. If the SOAP/HTTP or SOAP/HTTPS protocol listener successfully processes an HTTP request message, the body will contain the web service response and an HTTP status code of `200 OK`. If a fault is returned, then the body contains the fault message and a status code of `500`.

Asynchronous SOAP/HTTP(S) web service

From the perspective of connector processing, an asynchronous HTTP web service is one that follows a request-only path. If the SOAP/HTTP or SOAP/HTTPS protocol listener successfully receives and processes a request-only web service operation, an HTTP status code of `202 Accepted` is generated. You can also configure the connector to generate an HTTP status code of `200 OK` —for further information see the HTTPAsyncResponseCode property inTable 40. If a fault occurs, an HTTP status code of `500` is generated. There is no response, although a fault body may be returned.

SOAP/JMS web services

JMS is a transport level API that enterprises can combine with web service solutions for messaging, data persistence, and access to Java-based applications. A SOAP/JMS web service is a web service that implements a JMS queue-based transport.

A web service solution may implement a JMS destination for a queue or a topic. The connector's SOAP/JMS protocol listener supports queue destinations only; topics are not supported. JMS text messages only are supported.

During event processing, a SOAP/JMS web service client wraps a request message with a JMS message and publishes it to the queue whose JMS destination is a connector. The JMS destination retrieves the JMS message containing the web

service request and extracts the SOAP request message from the JMS message. It then processes the SOAP request message.

Synchronous SOAP/JMS web service

For synchronous connector processing (Request/Response), a response message is wrapped with a JMS message (like that of the request message). The JMS message containing the web service response is then sent to the JMSReplyTo queue from the incoming request. JMS headers in the response message are set to the values of the headers in the JMS request message as follows:

- The JMSCorrelationID of the response message must be set to the value of JMSMessageID from the JMS request message
- The JMS DeliveryMode of the response message is set to the JMSDeliveryMode of the request.
- The JMSPriority of the response message is set to the JMSPriority of the request.
- JMSExpiration of the request message is set to the JMSExpiration of the request

This processing is discussed in detail in "SOAP/JMS protocol listener processing" on page 66.

Asynchronous SOAP/JMS web service

From the perspective of connector processing, an asynchronous SOAP/JMS web service is one that follows a request-only path. If the SOAP/JMS protocol listener successfully receives and processes a request-only web service message, no JMS message containing a response is returned to the client. If a ReplyToQueue is configured and a fault occurs upon receipt of a JMS message, a fault message is returned to the web service client. In addition, if an ErrorQueue is specified in the SOAP/JMS listener, the fault message is archived there.

Event processing

The first step in implementing an event processing capability is exposing a business process -- a collaboration -- as a web service. You then publish this web service, in a UDDI registry, for example, and configure the connector to respond to web service clients that invoke the collaboration.

During event processing, the connector uses protocol listeners and the SOAP data handler to convert SOAP request messages from web service clients to business objects that can be manipulated by collaborations that have been exposed as web services. Protocol listeners play a crucial role in event processing.

Protocol listeners

Web Service requests may come over variety of transports, including HTTP, HTTPS, and JMS. The Web Services protocol listener monitors the arrival of such requests on its transport channel. There are three protocol listeners and corresponding channels:

- SOAP/HTTP protocol listener
- SOAP/HTTPS protocol listener
- SOAP/JMS protocol listener

Each of these consists of a thread that listens on its transport. When it receives a SOAP request message from a client, the listener registers the event with the protocol listener framework.

The protocol listener framework manages the protocol listeners, scheduling requests as resources are available. You configure the listeners and aspects of the protocol listener framework when you set values to connector-specific properties. Among the protocol listener framework properties you can configure are the following:

- **WorkerThreadCount** Total number of threads available to the protocol listener framework, which is the number of requests that it can process in parallel.
- **RequestPoolSize** Maximum number of requests that can be registered with the protocol listener framework. If it receives more than this maximum requests, it will no longer register new requests.

These two connector-specific properties control memory allocation in a way that prevents protocol listeners from clogging the connector with infinite web service events. The allocation algorithm is as follows: At any time, the connector can receive a total number of events equal to WorkerThreadCount + RequestPoolSize. It can process *WorkerThreadCount* number of requests in parallel.

You can plug additional protocol listeners into the protocol listener framework. For further information, see "Creating multiple protocol listeners" on page 103 and"Connector-specific configuration properties" on page 85.

SOAP/HTTP and SOAP/HTTPS protocol listener processing

The SOAP/HTTP(S) protocol listener consists of a thread that continuously listens for HTTP(S) requests from web service clients. The listener thread binds the host and port that are specified in the Host and Port connector-specific configuration (listener) properties. Another configuration property—RequestWaitTimeout—defines the interval during which the listener waits for a request before checking whether the connector has shut down.

Figure 24 illustrates SOAP/HTTP protocol listener processing for a synchronous operation.

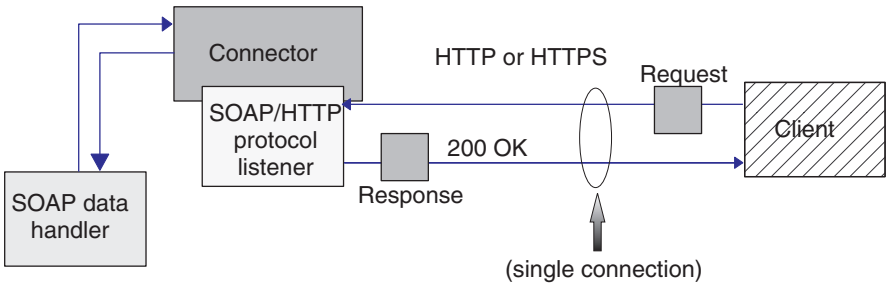


Figure 24. SOAP/HTTP protocol listener: synchronous event processing

Figure 25 shows SOAP/HTTP protocol listener processing for an asynchronous operation.

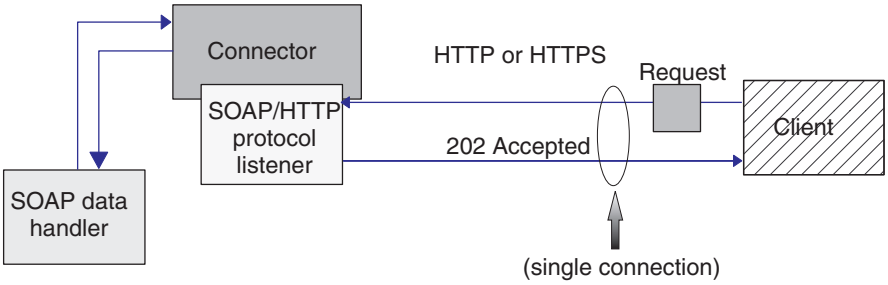


Figure 25. SOAP/HTTP protocol listener: asynchronous event processing

When a web services client initiates a SOAP/HTTP or SOAP/HTTPS request, it posts a SOAP request message to the URL of the SOAP/HTTP or SOAP/HTTPS listener. The client should use the HTTP POST method to invoke the protocol listener URL.

When an HTTP(S) request arrives, the listener registers the request with protocol listener framework, which schedules the event for processing as resources become available. The listener then extracts the protocol headers and the payload from the request.

Table 27 summarizes the order of precedence of rules used by the listener to determine the Charset, MmeType, ContentType and Content-Type header for inbound messages.

Table 27. SOAP/HTTP(s) protocol listener processing rules for inbound message

Order of Precedence	Charset	MimeType	ContentType	Content-Type header
1	Charset parameter value from the incoming HTTP message Content-Type header value	URLsConfiguration connector property value for this listener	Incoming HTTP message type/subtype value from the Content-Type header value	Incoming HTTP message Content-Type header
2	URLsConfiguration property value for this listener	SOAPDHMimeType connector property value		
3	If the type of the request message ContentType is `text` with any subtype (for example, `text/xml`, `text/plain`, etc.), default to ISO-8859-1. Otherwise, charset will not be used.	Default to ContentType		

As shown in Table 27:

- The protocol listener determines the Charset of the inbound message according to the following rules:
  1. The listener attempts to extract the Charset from the charset parameter of HTTP message Content-Type header value.

2. If no Charset value is obtained from the Content-Type header, then the protocol listener attempts to read the URLsConfiguration property value for this listener.
3. If a Charset value is not obtained using methods described in the previous steps, and if type of the message ContentType is `text` with any subtype (for example, `text/xml`, `text/plain`, etc.), the listener uses a default Charset value of ISO-8859-1. Otherwise, Charset value is not used.

- The listener determines the MimeType for the response message according to these rules:
  1. If you have configured the TransformationRules for the URL used by the incoming request message, and if the request ContentType matches the ContentType of a TransformationRule, then the listener uses the TransformationRule to extract the MimeType for conversion of the request message into a SOAP Request business object. The listener attempts to find the exact TransformationRule match based on the ContentType value (for example, `text/xml`) in the URLsConfiguration property for the requested URL.
  2. If that fails, the listener attempts to find a TransformationRule that applies to more than one ContentType under the request URL (for example */*).
  3. If there is no TransformationRule match for the MimeType, then the listener uses the SOAPDHMimeType connector configuration property as the MimeType value.
  4. If all previous steps fail to determine the MimeType, the value of ContentType will be used as the MimeType to invoke the SOAP data handler and convert the request message into a SOAP Request business object.
- The listener determines the ContentType by extracting type/subtype from the incoming HTTP message Content-Type header.
- The listener determines the Content-Type header from that of the incoming HTTP message Content-Type header.

If the collaboration is invoked asynchronously, the listener delivers the request business object to the integration broker and responds to the web services client with the HTTP status code `202 Accepted`. This concludes listener processing.

If it is a synchronous invocation, the listener invokes the collaboration synchronously. The collaboration responds with a SOAP Response business object.

Table 28 summarizes the order of precedence for rules used by the listener when determining the Charset, MimeType, ContentType, and Content-Type header for response messages.

Table 28. SOAP/HTTP(s) protocol listener processing rules for outbound synchronous response message

Order of Precedence	Charset	MimeType	ContentType	Content-Type header
1	Protocol ConfigMO Content-Type Header	MimeType property in the TLO	Protocol ConfigMO Content-Type header	Protocol ConfigMO Content-Type header
2	The Charset property value in the TLO	The request message MimeType, but only if the request and response ContentType match.	Request message ContentType	Construct Content-Type Header using ContentType and Charset

Table 28. SOAP/HTTP(s) protocol listener processing rules for outbound synchronous response message (continued)

3	The request message Charset, but only if the request and response ContentType match.	SOAPDHMimeType connector property value		
4	If the ContentType is `text/*`, default to ISO-8859-1. Otherwise, charset will not be used.	Use ContentType value as the MimeType		

As shown in Table 28:

- The listener determines the Charset for the response message according to these rules:
  1. If Charset is specified in the Response business object Protocol Config MO, its value is used.
  2. If there is no Charset value specified in the Response business object Protocol Config MO header and if the Request and Response business object are children of TLOs, the listener checks if Charset is specified in the TLO.
  3. If there is no Charset specified in the TLO, or if the Response business object is not a TLO, then if the response has the same ContentType as the request, the Charset of the request will be used for the response.
  4. If the previous steps fail to determine the response Charset value, and if the type portion of the message ContentType is `text` with a subtype of anything (for example, `text/xml`, `text/plain`, etc.), the listener uses a default Charset value of ISO-8859-1. Otherwise, the Charset value is not used.
- The listener determines the MimeType for the response message according to these rules:
  1. The TLO's MimeType attribute
  2. If the TLO MimeType attribute is missing, and if the request and response ContentType match, the listener uses the request MimeType for the response message.
  3. If the previous steps fail, then the listener uses the value of the SOAPDHMimeType connector property.
  4. Otherwise the listener uses the ContentType value as the MimeType.
- The listener determines the ContentType for the response message according to these rules:
  1. If the Content-Type header is specified in the Response business object Protocol Config MO, the type/subtype portion of the Content-Type header will used as the ContentType.
  2. If the Content-Type header is not specified in the Response business object Protocol Config MO, the listener constructs a Content-Type header using the determined ContentType and Charset (if the Charset was determined for the response message).

The listener processes the HTTP Protocol Config MO. It is the responsibility of collaboration to ensure that the header values passed in the HTTP Protocol Config MO are correct in the context of the request-response event. The listener populates standard headers and custom properties according to the following rules:

1. The listener will investigate each item of the HTTP Protocol Config MO in order to ignore special attributes (such as ObjectEventId).
2. Each non-empty header will be put on the outgoing message and additional processing (for example, the Content-Type header) may take place.

3. Please note that with the above approach, the listener may set non-standard headers on the message, but will not check that the message is logically or semantically correct.
4. If there are one or more custom properties in the HTTP Protocol Config MO UserDefinedProperties attribute, the listener will add them in the Entity Headers Section (the last headers section). For more on custom properties, see "User-defined properties for event processing" on page 33.

Note: Specifying any of the following headers in the HTTP Protocol Config MO is very likely to result in an incorrect HTTP message: Connection, Trailer, Transfer-Encoding, Content-Encoding, Content-Length, Content-MD5, Content-Range.

The listener then invokes the SOAP data handler to convert the Response business object returned by the Collaboration into a SOAP response message.

The listener delivers the response message to the web service client and includes a `200 OK` HTTP status code. If the collaboration returns a SOAP Fault business object, it is converted to a Fault message. This fault message is delivered to the web service client with a `500 Internal Server Error HTTP` code.

The listener then closes the connection and the thread that processed the event becomes available.

Unsupported SOAP/HTTP protocol listener processing features

The SOAP/HTTP protocol listener does not support the following:

- Caching: The protocol listener does not perform any caching functions as defined in HTTP specifications (RFC2616)
- Proxy: The protocol listener does not perform any proxy functions as defined in HTTP specifications (RFC2616).
- Persistent Connection: The protocol listener does not support persistent connections as defined in HTTP specifications (RFC2616). Instead, the protocol listener assumes that the scope of each HTTP connection is a single client request. and closes the connection when the service request is completed. The protocol listener does not attempt to reuse the connection across the service invocations.
- Redirections: The protocol listener does not support redirections.
- Large file transfer: The protocol listener cannot be used for large file transfers. Alternatively, you may consider passing large files by reference instead.
- State management: The protocol listener does not support the HTTP state management mechanism described by RFC2965.
- Cookies: The protocol listener does not support cookies.

SOAP/HTTPS listener processing using secure sockets

SOAP/HTTPS protocol listener processing is the same as that described in the SOAP/HTTP protocol listener processing section except that HTTPS uses secure sockets. For further information, see "SSL" on page 82.

SOAP/JMS protocol listener processing

The SOAP/JMS protocol listener consists of a thread that continuously listens on the InputQueue, which is the JMS destination for requests from web service clients.

The RequestWaitTimeout connector configuration property defines how long the listener will wait for a request before checking whether the connector has shut down.

Figure 26 shows SOAP/JMS protocol listener processing for a synchronous operation. The figure does not show JMS provider information.

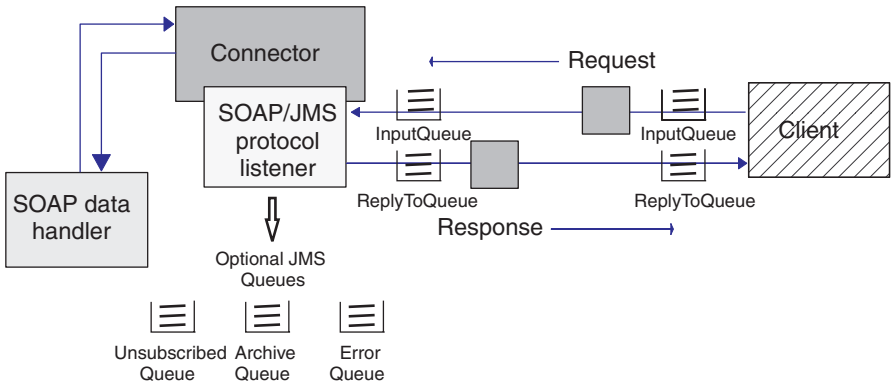


Figure 26. SOAP/JMS protocol listener: synchronous event processing

Figure 27 shows SOAP/JMS protocol listener processing for an asynchronous operation.

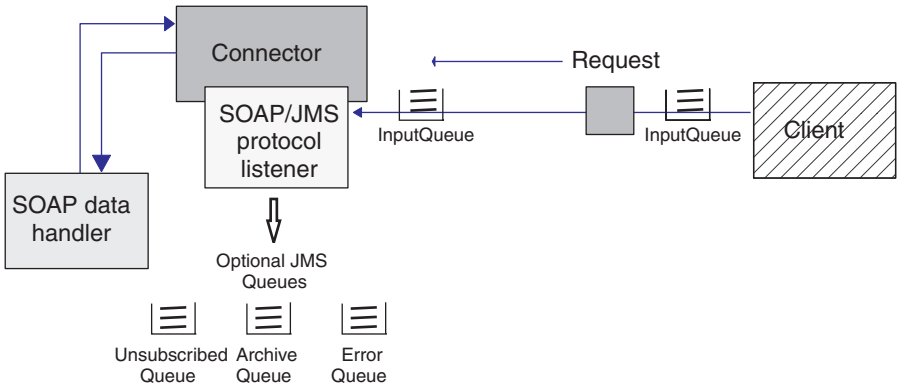


Figure 27. SOAP/JMS protocol listener: asynchronous event processing

Note: If the LookupQueueUsingJNDI configuration property is set to `true`, the SOAP/JMS protocol listener uses the JNDI to look up the queue. The JNDI properties are specified in connector properties. For further information, see "Connector and JMS" on page 80 and the JNDI-related properties in "Connector-specific configuration properties" on page 85.

When a web service client initiates a SOAP/JMS request, it sends a SOAP request message to the InputQueue on which the SOAP/JMS listener is listening. When it receives the SOAP request message from the InputQueue, the SOAP/JMS protocol listener registers the request with the protocol listener framework. The protocol listener framework schedules the request as and when resources are available.

Note: If the connector configuration property InDoubtEvents is set to `Reprocess`, the protocol listener framework will schedule JMS messages from the InProgressQueue before scheduling messages from the InputQueue.

The listener then dispatches this message—the body as well as the required JMS headers (JMSCorrelationID, JMSMessageID, JMSPriority, JMSExpiration, JMSDeliveryMode, JMSReplyTo, JMSTimeStamp, JMSType)— to the InProgressQueue. The protocol listener framework then registers the event.

The listener then reads the JMS message from the InProgressQueue, extracting the body of the message and the following headers:

- JMSDestination
- JMSRedelivered
- JMSCorrelationID
- JMSMessageID
- JMSPriority
- JMSExpiration
- JMSDeliveryMode
- JMSReplyTo
- JMSTimeStamp
- JMSType
- JMS Message Payload Type (not a header, but information from the message)

JMS Message Payload Type The listener will determine the payload type of the incoming message and store the information in the MessageType attribute of the JMS Protocol Config MO. The payload can be a TextMessage or BytesMessage. In TextMessage format, the listener invokes the data handler through String APIs with the web service request message extracted as a String. In the case of BytesMessage, the listener invokes the data handler via the Bytes Data Handler APIs with the web service request message extracted as a byte array.

Using the SOAPDHMimeType connector configuration property, the listener invokes the SOAP data handler to convert the request message into a SOAP Request business object. If errors occur during conversion and the JMSReplyTo JMS header is specified, the listener responds with a SOAP fault message, setting the faultcode to `Client` and the faultstring to `Cannot Parse`. The fault message provides no other detail.

The listener uses the object-level `cw_mo_jms` ASI of the SOAP Request business object returned by the data handler to determine the Protocol Config MO. Note that both the ASI and the Protocol Config MO are optional for event processing. If it finds a Protocol Config MO, the listener populates it with the JMS message headers extracted earlier. Table 42 shows the mapping between the attributes in the Protocol Config MO and the JMS message headers.

Table 29. JMS header-Protocol Config MO attribute mapping

Protocol Config MO attribute	JMS header name	Description
CorrelationID	JMSCorrelationID	The JMSCorrelationID header from the request message
MessageId	JMSMessageId	The JMSMessageID header from the request message

Table 29. JMS header-Protocol Config MO attribute mapping (continued)

Priority	JMSPriority	The JMSPriority header from the request message
Expiration	JMSExpiration	The JMSExpiration header from the request message
DeliveryMode	JMSDeliveryMode	The JMSDeliveryMode header from the request message
ReplyTo	JMSReplyTo	The JMSReplyTo header from the request message. The JMS API returns this header as JMSDestination, but the SOAP/JMS protocol listener returns the queue name.
Timestamp	JMSTimestamp	The JMSTimestamp header from the request message
Redelivered	JMSRedelivered	The JMSRedelivered header from the request message
Type	JMSType	The JMSType header from the request message
Destination	JMSDestination	The JMSDestination header from the request message
MessageType	na	The type of the request message payload. The value of this attribute is Text for TextMessage payloads, and Bytes for BytesMessage payloads.

If there are one or more custom properties in the SOAP/JMS Protocol Config MO UserDefinedProperties attribute, the listener will try to extract their values from the message and populate the UserDefinedProperties business object. For more on custom properties, see "User-defined properties for event processing" on page 33.

If the TLO (in the case of a non-TLO SOAP Request business object) is not subscribed by the integration broker, the listener logs an error. If the JMSReplyTo header is specified in the request message, the listener creates a SOAP fault message and places it on the JMSReplyTo queue. The faultcode is set to `Client` and the faultString is set to `Not subscribed to this message`. No other detail is provided in the fault message. If configured to do so, the listener also archives the original JMS request message including its JMS headers to the UnsubscribedQueue.

If the collaboration is invoked asynchronously, the listener delivers the Request business object to the integration broker. The listener then removes the message from the InProgressQueue. If configured to do so, the listener also archives the original JMS request message including its JMS headers to the ArchiveQueue.

If errors occur during asynchronous processing and JMSReplyTo is specified, the listener responds with a fault message. Its faultcode is set to `Server` and its faultstring is set to `Internal Error`. If configured to do so, the listener also archives the original JMS request message, including its JMS headers, to ErrorQueue.

If it is a synchronous invocation, the listener invokes the collaboration synchronously. The collaboration responds with a SOAP Response business object. The listener invokes the SOAP data handler to convert the Response business object returned by the Collaboration into a SOAP/JMS response message. The type

of the response payload depends on the value of the MessageType attribute in the JMS Protocol Config MO of the SOAP Response business object. If the MessageType is Text, the listener converts the SOAP Response business object into a String through String data handler APIs. If the MessageType is Bytes, the listener converts the SOAP Response business object into a bytes array via the Bytes data handler APIs. (The default message payload type is that of the corresponding synchronous request.) The listener delivers the response message to the ReplyTo queue (this is provided by the JMSReplyTo header on the original request message). The listener then sets the response message returned by the data handler as a TextMessage or BytesMessage (depending on the MessageType determined earlier), setting the headers shown in Table 30.

Table 30. Header values set by SOAP/JMS protocol listener in response message

JMS header name	Value
JMSCorrelationId	The JMSMessageId of the request message
JMSDeliveryMode	The JMSDeliveryMode of the request message
JMSPriority	The JMSPriority of the request message
JMSExpiration	The JMSExpiration of the request message
JMSRedelivered	The JMSRedelivered of the request message
JMSReplyTo	The JMSReplyTo of the request message
JMSTimestamp	The JMSTimestamp of the request message
JMSType	The JMSType of the request message

The listener will set JMS Custom Properties in the response message if they are present in the Response or Fault business objects' JMS Protocol Config MO UserDefinedProperties attribute.

If configured to do so, the listener then moves the original JMS message (request from the web service client), including its headers, from the InProgressQueue to the ArchiveQueue.

If errors occur and JMSReplyTo is specified, the listener responds with a fault message, and, if configured to do so, also archives the original JMS request message to the ErrorQueue.

Event persistence and delivery

Event persistence is protocol contingent:

- **SOAP/HTTP protocol listener** no persistence and therefore no guaranteed delivery
- **SOAP/HTTPS protocol listener** no persistence and therefore no guaranteed delivery
- **SOAP/JMS protocol listener** JMS queue event persistence and at-least-once guaranteed delivery. For more on the JMS queues, see "Connector-specific configuration properties" on page 85.

Event sequencing

The connector may deliver events in any sequence.

Event triggering

The event triggering mechanism depends on how the protocol listener is configured.

- **SOAP/HTTP protocol listener** Listening occurs over a ServerSocket for HTTP connection requests
- **SOAP/HTTPS protocol listener** Listening occurs over a secure ServerSocket layer for HTTPS connection requests
- **SOAP/JMS protocol listener** Listening occurs over the input queue for incoming JMS messages carrying web service requests.For more on the JMS queues, see "Connector-specific configuration properties" on page 85.

Note: Connector does not distinguish between Create or Update or Retrieve or Delete. All such events follow the same approach.

Event detection

Event detection is performed by each protocol listener. The event detection mechanism depends utterly on the transport and how you configure the connector-specific properties for each listener. For more on these properties, see "Connector-specific configuration properties" on page 85.

Event status

Event status is managed by the protocol listener and depends on the transport and also on how you configure the listener.

- **SOAP/HTTP protocol listener** HTTP is inherently non-persistent and synchronous in nature. Accordingly, event status is not maintained.
- **SOAP/HTTPS protocol listener** HTTP is inherently non-persistent and synchronous in nature. Accordingly, event status is not maintained.
- **SOAP/JMS protocol listener** JMS is a persistent transport. Event status is maintained using queues. For more on the JMS queues, see "Connector-specific configuration properties" on page 85.

Event retrieval

Event retrieval is managed by the protocol listener and depends on the transport and also on how you configure the listener.

- **SOAP/HTTP protocol listener** Events are retrieved by extracting HTTP requests from the socket.
- **SOAP/HTTPS protocol listener** Events are retrieved by extracting HTTP requests from the socket.
- **SOAP/JMS protocol listener** Events are retrieved using the JMS API. The JMS protocol listener retrieves events from the JMS input queue and moves them to the in-progress queue. For more on the JMS queues, see "Connector-specific configuration properties" on page 85.

Event archiving

Event archiving is managed by the protocol listener and depends on the transport and also on how you configure the listener.

- **SOAP/HTTP protocol listener** Because of the non-persistent and synchronous nature of the transport, archiving is not performed.
- **SOAP/HTTPS protocol listener** Because of the non-persistent and synchronous nature of the transport, archiving is not performed.

- **SOAP/JMS protocol listener** You can configure the connector to archive events into a JMS queues including those for unsubscribed events, successful events, and failed events. For more on the JMS queues, see "Connector-specific configuration properties" on page 85.

Event recovery

Event recovery is managed by the protocol listener and depends on the transport and also on how you configure the listener.

- **SOAP/HTTP protocol listener** Because of the non-persistent nature of the transport, event recovery is not performed.
- **SOAP/HTTPS protocol listener** Because of the non-persistent nature of the transport, event recovery is not performed.
- **SOAP/JMS protocol listener** JMS is a persistent transport. If the connector shuts down while events are being processed, they remain available in the InProgressQueue. You can configure the connector to process these events at startup, thereby enabling event recovery. The InDoubtEvents connector configuration property determines the event recovery mechanism.

  Note: The SOAP/JMS listener assures at-least once delivery to the integration broker. The listener cannot assure once and only once delivery. Also, events received by the listener may be delivered in any order to the integration broker.

  At startup, the JMS protocol listener first attempts to retrieve events from the InProgressQueue. What happens next is determined by the value you assign to the InDoubtEvents configuration property. The recovery scenarios are illustrated in table. For more on the JMS queues, see "Connector-specific configuration properties" on page 85.

Table 31. Header values set by SOAP/JMS protocol listener in response message

InDoubtEvents value	Event recovery processing
`FailOnStartup`	If it finds events in the InProgressQueue, the listener logs a fatal error and immediately shuts down.
Reprocess	If it finds events in the InProgressQueue, the listener processes those events first. The listener can trace the number of messages found in the InProgressQueue.
Ignore	Events in the InProgressQueue are ignored. The listener can trace the events found in the InProgressQueue and the ignoring of those events by the listener.
`LogError`	If it finds events in the InProgressQueue, the listener logs error and continues processing.

Request processing

You use the request processing capability of the connector to enable a collaboration to invoke a web service. The development tasks include using the WSDL ODA to generate a web services top-level object (TLO) and configuring a collaboration to deploy it. For further information, see Chapter 6, "Enabling collaborations for request processing," on page 141. You must also configure the connector and its request processing components: the protocol handler framework and protocol handlers.

At run time, the connector receives requests from the collaboration in the form of business objects. The business objects— SOAP Request, and optionally SOAP

Response and SOAP Fault business objects— are contained by the TLO generated by the WSDL ODA and issued by a collaboration that is configured to use web services. The TLO and its child business objects contain attributes and ASI that specify the processing mode (synchronous or asynchronous), the data handler mime type, which protocol handler to use, as well as the address of the target web service. The protocol handler uses this information to invoke an instance of the SOAP data handler, convert the Request business object to a SOAP request message, and invoke the target web service. If the mode is synchronous, the protocol handler again invokes the data handler to convert the response message into a SOAP Response business object and returns this to the collaboration.

In response to a SOAP request message, the connector can receive any of the following from the remote trading partner:

- A SOAP response message that contains data
- A SOAP response message that contains fault information

Protocol handlers play a key role in request processing.

Protocol handlers

A collaboration can invoke a web service over HTTP, HTTPS, or JMS transports. The connector has two protocol handlers and corresponding channels:

- A SOAP/HTTP-HTTPS protocol handler for invoking SOAP/HTTP and SOAP/HTTPS web services
- A SOAP/JMS protocol handler for invoking SOAP/JMS web services

The protocol handler framework manages the protocol handlers, loading them at startup time. When the connector receives a Request business object, the request thread (note that each collaboration request comes in a thread of its own) invokes the protocol handler framework to process the request.

The protocol handler framework reads the TLOs Handler attribute ASI to determine which protocol handler to use. Applying a series of rules (see "SOAP/HTTP-HTTPS protocol handler processing" on page 74 and "SOAP/JMS protocol handler processing" on page 77), the protocol handler invokes a data handler to convert the Request business object into a SOAP request message. The protocol handler packages the request message into the transport—HTTP(S) or JMS— message. If it finds SOAPAction ASI in the Request business object, the protocol handler adds this to the request message header.

The protocol handler then reads the Destination attribute of the Request business object Protocol Config MO to determine the target address. The protocol handler then invokes the target web service with the request message.

Reading the `ws_mode TLO` ASI, the protocol handler determines whether the processing mode is synchronous or asynchronous. If this ASI is set to `asynch`, the protocol handler processing is completed. Otherwise the protocol handler waits for a response message. If a response message arrives, the protocol handler extracts the protocol headers and the payload. It then invokes the data handler (indicated by the MimeType TLO attribute) to convert the message into a Response or Fault business object. Again using the Protocol Config MO, the protocol handler sets the protocol headers in the business object. The protocol handler then returns the Response or Fault business object to the collaboration.

Depending on connector configuration, there may be one or more protocol handlers plugged into the connector. Connector-specific properties allow you to configure protocol handlers.

SOAP/HTTP-HTTPS protocol handler processing

The SOAP/HTTP(S) protocol handler performs as described in "Protocol handlers" on page 73 with exceptions noted in this section. Figure 28 shows the SOAP/HTTP-HTTPS protocol handler for a synchronous operation.

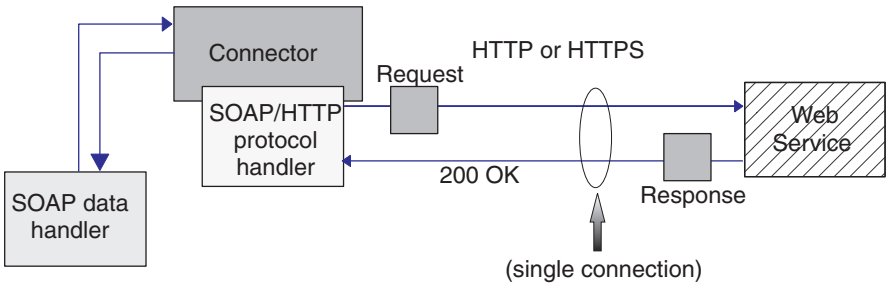


Figure 28. SOAP/HTTP-HTTPS protocol handler: synchronous request processing

Figure 29 shows the SOAP/HTTP-HTTPS protocol handler for an asynchronous request process

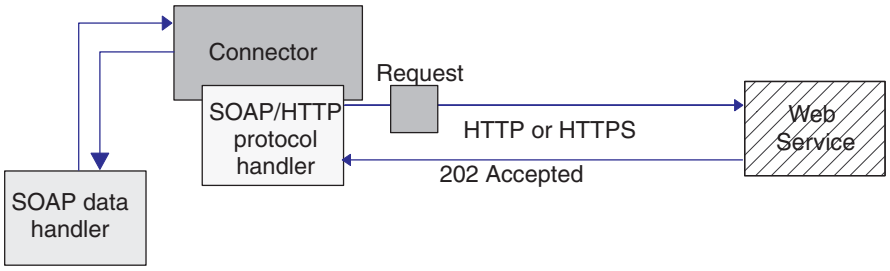


Figure 29. SOAP/HTTP-HTTPS protocol handler: asynchronous request processing

Note: This section describes SOAP/HTTP protocol handling only.

The SOAP/HTTP-HTTPS protocol handler uses the object-level ASI (`cw_mo_http`) of the SOAP Request business object to determine the Protocol Config MO. The SOAP/HTTP-HTTPS protocol handler determines the URL of the target web service by reading the Destination attribute in the HTTP Protocol Config MO. If the URL is missing or is incomplete, the protocol handler fails the service call. For further information on the HTTP Protocol Config MO and its attributes, see "HTTP Protocol Config MO for request processing" on page 47.

The SOAP/HTTP-HTTPS protocol handler invokes the web service using the SOAP request message returned by the SOAP data handler. If HTTP Proxy connector configuration properties are specified, the SOAP/HTTP(S) protocol handler behaves accordingly. If a response is returned, the SOAP/HTTP(S) protocol handler reads it.

Table 32 summarizes the order of precedence of rules used by the SOAP/HTTP-HTTPS protocol handler to determine the Charset, MimeType, ContentType, and ContentType header for outgoing request messages.

Table 32. SOAP/HTTP-HTTPS protocol handler processing rules for outbound messages

Order of Precedence	Charset	MimeType	ContentType	ContentType header
1	Protocol Config MO's Content-Type Header	MimeType property in TLO attribute	Protocol Config MO's Content-Type Header	Protocol Config MO's Content-Type Header
2	Charset property in TLO attribute	Default to ContentType	Default to `text/xml`	Construct Content-Type header using ContentType and Charset
3	If the ContentType is `text/*`, default to ISO-8859-1. Otherwise, charset will not be used.			

As shown in Table 32:

- The SOAP/HTTP-HTTPS protocol handler determines the Charset for the response message according to these rules:
  1. If specified in the Request business object Protocol Config MO headers, the Charset value is used.
  2. If Charset is not determined by the previous step, the protocol handler attempts to extract the Charset from the TLO attribute.
  3. If the operation described in the previous step is unsuccessful, the table is used to determine the Charset:

Table 33. Default request processing Charsets

ContentType	Default Charset
`text/*`	ISO-8859-1 For further information, see RFC2616,
`application/*`	No default
All others	No default

  4. If Charset was determined by the previous step, the Charset is set on the data handler.
  5. The data Handler is invoked with Stream or Byte array APIs, depending on the data structure needed for writing out the request.
- The SOAP/HTTP-HTTPS protocol handler determines the MimeType for the request according to these rules:
  1. The TLO MimeType attribute.
  2. If the TLO MimeType attribute is missing, the protocol handler uses the ContentType to determine the MimeType.
- The SOAP/HTTP-HTTPS protocol handler determines the ContentType for the request message according to these rules:
  1. If the Content-Type header is specified in the Request business object Protocol Config MO, the type/subtype of the header will be used as ContentType.
  2. Otherwise, the handler uses the default ContentType: `text/xml`.

- The SOAP/HTTP-HTTPS protocol handler determines the Content-Type header for the request message according to these rules:
  1. If the Content-Type header is specified in the Request business object Protocol Config MO, its value is set on the outgoing message.
  2. If the Content-Type header is not specified in the Request business object Protocol Config MO, the listener constructs a Content-Type header using the ContentType and Charset parameter (if the Charset was determined for the request message).

Table 34 summarizes the order of precedence for rules used by the handler when determining the Charset, MimeType, ContentType, and ContentType header for response messages.

Table 34. SOAP/HTTP(s) protocol handler processing rules for inbound synchronous response message

Order of Precedence	Charset	MimeType	ContentType	ContentType header
1	Charset parameter value from the incoming HTTP message Content-Type header value	Message TransformationMap child business object in the Request business object's Protocol Config MO	Incoming HTTP message type/subtype value from the Content-Type header value	Incoming HTTP message Content-Type header
2	Message TransformationMap child business object in the Request business object's Protocol Config MO	The request message MimeType, but only if the request and response ContentType match.		
3	The request message Charset, but only if the request and response ContentType match.	MimeType property in TLO		
4	Charset property in TLO.	Default to ContentType		
5	If the Content-Type is `text/*`, default to ISO-8859-1. Otherwise, Charset is not used.			

As shown in Table 34:

- The protocol handler determines the Charset of the synchronous response message according to the following rules:
  1. If the Charset parameter is set in the Content-Type header of the incoming response message, the protocol handler uses the Charset value to set on the data handler.
  2. If there is no Charset value in the response message header, then the protocol handler attempts to read the collaboration-defined Charset from the TLO Request Protocol Config MO MessageTranformationMap.
  3. If there is no Charset value specified in the TLO, or if there is no TLO, then if the response has the same ContentType as the request, the Charset of the request will be used for the response.
  4. If the previous step fails to yield a Charset value, then the protocol handler attempts to read the TLO Charset attribute.

5. If a Charset value is not obtained using methods described in the previous steps, and if type of the message ContentType is `text` with any subtype (for example, `text/xml`, `text/plain`, etc.),default ISO-8859-1. Otherwise, charset value is not used.

- The protocol handler determines the MimeType of the synchronous response message according to the following rules:
  1. The protocol handler first attempts to extract the MimeType from the TLO Request Protocol Config MO's MessageTransformationMap. Specifically, the protocol handler tries to find an exact ContentType match in the MTM to extract MessageTransformationRule and then use the MimeType property value from it. Otherwise, the protocol handler looks for a MessageTransformationRule that applies to more than one ContentType (ContentType is */*).
  2. If the MimeType is not determined by using a MessageTransformationMap, the protocol handler uses the request MimeType for that of the response if and only if the request and response ContentTypes match.
  3. If the MimeType cannot be extracted using the previous steps, the protocol handler uses the MimeType attribute of the TLO. or the default MimeType, if available to the protocol handler.
  4. If all previous steps fail, the protocol handler uses the ContentType to set the MimeType.
- The handler determines the ContentType by extracting type/subtype from the incoming HTTP message Content-Type header.

The handler processes the HTTP Protocol Config MO. It is the responsibility of the collaboration to ensure that the header values passed in the HTTP Protocol Config MO are correct in the context of the request-response event. The handler populates standard headers and custom properties according to the following rules:

1. The handler will investigate each item of the HTTP Protocol Config MO in order to ignore special attributes (such as ObjectEventId).
2. Each non-empty header will be put on the outgoing message and additional processing (for example, the Content-Type header) may take place.
3. Please note that with the above approach, the handler may set non-standard headers on the message, but will not guarantee that the message is logically or semantically correct.
4. If there are one or more custom properties in the HTTP Protocol Config MO UserDefinedProperties attribute, the handler will add them in the Entity Headers Section (the last headers section). For more on custom properties, see "User-defined properties for request processing" on page 48.

Note: Specifying any of the following headers in the HTTP Protocol Config MO is very likely to result in incorrect HTTP messages: Connection, Trailer, Transfer-Encoding, Content-Encoding, Content-Length, Content-MD5, Content-Range.

SOAP/JMS protocol handler processing

The SOAP/JMS protocol handler performs as described in "Protocol handlers" on page 73 with exceptions noted in this section.

Note: If the LookupQueueUsingJNDI configuration property is set to `true`, the SOAP/JMS protocol handler uses the JNDI to look up the destination queue. The JNDI properties are specified in connector properties. For further information, see "Connector and JMS" on page 80 and the JNDI-related properties in "Connector-specific configuration properties" on page 85.

The SOAP/JMS protocol handler creates a JMS transport message using the body of the web service request message returned by the SOAP data handler and with JMS headers set as shown in Table 35.. The type of the response payload depends on the value of MessageType attribute in the JMS Protocol Config MO of the SOAP Request business object. If the MessageType is Text, the handler converts the SOAP Request business object into a String via the String data handler APIs. If the MessageType is Bytes, the handler converts the SOAP Request business object into a bytes array via the Bytes data handler APIs. (The default message payload type is TextMessage.).

Table 35. Header values set by SOAP/JMS protocol handler in request message

JMS header name	Default value if not set in SOAP/JMS Protocol Config MO
JMSPriority	4
JMSExpiration	0
JMSDeliveryMode	PERSISTENT
JMSReply	
JMSCorrelationId	
JMSRedelivered	
JMSTimestamp	
JMSType	

If the target web service is invoked asynchronously, the JMSReplyTo header is not set. Otherwise (for synchronous processing), the SOAP/JMS protocol handler sets the JMSReplyTo header. Using the ReplyToQueue configuration property, the SOAP/JMS protocol handler obtains the JMSDestination—the return destination for a response or fault from the target web service— and sets it on the JMSReplyTo header on the JMS transport message.

Figure 30 shows SOAP/JMS protocol handler processing for a synchronous request operation.

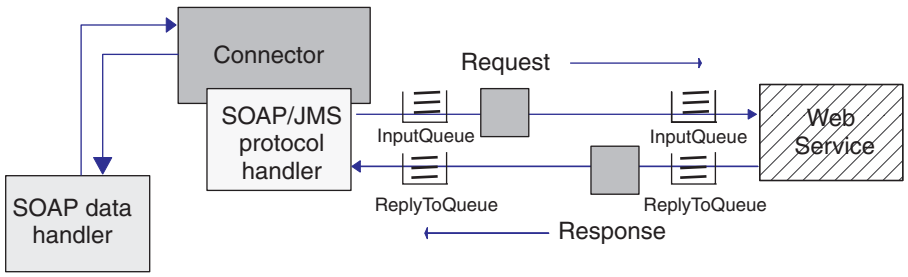


Figure 30. SOAP/JMS protocol handler: synchronous request processing

Figure 31 shows SOAP/JMS protocol handler processing for an asynchronous request operation.

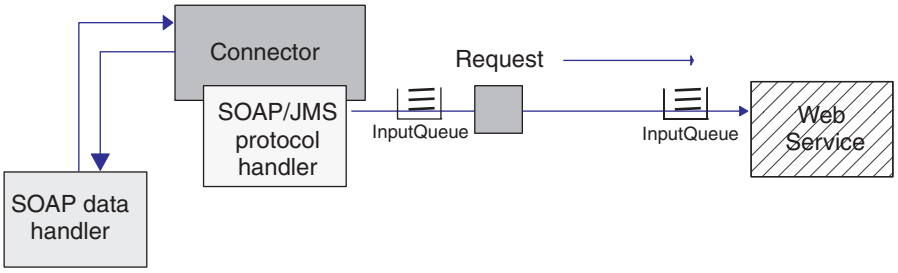


Figure 31. SOAP/JMS protocol handler: asynchronous request processing

The SOAP/JMS protocol handler uses object-level ASI (`cw_mo_jms`) of the SOAP Request business object to determine the Protocol Config MO. The Destination attribute of the Protocol Config MO gives the queue name of the target web service. If JNDI is enabled, the SOAP/JMS protocol handler obtains the JMSDestination for the SOAP request message by looking up the JNDI object. Otherwise it uses the Destination attribute in the SOAP Protocol Config MO.

If the response does not arrive in the interval specified in the ResponseWaitTimeout property, the SOAP/JMS protocol handler fails the collaboration request. On arrival of the SOAP response (or fault) message, the SOAP/JMS protocol handler extracts the JMS headers and payload for conversion by the SOAP data handler. The handler determines the payload type of the incoming message and stores the information in the MessageType attribute of JMS Protocol Config MO. The payload can be a TextMessage or BytesMessage. In TextMessage format, the handler invokes the data handler through the String APIs with the web service response message extracted as a String. In the case of a BytesMessage, the handler invokes the data handler via the Bytes data handler APIs with the web service response message extracted as a byte array. The SOAP/JMS protocol handler then sets the SOAP Response (or Fault) business object in the TLO, using the Protocol Config MO in the Response (or Fault) business object to map the JMS headers. Table 36 shows this mapping.

Table 36. Protocol Config MO—JMS header attribute mapping for response during synchronous request processing

Protocol Config MO attribute	**JMS header name**	**Description**
Destination	JMSDestination	The JMSDestination header from the response message.
MessageId	JMSMessageId	The JMSMessageId header from the response message
Priority	JMSPriority	The JMSPriority header from the response message
Expiration	JMSExpiration	The JMSExpiration header from the response message
DeliveryMode	JMSDeliveryMode	The JMSDeliveryMode header from the response message
ReplyTo	JMSReplyTo	The JMSReplyTo header from the response message. The JMS API returns this header as JMSDestination, but the SOAP/JMS protocol listener returns the queue name.

Table 36. Protocol Config MO—JMS header attribute mapping for response during synchronous request processing (continued)

CorrelationId	JMSCorrelationId	The JMSCorrelationId header from the response message
Redelivered	JMSRedelivered	The JMSRedelivered header from the response message
TimeStamp	JMSTimeStamp	The JMSTimeStamp header from the response message
Type	JMSType	The JMSType header from the response message
MessageType	n/a	The type of the response message payload. The value of this attribute is Text for TextMessage payload; Bytes for BytesMessage payload.

The SOAP/JMS protocol handler then returns the TLO to the collaboration.

Connector and JMS

Note: This section assumes that you are familiar with JMS and JNDI, especially how JMS works. For further information, refer to your JMS and JNDI source documentation.

The connector can expose collaborations as SOAP/JMS web services as well as enable collaborations to invoke SOAP/JMS web services. The requirements for using SOAP/JMS with the web services connector are as follows:

1. You have installed and configured your JMS service provider.
2. You have installed and configured your JNDI.
3. Your JMS provider supports JMS API version 1.0.2.
4. All required jar files are in the classpath of the connector. (See your JMS provider documentation to determine all required jar files.)
5. All required libraries are in the path of the connector. (See your JMS provider documentation to determine all required libraries.)

JNDI

For SOAP/JMS, the connector uses JNDI to lookup the connection factory using JNDI context. During initialization, the connector reads the JNDI connector-specific property to connect to JNDI. If you do not configure this property, you will be unable to use SOAP/JMS. You can specify following JNDI connector specific properties:

- JNDIProviderURL
- InitialContextFactory
- JNDIConnectionFactoryName
- CTX_ObjectFactories
- CTX_ObjectFactories
- CTX_StateFactories
- CTX_URLPackagePrefixes
- CTX_DNS_URL
- CTX_Authoritative

- CTX_Batchsize
- CTX_Referral
- CTX_SecurityProtocol
- CTX_SecurityAuthentication
- CTX_SecurityPrincipal
- CTX_SecurityCredentials
- CTX_Language
- LookupQueuesUsingJNDI

Refer to your JNDI documentation for guidance in specifying these properties. To use SOAP/JMS with the connector, the following JNDI connector-specific properties are required:

- **JNDIProviderURL** Set this property to the URL of the JNDI Service provider. For the value of this property, refer to your JNDI provider documentation.
- **InitialContextFactory** Set this property to the fully qualified class name of the factory class that will create the JNDI initial context. For the value of this property, refer to your JNDI provider documentation. Make sure that this class and its dependencies are in the classpath of the connector.
- **JNDIConnectionFactoryName** Set this property to the JNDI name of the Connection factory to lookup (using JNDI context). Make sure that this name can be looked up using the JNDI.

If you set LookupQueuesUsingJNDI to `true`, make sure all the queues used by the connector can be looked up using JNDI.

Exposing collaborations as SOAP/JMS web services

To expose collaborations as SOAP/JMS web services, you must use the SOAP/JMS protocol listener. Using the SOAP/JMS protocol listener requires that you specify JNDI connector properties.

Your JMS provider configuration should reflect the requirements of the SOAP/JMS protocol listener. Make sure all the queues required by the SOAP/JMS protocol listener are defined by your JMS service provider. Be sure to check your JMS provider documentation— the task of defining queues varies by provider. You must define six queues for the SOAP/JMS protocol listener. You must set the queue names in SOAP/JMS listener configuration properties and, if you have set JNDI " LookupQueuesUsingJNDI to `true`, you also must specify the JNDI names of the queues in the SOAP/JMS listener configuration properties.

You should specify the queues names as the values of the following SOAP/JMS Listener configuration properties:

- InputQueue
- InProgressQueue
- ArchiveQueue
- UnsubscribedQueue
- ErrorQueue
- ReplyToQueue

InputQueue and InProgressQueue are required properties. Make sure that you have correctly configured these queues.

ArchiveQueue, UnsubscribedQueue and ErrorQueue are optional properties. These queues are used to archive web service requests. If you plan to use any of these properties, make sure you have configured the corresponding JMS queues correctly. When defining these queues with your JMS provider, you should carefully specify the capacity of these queues.

Collaborations invoking SOAP/JMS web services

To enable collaborations to invoke SOAP/JMS web services, you use the SOAP/JMS protocol handler. The SOAP/JMS protocol handler requires that you specify JNDI connector properties. Work with your web service provider to determine the JMS and JNDI requirements.

To invoke SOAP/JMS web services, the connector requires that the value of the Destination attribute in the SOAP/JMS Protocol Config MO be set to the input queue o f the target web service. If you have set JNDI " LookupQueuesUsingJNDI to `true`, you must specify the JNDI name of the input queue.

If you are invoking request-reply web services, you must work with your web service provider to determine the requirements for the ReplyTo queue. Make sure that the ReplyTo queue is defined. Also make sure that you have specified the name of the ReplyTo queue in the ReplyToQueue configuration property of the SOAP/JMS protocol handler. If JNDI " LookupQueuesUsingJNDI is set to `true`, the value of the ReplyToQueue configuration property should give the JNDI name of this queue.

It is important to note that, unlike protocol listeners, protocol handlers are not pluggable to the web services connector. As a result, the connector uses the same ReplyTo queue for all the request-reply web services that the connector invokes.

SSL

This section discusses the how the connector implements an SSL capability. For background information, see your SSL documentation. This section assumes a familiarity with SSL technology.

JSSE

The connector can expose collaborations as SOAP/HTTPS web services and enable collaborations to invoke SOAP/HTTPS web services. The connector uses JSSE to provide support for HTTPS and SSL. IBM JSSE is shipped with the connector. To enable this capability, make sure you have the following entry in the `java.security` file that is among the files installed with the connector:

```
security.provider.5=com.ibm.jsse.IBMJSSEProvider
```

Note that `java.security` is located in the `$ProductDir\lib\security` directory of your connector installation. The connector uses the value of the JavaProtocolHandlerPackages connector property to set the system property java.protocol.handler.pkgs. Note that for the IBM JSSE that is shipped with the connector, the value of this property should be set to `com.ibm.net.ssl.internal.www.protocol`. The JavaProtocolHandlerPackages configuration property defaults to this value. However, if your system has a java.protocol.handler.pkgs system property with a non-empty value, the connector would overwrite it only if the JavaProtocolHandlerPackages connector property is also set.

During initialization, the connector disables all anonymous cipher suites supported by JSSE.

KeyStore and TrustStore

To use SSL with the connector, you must set up keystores and truststores. No tool is provided to set up keystores, certificates, and key generation. You must use third party software tools to complete these tasks.

SSL Properties

You can specify the following SSL connector-specific properties:

- SSLVersion
- SSLDebug
- KeyStore
- KeyStoreAlias
- KeyStorePassword
- TrustStore
- TrustStorePassword

Note that these properties apply to a connector instance. The same set of SSL property values are used by all of the SOAP/HTTPS protocol listeners plugged into the connector and by the SOAP/HTTP-HTTPS protocol handler for each connector instance. For further information on HTTPS/SSL setup, see Appendix E, "Configuring HTTPS/SSL," on page 221.

Exposing collaborations as SOAP/HTTPS web services

When you expose collaborations as SOAP/HTTPS web services, you use the SOAP/HTTPS protocol listener. To use the SOAP/HTTPS protocol listener, you must specify SSL connector-specific properties. The values you assign to these properties should reflect your SSL requirements:

- **SSLVersion** Make sure that the SSLVersion you want to use is supported by JSSE.
- **KeyStore** Because the SOAP/HTTPS protocol listener acts as a server in SSL communications, you must specify the keystore. The listener uses the keystore specified in the SSL " KeyStore configuration property. The value of this property must be the complete path to your keystore file. Make sure that the keystore has key pair (private key and public key) for the connector. The alias of the private key should be specified as the SSL " KeyStoreAlias property. You must specify the password required to access the keystore as the SSL " KeyStorePassword property. Also make sure that the password required to access keystore and the private key (in the keystore) are same. Finally, you must distribute the digital certificate of the connector to your web service clients so that they can authenticate the connector.
- **TrustStore** If you want the SOAP/HTTPS protocol listener to authenticate web service clients, you must activate client authentication. You do this by setting the SSL " UseClientAuth property to `true`. You must also specify:
  - the location of your truststore as the value of the SSL " TrustStore configuration property
  - the password required to access the truststore as the value of the SSL " TrustStorePassword property

  Make sure that your truststore contains the digital certificate of your web service clients. Digital certificates used by your Web Service clients may be self-signed

or issued by CA. Note that if your truststore trusts the root certificate of the CA, JSSE will authenticate all the digital certificates issued by that CA.

For further information on HTTPS/SSL setup, see Appendix E, “Configuring HTTPS/SSL,” on page 221.

Collaborations invoking SOAP/HTTPS web services

To enable collaborations to invoke SOAP/HTTPS web services, you use the SOAP/HTTP-HTTPS protocol handler. If you are using SSL with the SOAP/HTTP-HTTPS protocol handler, you must specify SSL connector-specific properties. The values you assign to these properties should reflect the HTTPS/SSL requirements of your web services provider:

- **SSLVersion** Make sure that the SSLVersion you want to use is supported by your web service provider and by JSSE.
- **TrustStore** Because the SOAP/HTTP-HTTPS protocol handler acts as a client in SSL communications, you must set up a truststore. The handler uses the truststore specified in the SSL -> Truststore configuration property. The value of this property must be the complete path to your truststore file. You must specify the password required to access the truststore in the SSL -> TrustStorePassword property. Make sure that your truststore contains the digital certificate of your web service provider. Digital certificates used by your web service provider may be self-signed or they may be issued by CA. Note that if your truststore trusts the root certificate of the CA, JSSE will authenticate all the digital certificates issued by that CA.
- **KeyStore** If your web service provider requires client authentication, you must set up a keystore. The SOAP/HTTP-HTTPS protocol handler uses the keystore specified in the SSL ” KeyStore configuration property. This value must be the complete path to your keystore file. Make sure that keystore has a key pair (private key and public key) configured for the connector. The alias of the private key must be specified in the SSL ” KeyStoreAlias property. The password required to access the keystore must be specified in the SSL ” KeyStorePassword property. Finally, make sure that the password required to access the keystore and the private key (in the keystore) are the same. You must distribute the connector’s digital certificate to your web service provider for authentication.

For further information on HTTPS/SSL setup, see Appendix E, “Configuring HTTPS/SSL,” on page 221.

Configuring the connector

After using the Installer to install the connector files to your system, you must set the standard and application-specific connector configuration properties.

Setting configuration properties

Connectors have two types of configuration properties: standard configuration properties and connector-specific configuration properties. You must set the values of these properties using System Manager (SM) before running the connector.

Standard configuration properties

Standard configuration properties provide information that all connectors use. See Appendix A, “Standard configuration properties for connectors,” on page 169 for documentation of these properties. The table below provides information specific to this connector about configuration properties in the appendix.

Property	Description
CharacterEncoding	This connector does not use this property.
Locale	Because this connector has not been internationalized, you cannot change the value of this property. See release notes for the connector to determine currently supported locales.

Because this connector supports only InterChange Server (ICS) as the integration broker, the only configuration properties relevant to it are for ICS.

You must set at least the following standard connector configuration properties:

- AgentTraceLevel
- ApplicationName
- ControllerTraceLevel
- DeliveryTransport

Connector-specific configuration properties

Connector-specific configuration properties provide information needed by the connector agent at runtime. Connector-specific properties also provide a way of changing static information or logic within the connector agent without having to recode and rebuild the agent.

Table 37 lists the connector-specific configuration properties. See the sections that follow for explanations of the properties. Note that some of the properties contain other properties. The + character indicates the entry's position in the property hierarchy.

Note: If you do not intend to use the SOAP/JMS protocol listener or SOAP/JMS protocol handler with the connector, be sure to delete SOAP/JMS-related connector-specific properties or to leave them blank.

Table 37. Connector-specific configuration properties

Name	Possible values	Default value	Required
ConnectorType	*Any valid connector type*	`WebService`	Yes
DataHandlerMetaObjectName	*Data handler meta-object name*	`MO_DataHandler_ Default`	Yes
JavaProtocolHandlerPackages	*Valid Java protocol handler packages*	`com.ibm.net.ssl. internal.www.protocol`	No
ProtocolHandlerFramework	*This is a hierarchical property and has no value*	None	No
+ProtocolHandlers	*This is a hierarchical property and has no value*		No
++SOAPHTTPHTTPSHandler	*This is a hierarchical property. For information on its sub-properties, see "SOAPHTTPHTTPSHandler" on page 87.*		Yes
++SOAPJMSHandler	*This is a hierarchical property. For information on its sub-properties, see "SOAPJMSHandler" on page 88.*		
ProtocolListenerFramework	*This is a hierarchical property and has no value.*		No
+WorkerThreadCount	*An integer of 1 or greater that gives the number of available listener threads.*	`10`	No

Table 37. Connector-specific configuration properties (continued)

Name	Possible values	Default value	Required
+RequestPoolSize	*Integer greater than WorkerThreadCount that gives the resource pool size.*	`20`	No
+ProtocolListeners	*This is a hierarchical property and has no value*		
++Listener1	*Uniquely named protocol listener*		Yes
+++Protocol	`soap/http`, `soap/https`, `soap/jms`		Yes
+++SOAPDHMimeType	*Any valid mime type of a SOAP data handler*	`xml/soap`	
+++ListenerSpecific	*Properties unique to or required by the listener* See "ListenerSpecific" on page 90.		
ProxyServer	*This is a hierarchical property and has no value*		No
+HttpProxyHost	*Host name for the HTTP proxy server*		No
+HttpProxyPort	*Port number for the HTTP proxy server*	`80`	No
+HttpNonProxyHosts	*HTTP host(s) requiring direct connection*		No
+HttpsProxyHost	*Host name for the HTTPS proxy server*		No
+HttpsProxyPort	*Port number for the HTTPS proxy server*	`443`	No
+HttpsNonProxyHosts	*HTTPS host(s) requiring direct connection*		No
+SocksProxyHost	*Socks proxy server name*		No
+SocksProxyPort	*Socks proxy server port*		No
+HttpProxyUsername	*Http proxy server username*		No
+HttpProxyPassword	*Http proxy server password*		No
+HttpsProxyUsername	*Https proxy server username*		No
+HttpsProxyPassword	*Https proxy server password*		No
SSL	*This is a hierarchical property and has no value*		No
+SSLVersion	`SSL`, `SSLv2`, `SSLv3`, `TLS`, `TLSv1`	SSL	No
+SSLDebug	`true`, `false`	`false`	No
+KeyStoreType	*Any valid keystore type*	`JKS`	No
+KeyStore	*Path to KeyStore file.*		No
+KeyStorePassword	*Password for private key in KeyStore*		No
+KeyStoreAlias	*Alias for key pair in KeyStore*		No
+TrustStore	*Path to TrustStore file*		No
+TrustStorePassword	*Password for TrustStore*		No
+UseClientAuth	`true` `false`	`false`	No
WSCollaborations	*This is a hierarchical property creating by the WSDL Configuration Wizard and has no value* See "WSCollaborations" on page 100.		
+Collaboration1	*This is a hierarchical property and has no value*		
++CollaborationPort1	*Name of the collaboration port*		Yes
+++WebServiceOperation1	*This is a hierarchical property and has no value*		Yes
++++BodyName	*Name of web service method; must be valid XML element name*		Yes

Table 37. Connector-specific configuration properties (continued)

Name	Possible values	Default value	Required
++++BodyNS	*Namespace of web service method; must be valid XML namespace*		Yes
++++BOName	*Name of Request business object for operation*		Yes
++++Mode	`synch asynch`	asynch	No
JNDI	*This is a JMS-related hierarchical property and has no value*		No
+LookupQueuesUsingJNDI	`true false`	false	No
+JNDIProviderURL	*Valid JNDI URL*		No
+InitialContextFactory	*Name of factory class for initial context*		No
+JNDIConnectionFactoryName	*Name of connection factory to look up using JNDI context*.		No
+CTX_ObjectFactories +CTX_*properties*	*Properties specifying additional information about security and object lookup in the JNDI context*		N

ConnectorType: If this property is set to `WebService`, when binding the collaboration port, System Manager displays the connector as a web services connector. Otherwise it is displayed as a normal connector.

Default = `WebService`.

DataHandlerMetaObjectName: This is the name of the meta-object that the data handler uses to set configuration properties.

Default = `MO_DataHandler_Default`.

JavaProtocolHandlerPackages: The value of this property gives the Java Protocol Handler packages. The connector uses the value of this property to set the system property java.protocol.handler.pkgs.

Default = `com.ibm.net.ssl.internal.www.protocol`.

ProtocolHandlerFramework: The Protocol Handler Framework uses this property to load and configure its protocol handlers. This is a hierarchical property and has no value.

Default = none.

ProtocolHandlers: This hierarchical property has no value. Its first-level children represent discrete protocol handlers.

Default = none.

SOAPHTTPHTTPSHandler: The name of a SOAP/HTTP-HTTPS protocol handler. Note that this is a hierarchical property. Unlike listeners, protocol handlers may not be duplicated, and there can be only one handler for each protocol. Table 38 below shows the sub-properties for the SOAP/HTTP-HTTPS protocol handler. The + character indicates the entry's position in the property hierarchy.

Table 38. SOAP/HTTP-HTTPS protocol handler configuration properties

Name	Possible values	Default value	Required
++SOAPHTTPHTTPSHandler	This is a hierarchical property and has no value.		Yes
+++Protocol	*The kind of protocol the handler is implementing. For SOAP/HTTP and SOAP/HTTPS, the value is* `soap/http`. **Note:** If you do not specify a value for this property, the connector will not initialize this protocol handler.		Yes
+++HTTPReadTimeout	*A SOAP/HTTP-specific property that specifies the timeout interval (in milliseconds) while reading from the remote host (web service).If this property is not specified or if set to 0, the SOAP/HTTP protocol handler blocks indefinitely while reading from the remote host.*	`0`	No

Figure 32 shows the properties as displayed in Connector Configurator.

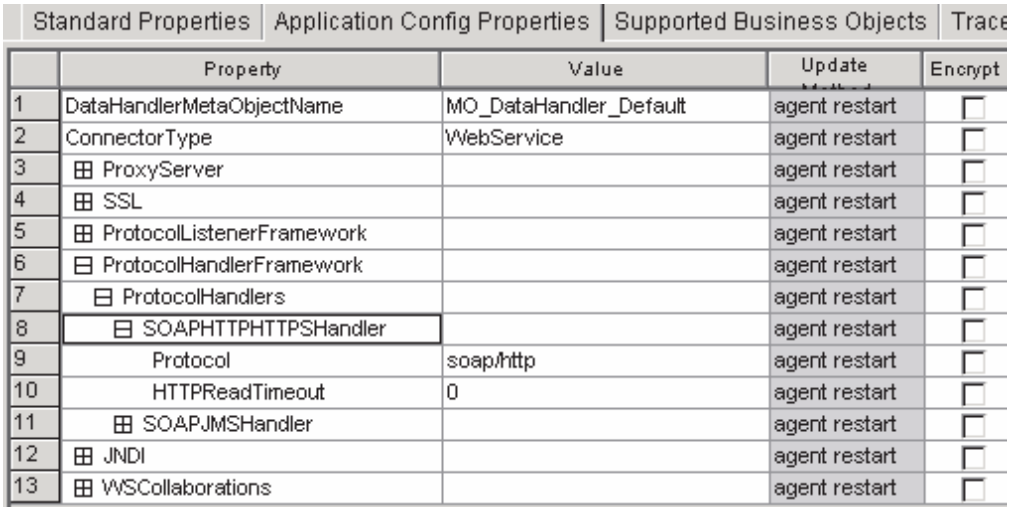


Figure 32. SOAP/HTTP-HTTPS protocol handler properties

SOAPJMSHandler: The name of a SOAP/JMS protocol handler. Note that this is a hierarchical property. Unlike listeners, protocol handlers may not be duplicated, and there can be only one handler for each protocol. Table 39 below shows the sub-properties for the SOAP/JMS protocol handler. The + character indicates the entry's position in the property hierarchy.

Table 39. SOAP/JMS protocol handler configuration properties

Name	Possible values	Default value	Required
++SOAPJMSHandler	This is a hierarchical property and has no value.		Yes
+++Protocol	*The kind of protocol the handler is implementing. For SOAP/JMS, the value is* `soap/jms`. **Note:** If you do not specify a value for this property, the connector will not initialize this protocol handler.		Yes

Table 39. SOAP/JMS protocol handler configuration properties (continued)

Name	Possible values	Default value	Required
+++ResponseWaitTimeout	*This is a JMS protocol handler-specific property that specifies the timeout interval (in milliseconds) that the protocol handler waits on ReplyToQueue for synchronous request processing. If the response does not arrive during this interval, the handler fails the collaboration request. If this property is not specified or if set to 0, the protocol handler waits on ReplyToQueue indefinitely.*	`0`	No
+++ReplyToQueue	*This is a required JMS protocol handler-specific property that names the ReplyTo queue. For synchronous request processing, the handler sets the JMSReplyTo field to this JMS destination. If LookupQueuesUsingJNDI =* `true`*, the SOAP/JMS protocol handler looks up this queue using JNDI.*	none	Yes

Figure 33 shows the properties as displayed in Connector Configurator.

Standard Properties | Application Config Properties | Supported Business Objects | Trace/Log Files | Da

	Property	Value	Update	Encrypt	Description
1	DataHandlerMetaObjectName	MO_DataHandler_Default	agent restart	☐	
2	ConnectorType	WebService	agent restart	☐	
3	⊞ ProxyServer		agent restart	☐	
4	⊞ SSL		agent restart	☐	
5	⊞ ProtocolListenerFramework		agent restart	☐	
6	⊟ ProtocolHandlerFramework		agent restart	☐	
7	⊟ ProtocolHandlers		agent restart	☐	
8	⊞ SOAPHTTPHTTPSHandler		agent restart	☐	
9	⊟ SOAPJMSHandler		agent restart	☐	
10	Protocol	soap/jms	agent restart	☐	
11	ResponseWaitTimeout	0	agent restart	☐	
12	ReplyToQueue		agent restart	☐	
13	⊞ JNDI		agent restart	☐	
14	⊞ WSCollaborations		agent restart	☐	

Figure 33. SOAP/JMS protocol handler properties

ProtocolListenerFramework: The protocol listener framework uses this property to load protocol listeners. This is a hierarchical property and has no value.

WorkerThreadCount: This property, which must be an integer of 1 or greater, establishes the number of protocol listener worker threads available to the protocol listener framework. For further information, see “Protocol listeners” on page 61.Default = `10`.

RequestPoolSize: This property, which must be an integer greater than WorkerThreadCount, sets the resource pool size of the protocol listener framework. The framework can process a maximum of WorkerThreadCount + RequestPoolSize requests concurrently.

Default = `20`.

ProtocolListeners: This is a hierarchical property and has no value. Each first-level child of this property represents a discrete protocol listener.

Listener1: The name of a protocol listener. There may be multiple protocol listeners. Note that this is a hierarchical property. You can create multiple instances of this property and create additional, uniquely named listeners. When doing so, you can change the listener-specific properties but not the protocol property. The names of multiple listeners must be unique. Possible names (not values): `SOAPHTTPListener1`, `SOAPHTTPSListener1`, `SOAPJMSListener1`

Protocol: This property specifies the protocol this listener is implementing. Possible values: `soap/http`, `soap/https`, `soap/jms`.

Note: If you do not specify a value for this property, the connector will not initialize this protocol listener.

SOAPDHMimeType: The SOAP data handler mime type to use for the requests received by this listener.

Default = `xml/soap`

ListenerSpecific: Listener specific properties are unique to, or required by, the specified protocol listener. For example, the HTTP listener has a listener-specific property Port, which represents the Port number on which Listener monitors requests. Table 40 summarizes the HTTP-HTTPS listener specific properties. The + character indicates the entry's position in the property hierarchy.

Table 40. SOAP/HTTP and SOAP/HTTPS protocol listener-specific configuration properties

Name	Possible values	Default value	Required
+++SOAPHTTPListener1	*Unique name of an HTTP protocol listener. This is a child of the ProtocolListenerFramework -> ProtocolListeners hierarchical property. There can be multiple listeners: you may plug-in additional HTTP listeners by creating another instance of this property and its hierarchy.*		Yes
++++Protocol	`soap/http` if SOAP/HTTP protocol listener `soap/https` if SOAP/HTTPS protocol listener **Note:** If you do not specify a value for this property, the connector will not initialize this protocol listener.		Yes
++++SOAPDHMimeType	`xml/soap`	`xml/soap`	No
++++BOPrefix	*The value of this property is passed to the data handler.*		No
++++Host	*The listener will listen at the IP address specified by value of this property. If* Host *is not specified, it defaults to localhost. Note that you may either specify a host name (DNS name) or an IP address for the machine on which the listener is running. A machine may have multiple IP addresses or multiple names.*	`localhost`	No
++++Port	*The port on which the listener listens for requests. If unspecified, the port defaults to* `80` *for SOAP/HTTP and* `443` *for SOAP/HTTPS.If you clone the listener within a connector, then the combination of Host and Port properties is unique or the listener may be unable to bind to the port to accept requests.*	`80` for SOAP/HTTP listener `443` for SOAP/HTTPS listener	No

Table 40. SOAP/HTTP and SOAP/HTTPS protocol listener-specific configuration properties (continued)

Name	Possible values	Default value	Required
++++SocketQueueLength	*Length of the queue (socket queue) for incoming connection requests. Specifies how many incoming connections can be stored at one time before the host refuses connections. The maximum queue length is operating system dependent.*	`5`	No
++++RequestWaitTimeout	*The time interval in milli-seconds that the listener thread will block on the host and port while waiting for web service requests to arrive. If it receives a web service request before this interval, the listener will process it. Otherwise the listener thread checks whether the connector shutdown flag is set. If it is set, the connector will terminate. Otherwise it will continue to block for RequestWaitTimeout interval. If this property is set to 0, it will block for ever. If unspecified, it defaults to 60000ms.*	`60000` (ms)	No
++++HTTPReadTimeout	*The time interval in milli-seconds that the listener will be blocked while reading a web service request from a client. If this parameter is set to 0, the listener indefinitely blocks until it receives the entire request message.*	`0`	No
++++HttpAsyncResponseCode	*The HTTP response code for asynchronous requests to the listener:* 200 (OK) 202 (ACCEPTED)	`202` (ACCEPTED)	No
++++URLsConfiguration	*This is a hierarchical property and has no value. It contains 1 or more configurations for URLs supported by this listener and, optionally, mime type and charset values. Note that this is child property of* ProtocolListenerFramework->ProtocolListeners->SOAPHTTPListener1 *hierarchical property. If this property is not specified, the listener assumes default values.*	ContextPath: / Enabled: true Data handler MimeType: equal to the ContentType of the request Charset: NONE. For further information, see "SOAP/HTTP and SOAP/HTTPS protocol listener processing" on page 62.	No
+++++URL1	This is a hierarchical property and has no value. Its children provide the *name of the URL supported by this listener. There can be multiple supported URLs. Note that you can plug in additional URLs by cloning this property and its hierarchy.*		No

Table 40. SOAP/HTTP and SOAP/HTTPS protocol listener-specific configuration properties (continued)

Name	Possible values	Default value	Required
++++++ContextPath	*The URI for the HTTP requests received by the listener. This value must be unique among ContextPath values under the URLsConfiguration property. Otherwise the connector will log an error and fail to start. ContextPath is case sensitive. However it may contain protocol, host name and port which are case-insensitive. If protocol is specified in ContextPath, it should be http. If host is specified, it should be equal to the value of the Host listener property. If port is specified, it should be equal to the value of Port listener property.*		No
++++++Enabled	*The value of this property determines if the parent URL hierarchical property is enabled for the connector.*	`True`	No
++++++TransformationRules	This is a hierarchical property and has no value. *It holds one or more transformation rules.*		
+++++++TransformationRule1	This is a hierarchical property and has no value. *It holds the transformation rule.*		No
++++++++ContentType	*The value of this property specifies the ContentType of the incoming request for which special handling (data handler mime type or charset) should be applied. If ContentType is not specified by the TransformationRule*N *hierarchical property, the connector logs a warning message and ignore*s *the TransformationRule*N *property. Specifying the special value `*/*` for this property enables the protocol listeners to apply this rule to any ContentType. Note that if a listener finds more than one rule for the same context path that shares a ContentType, the listener logs an error and fails to initialize.*		No
++++++++MimeType	*The mime type to use when calling a data handler to process requests of the specified ContentType.*		No
++++++++Charset	*Charset to use when transforming the request of the specified ContentType into a business object.*		No

Figure 34 shows the properties as displayed in Connector Configurator.

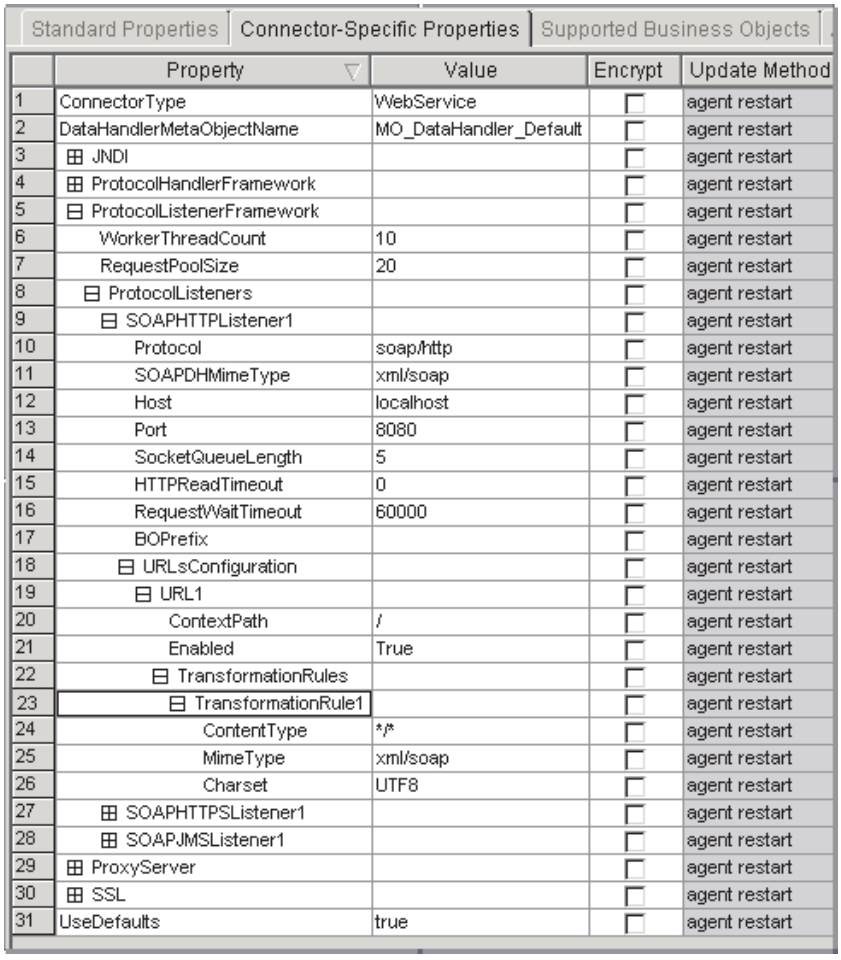


Figure 34. SOAP/HTTP protocol listener properties

Table 41 summarizes the SOAP/JMS protocol listener-specific properties. The + character indicates the entry's position in the property hierarchy.

Table 41. SOAP/JMS protocol listener-specific configuration properties

Name	Possible values	Default value	Required
+++SOAPJMSListener1	*Unique name of a JMS protocol listener. This is a child of the ProtocolListenerFramework -> ProtocolListeners hierarchical property. There can be multiple listeners: you may plug-in additional JMS listeners by creating another instance of this property and its hierarchy.*		Yes
++++Protocol	`soap/jms`		Yes
++++SOAPDHMimeType	`xml/soap`	`xml/soap`	No
++++BOPrefix	*The value of this property is passed to the data handler specified by SOAPDHMimeType property.*		No

Table 41. SOAP/JMS protocol listener-specific configuration properties (continued)

Name	Possible values	Default value	Required
++++RequestWaitTimeout	*This property sets the time interval that the SOAP/JMS listener thread blocks the InputQueue while waiting for a web service request. If it receives a web service request within this interval, the listener processes it. If it does not receive the request within this interval, the listener thread first checks if the connector shutdown flag is set. If the connector shutdown flag is set, the connector will terminate. Otherwise it will continue to block for RequestWaitTimeout interval. If this property is set to 0, it will block indefinitely.*	`60000` milliseconds	No
++++SessionPoolSize	*Maximum number of sessions that can be allocated for a given listener and its worker threads. The minimum number of sessions (and default) is 2. For larger session pool sizes, the connector requires more memory.*	2	No
++++InputQueue	*This property gives the name of the input queue that the listener polls for inbound messages from web services. If LookupQueuesUsingJNDI = `true`, the listener looks up this queue using JNDI and the value of the InputQueue property is set to the jndiDestinationName attribute of the jms:address element of the SOAP/JMS binding. The jms:address element is specified in the wsdl:port section of the WSDL document. If during WSDL generation you select the SOAP/JMS listener, System Manager automatically creates the jndiDestinationName attribute using the value of this property. If LookupQueueUsingJNDI = `false`, then System Manager creates the jmsProviderDestinationName attribute instead.*		Yes
++++InProgressQueue	*This property gives the name of the in-progress queue. The listener sends copies of inbound messages from the InputQueue to InProgressQueue. If LookupQueuesUsingJNDI = `true`, the listener looks up this queue using JNDI.*		Yes
++++ArchiveQueue	*This property gives the name of the archival queue. The listener sends copies of successfully processed messages from the InProgressQueue to ArchiveQueue. If LookupQueuesUsingJNDI = `true`, the listener looks up this queue using JNDI.*		No
++++UnsubscribedQueue	*This property gives the name of the unsubscribed queue. The listener sends copies of unsubscribed messages from the InProgressQueue to UnsubscribedQueue. If LookupQueueUsingJNDI = `true`, the listener looks up this queue using JNDI.*		No
++++ErrorQueue	*This property gives the name of the error queue. The listener sends copies of failed messages to the ErrorQueue. If LookupQueueUsingJNDI = `true`, the listener looks up this queue using JNDI.*		No

Table 41. SOAP/JMS protocol listener-specific configuration properties (continued)

Name	Possible values	Default value	Required
++++InDoubtEvents	*This property specifies how to handle messages in the InProgressQueue that are not fully processed due to unexpected connector termination. It can take one of the following values:* • `FailOnStartup` *Log an error and immediately shutdown* • `Reprocess` *Process the remaining messages in the InProgressQueue* • `Ignore` *Disregard any messages in the in-progress queue* • `LogError` *Log an error but do not shutdown*	`Ignore`	No
++++ReplyToQueue	*This property gives the name of the ReplyTo queue. The WSDL Configuration Wizard reads this property and writes it to the WSDL document. If this property is not specified, the utility does not create a ReplyTo JMS header in the SOAP/JMS binding in the WSDL document. (The listener does not use this property.) If JNDI properties are specified and LookupQueueUsingJNDI =* `false`*, the WSDL Generation Utility still create JNDI specific attributes in the WSDL document. Note that these JNDI-specific attributes are required because the SOAP/JMS binding does not provide any way to specify a ReplyTo attribute without JNDI. Though JNDI lookup for the InputQueue is not required, JNDI-specific properties are required for the ReplyTo queue. If the WSDL utility does not find JNDI-specific properties, the utility cannot create a ReplyTo attribute in the SOAP/JMS binding.*		
++++ JMSVendorURI	*A string that uniquely identifies the JMS implementation and that corresponds to the jmsVendorURI attribute of the jms:address element of the SOAP/JMS binding. The jms:address element is specified in wsdl:port section of the WSDL document. The listener does not use this property.*		No

Figure 35 shows the properties as displayed in Connector Configurator.

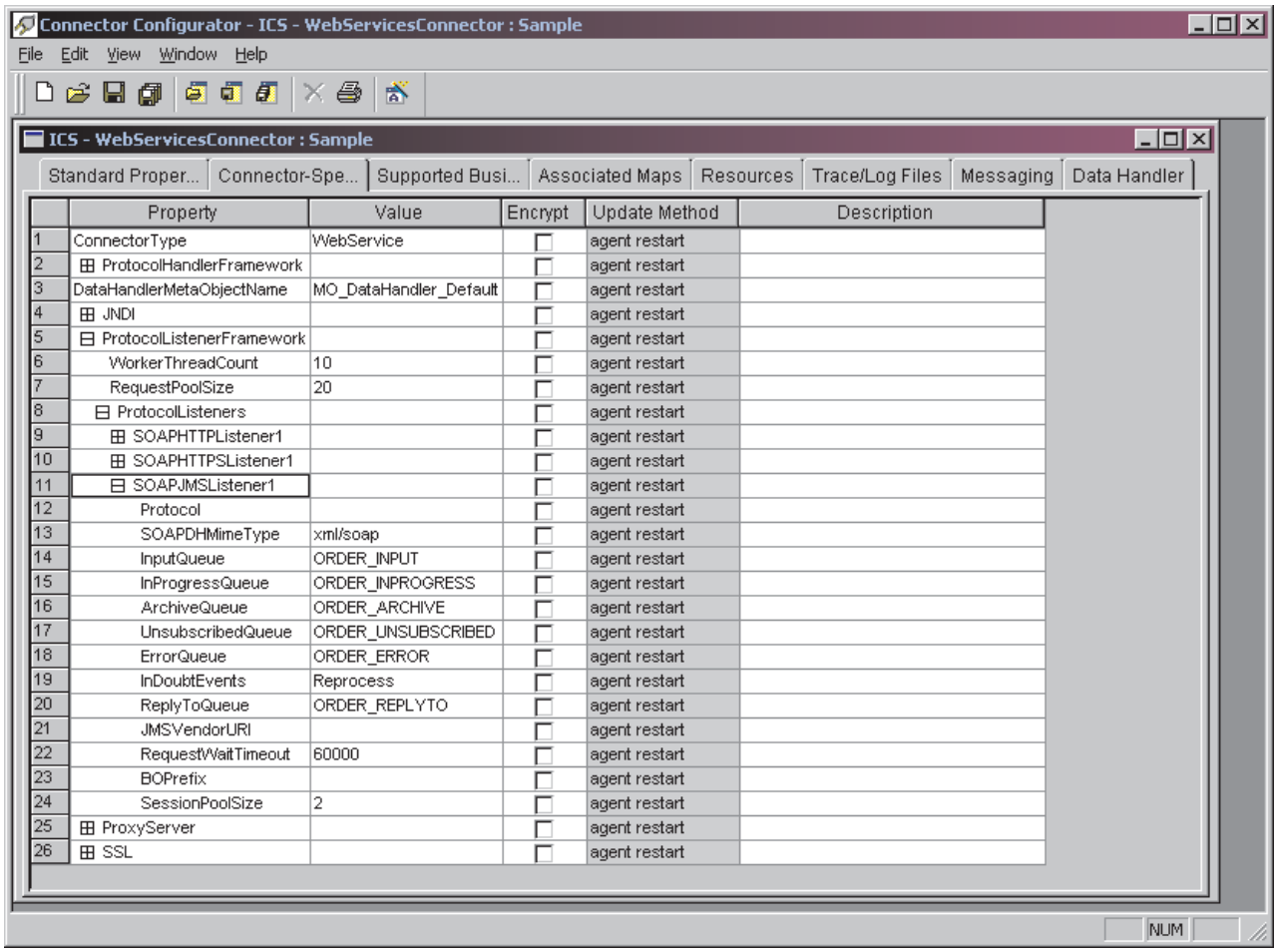


Figure 35. SOAP/JMS protocol listener properties

Note: Make sure that queue names specified in following properties are unique:

- InputQueue
- InProgressQueue
- ArchiveQueue
- UnsubscribedQueue
- ErrorQueue

ProxyServer: Configure the values under this property when the network uses a proxy server. This is a hierarchical property and has no value. The values specified under this property are used by the SOAP/HTTP/HTTPS protocol handlers.

Figure 36 shows the ProxyServer properties as displayed in Connector Configurator and discussed below.

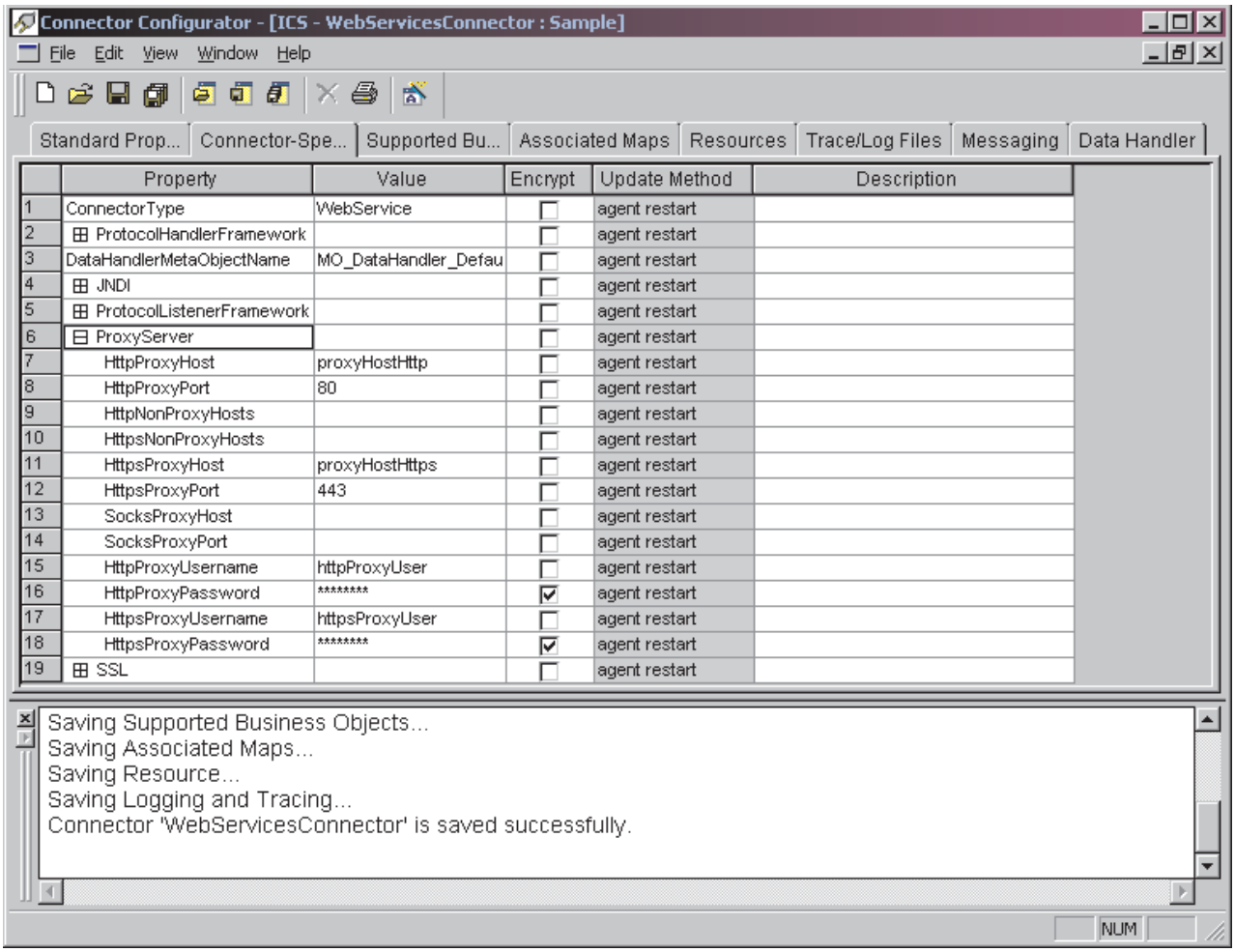


Figure 36. ProxyServer properties

HttpProxyHost: The host name for the HTTP proxy server. Specify this property if the network uses a proxy server for HTTP protocol.

Default = none

HttpProxyPort: The port number that the connector uses to connect to the HTTP proxy server.

Default = 80

HttpNonProxyHosts: The value of this property gives one or more hosts (for HTTP) that must be connected not through the proxy server but directly. The value can be a list of hosts, each separated by a "|".

Default = none

HttpsProxyHost: The host name for the HTTPS proxy server.

Default = none

HttpsProxyPort: The port number that the connector uses to connect to the HTTPS proxy server.

Default = 443

HttpsNonProxyHosts: The value of this property gives one or more hosts (for HTTPS) that must be connected not through the proxy server but directly. The value can be a list of hosts, each separated by a "|".

Default = none

SocksProxyHost: The host name for the Socks Proxy server. Specify this property when the network uses a socks proxy.

Note: The underlying JDK must support socks.

Default = none

SocksProxyPort: The port number to connect to the Socks Proxy server. Specify this property when the network uses a socks proxy.

Default = none

HttpProxyUsername: The username for the HTTP proxy server. If the destination for the web service request is an HTTP URL and you specify ProxyServer ->HttpProxyUsername, the SOAP HTTP/HTTPS protocol handler creates a Proxy-Authorization header when authenticating with the proxy. The handler uses the CONNECT method for authentication.

The proxy-authentication header is base64 encoded and has the following structure:
Proxy-Authorization: Basic
Base64EncodedString

The handler concatenates the username and the password property values, separated by a colon (:), to create the base64 encoded string.

Default = none

HttpProxyPassword: The password for the HTTP proxy server. For more on how this value is used, see "HttpProxyUsername."

Default = none

HttpsProxyUsername: The username for the HTTPS proxy server. If the destination for the web service request is an HTTPS URL and you specify ProxyServer ->HttpsProxyUsername, the SOAP HTTP/HTTPS protocol handler creates a Proxy-Authorization header for authentication with the proxy. The handler concatenates the HttpsProxyUsername and HttpsProxyPassword configuration property values, separated by colon (:), to create the base64 encoded string.

Default = none

HttpsProxyPassword: The password for the HTTPS proxy server. For more on how this value is used, see "HttpsProxyUsername."

Default = none

SSL: Specify values under this property to configure SSL for the connector. This is a hierarchical property and has no value.

Figure 37 shows the SSL properties as displayed in Connector Configurator and discussed below.

Standard Properties | Application Config Properties | Supported Business Objects | Trace/Log Files | D:

	Property	Value	Update	Encrypt	Description
1	DataHandlerMetaObjectName	MO_DataHandler_Default	agent restart	☐	
2	ConnectorType	WebService	agent restart	☐	
3	⊞ ProxyServer		agent restart	☐	
4	⊟ SSL		agent restart	☐	
5	SSLVersion	SSL	agent restart	☐	
6	SSLDebug	False	agent restart	☐	
7	KeyStoreType	JKS	agent restart	☐	
8	KeyStore		agent restart	☐	
9	KeyStorePassword		agent restart	☐	
10	KeyStoreAlias		agent restart	☐	
11	TrustStore		agent restart	☐	
12	TrustStorePassword		agent restart	☐	
13	UseClientAuth	False	agent restart	☐	
14	⊞ ProtocolListenerFramework		agent restart	☐	
15	⊞ ProtocolHandlerFramework		agent restart	☐	
16	⊞ JNDI		agent restart	☐	
17	⊞ WSCollaborations		agent restart	☐	

Figure 37. SSL properties

SSLVersion: The SSL version to be used by the connector. For further information, see IBM JSSE documentation for the supported SSL versions.

Default = `SSL`

SSLDebug: If value of this property is set to `true`, the connector sets the value of the`javax.net.debug` system property to `true`. IBM JSSE uses this property to turn on the trace facility. For further information, refer to IBM JSSE documentation.

Default = `false`

KeyStoreType: The value of this property gives the type of the KeyStore and TrustStore. For further information, see IBM JSSE documentation for valid keystore types.

Default = `JKS`

KeyStore: This property gives the complete path to keystore file. If KeyStore and/or KeyStoreAlias properties are not specified, KeyStorePassword, KeyStoreAlias, TrustStore, TrustStorePassword properties are ignored. The connector will fail to startup if it cannot load the keystore using the path specified in this property. The path must be the complete path to the keystore file.

Default = None

KeyStorePassword: This property gives the password for the private key in the Keystore.

Default = None

KeyStoreAlias: This property gives the alias for the key pair in the KeyStore. SOAP/HTTPS listeners use this private key from the KeyStore. Also, the SOAP/HTTP-HTTPS protocol handler uses this alias from the KeyStore when invoking web services that require client authentication. The property must be set to a valid JSSE alias.

Default = None

TrustStore: This property gives the complete path to the TrustStore. TrustStore is used for storing the certificates that are trusted by the connector. TrustStore must be of the same type as KeyStore. You must specify the complete path to the TrustStore file.

Default = None

TrustStorePassword: This property gives the password for the Truststore.

Default = None

UseClientAuth: This property specifies whether SSL client authentication is used. When it is set to `true`, SOAP/HTTPS listeners use client authentication.

Default = `false`

WSCollaborations: This property is created automatically when you expose a collaboration object as a web services and is used for non-TLOs. This is a hierarchical property and has no value. Each first-level child of this property represents a collaboration exposed as a web service. For information on the tools used to automatically create these properties, see Chapter 7, "Exposing collaborations as web services," on page 143.

Note: If you delete a collaboration or its port in System Manager, the connector will not automatically delete the properties representing the collaboration. You must delete these properties using Connector Configurator.

Figure 38 shows WSCollaborations properties as displayed in Connector Configurator and discussed below.

Figure 38. WSCollaborations properties

Collaboration1: This property names the collaboration object that is exposed as web service via this connector. This is a hierarchical property and has no value. There can be multiple such properties, one for each of collaboration object that is exposed as a web service. Each first-level child of this property represents a port of this collaboration object.

CollaborationPort1: This property names the collaboration port. This is a hierarchical property and has no value. There can be multiple such properties, one for each of the ports of this collaboration that are bound to the connector. Each first- level child of this property represents a web services operation.

WebServiceOperation1: This property represents a web services operation for the collaboration object. This is a hierarchical property and has no value. There may be one or more such properties, one for each of the web services operation defined by the user at the time of WSDL document generation.

BodyName: This property gives the name of the web service method and must be a valid XML element name.

Default = none

BodyNS: This property gives the namespace of the web service method and must be a valid XML namespace.

Default = none

BOName: This property gives the name of the Request business object for this operation.

Default = none

Mode: This property specifies the processing mode for the operation. It it is set to `synch`, the connector synchronously invokes the collaboration. Otherwise and by default, the connector asynchronously invokes the collaboration as a request only operation.

Default = asynch

JNDI: The connector maintains one set of JNDI (Java Naming and Directory Interface) provider properties that are used by the SOAP/JMS protocol handler and JMS protocol listener when connecting to JNDI. This is a hierarchical property and has no value. The connector uses JNDI to lookup the JMS connection factory object. Note that the WSDL Configuration Wizard uses this property when generating SOAP/JMS bindings.

Figure 39 shows JNDI properties as displayed in Connector Configurator and discussed below.

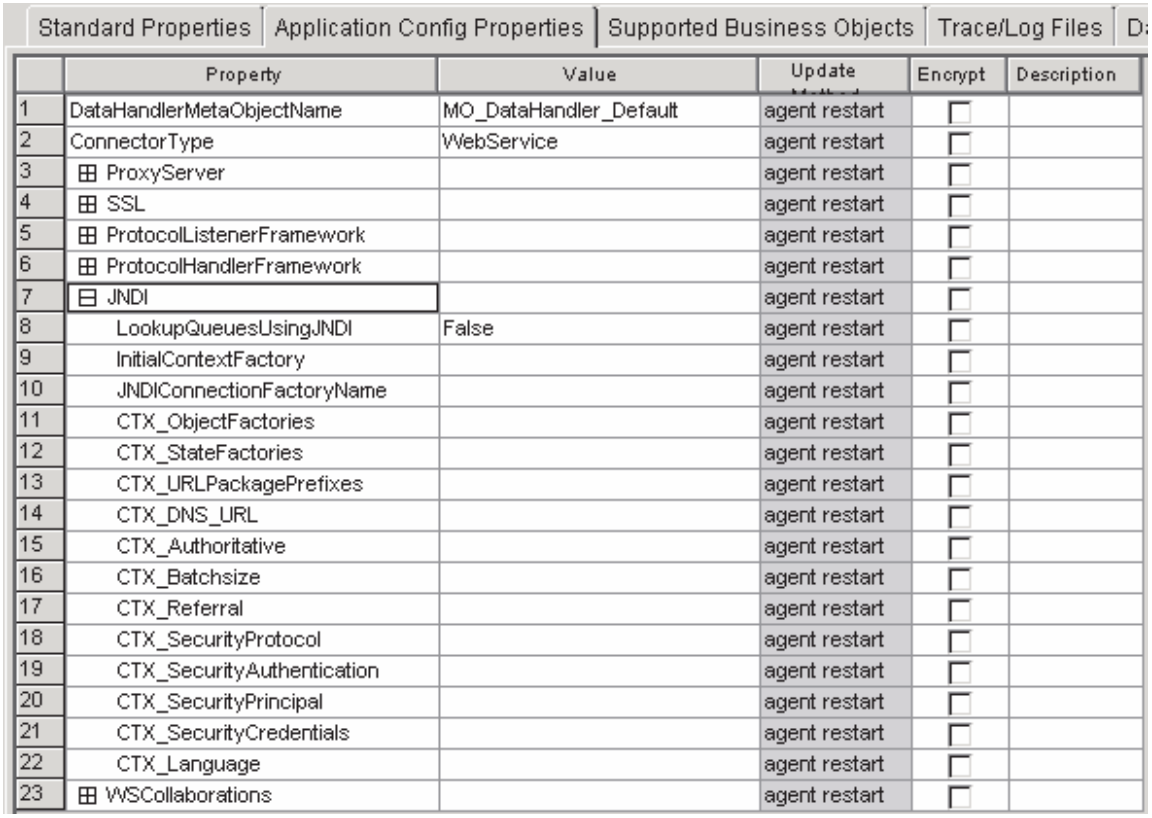


Figure 39. JNDI properties

LookupQueuesUsingJNDI: If the value of this property is set to true, the connector's SOAP/JMS listeners and SOAP/JMS protocol handler will look up queues using JNDI.

Default = false

JNDIProviderURL: This property gives the URL of the JNDI service provider, which corresponds to jndiProviderURL attribute of the jms:address element of the SOAP/JMS binding. The jms:address element is specified in the wsdl:port section. This is used as the default JNDI provider and must be a valid JNDI URL. For further information, see JNDI specifications.

Default = none

InitialContextFactory: This property gives the fully qualified class name of the factory class (for example, com.ibm.NamingFactory)that creates an initial context. Note that this corresponds to the initialContextFactory attribute of the jms:address element of the SOAP/JMS binding. The jms:address element is specified in the wsdl:port section.

Default = none

JNDIConnectionFactoryName: This property gives the name of the connection factory to look up using JNDI context. Note that this corresponds to the jndiConnectionFactoryName attribute of the jms:address element of the SOAP/JMS binding. The jms:address element is specified in the wsdl:port section.

Default = none

CTX_ObjectFactories: Properties specifying additional information about security and object lookup in the JNDI context. Table 42 summarizes these properties. The + character indicates the entry's position in the property hierarchy.

Table 42. Java Naming and Directory Interface (JNDI) provider properties

Property Name	Description
+CTX_StateFactories +CTX_URLPackagePrefixes +CTX_DNS_URL +CTX_Authoritative +CTX_Batchsize +CTX_Referral +CTX_SecurityProtocol +CTX_SecutiryAuthentication +CTX_SecurityPrincipal +CTX_SecurityCredentials +CTX_Language	Properties specifying additional information about security and object lookup in the JNDI context. See J2EE documentation for more information. These properties reflect those used by the Adapter for JMS.

Creating multiple protocol listeners

You can create multiple instances of protocol listeners. Protocol listeners are configured as child properties of the ProtocolListenerFramework -> ProtocolListeners connector property. Each child (of ProtocolListenerFramework -> ProtocolListeners) identifies a distinct protocol listener for the connector. Accordingly, you can create additional protocol listeners by configuring new child properties under the ProtocolListeners property. Make sure that you specify all of the child properties of the newly created listener property. Each listener must be uniquely named. However, you do not change the listener Protocol property (`soap/http`, `soap/https`, or `soap/jms`), which remains the same for multiple instances of a listener.

Note: The Protocol property is very important because it serves as a switch. If you do not want to use a listener or a handler, leave this property empty.

If you are creating multiple instances of a SOAP/HTTP or SOAP/HTTPS listener, be sure to specify different Port and Host properties for each instance. If you are specifying multiple SOAP/JMS listeners, be sure to use a different set of queues for each instance.

You cannot create multiple instances of a handler. There can be only one handler for each protocol.

Connector at startup

When you start the connector, the `init()` method reads the configuration properties that were set using System Manager's Connector Configurator. For proper functioning, be sure not to disable connector polling (connector polling is enabled by default). The sections below describe what occurs.

Proxy setup

If you specify the ProxyServer connector-specific property, the connector sets up the proxy system properties. A proxy server is used with the SOAP/HTTP-HTTPS protocol handler for request processing only. The connector also traces each of the system properties it sets up. For more on the ProxyServer property, see "Connector-specific configuration properties" on page 85.

JNDI initialization

The connector-specific property JNDI specifies the JNDI to be used by the connector. The connector uses JNDI to lookup the JMS Connection Factory object. If JNDI " LookupQueuesUsingJNDI is set to true, the connector looks up JMS queue objects using JNDI.

If you do not want to use SOAP/JMS (the SOAP/JMS protocol listener and SOAP/JMS protocol handler), you need not specify JNDI properties. If you specify JNDI properties and the connector cannot initialize JNDI, the connector terminates. Note that the connector will not initialize JNDI unless all of the following connector-specific JNDI properties are specified:

- JNDIProviderURL
- InitialContextFactory
- JNDIConnectionFactoryName

Note: JNDI implementation is not provided with the connector

Protocol listener framework initialization

During startup the connector instantiates the protocol listener framework and initializes it. This framework reads the connector-specific property ProtocolListenerFramework, The connector then reads the value of WorkerThreads and RequestPoolSize connector properties. If the ProtocolListenerFramework property is unspecified or missing, the connector cannot receive requests from web service clients and logs a warning.

The connector next reads the ProtocolListenerFramework -> ProtocolListeners property. All the first-level properties of the ProtocolListeners property represent protocol listeners. The protocol listener framework attempts to load and initialize each of the listeners and traces them. If persistent event capable, the listener attempts an event recovery.

Protocol handler framework initialization

The connector reads the connector-specific property ProtocolHandlerFramework and instantiates and initializes the protocol handler framework. If this property is missing or not set properly, the connector cannot perform request processing and logs a warning. Next the connector reads all the ProtocolHandlerFramework " ProtocolHandlers properties, which correspond to protocol handlers, and attempts to load, initialize, and trace them. Note that the protocol handlers are loaded

during connector initialization and are not instantiated when a collaboration makes a service request. The protocol handlers are multi-thread safe.

Logging

The connector logs a warning when:

- the ProtocolListenerFramework property is not specified. The connector warns that it cannot perform event notification. (Collaborations exposed as web services cannot be invoked by the connector.)
- the ProtocolHandlerFramework property is not specified. The connector warns that it cannot perform (collaboration) request processing.

Tracing

Tracing is an optional debugging feature you can turn on to closely follow connector behavior. Trace messages, by default, are written to STDOUT. See the connector configuration properties for more on configuring trace messages. For more information on tracing, including how to enable and set it, see the *Connector Development Guide for Java*.

Connector trace levels are as follows:

Level 0
: This level is used for trace messages that identify the connector version.

Level 1
: Trace each time the pollForEvents method is called. Trace the TLO name created by listeners for delivery to ICS. Trace the Request business object name and the corresponding attribute name in the TLO.

Level 2
: Use this level for trace messages that log each time a business object is posted to InterChange Server, either from `gotApplEvent()` or `executeCollaboration()`. Also, trace which protocol handler is processing the request.

Level 3
: Trace the ASI of the business object being processed. Trace attributes of the business object being processed. Trace the TLO of the SOAP Request business object during event notification. Trace the business object returned by the data handler.

Level 4
: Trace the transport headers associated with:
  - a SOAP request message retrieved by the protocol listener from the transport
  - a response message sent to the client by the protocol listener.

  Trace the spawning of threads, all ASI that is processed, and all entries and exits of important functions.

Level 5
: Trace the following:
  - the entries and exits for each important method
  - all of the configuration-specific properties
  - the loading of each of the protocol listeners
  - the request message retrieved by the protocol listener from the transport
  - the response message sent on the transport to the client by the protocol listener
  - the loading of each protocol handler

- the messages returned by the SOAP data handler
- business object dumps of the TLO sent to the collaboration
- dumps of the business objects returned by the data handler.

Chapter 5. SOAP data handler

The SOAP data handler is a data-conversion module whose primary roles are to convert business objects into SOAP messages and SOAP messages into business objects. The SOAP data handler performs the following functions:

- Request Processing
  - SOAP request business object to SOAP request message
  - SOAP response message to SOAP response business object
  - SOAP fault message to SOAP fault business object
- Event Processing
  - SOAP request message to SOAP request business object
  - SOAP response business object to SOAP response message
  - SOAP fault business object to SOAP fault message

This chapter describes how to configure the SOAP data handler, how the SOAP data handler processes messages and objects, and how to customize the data handler.

Configuring the SOAP data handler

The SOAP data handler is a pivotal component in the connector for web services. The connector calls the SOAP data handler to transform business objects into web services-compliant SOAP messages.

When collaborations are exposed as web services, the connector also calls the SOAP data handler. The data handler then transforms SOAP messages sent from a remote trading partner (or internal client) into business objects. The connector passes the business objects to collaborations that have been configured for web services.

The information in data handler meta-objects plays a crucial role in these transformations. You configure this information after you install the product files, but before startup. Unless you are adding a custom name handler, you can use the default SOAP data handler configuration to save time. You must, however, configure specific meta-object information for each data handler transformation. Data handler meta-objects are discussed in the sections below.

Meta-object requirements

Meta-objects are business objects that contain configuration information. The connector uses meta-objects at runtime to configure the data handler and create instances of it. The SOAP data handler also uses meta-objects to locate the body of a SOAP message, to determine the business object and verb that the body corresponds to, to encode a business object in a SOAP message, and to perform a number of other tasks discussed in this chapter. This section describes requirements for these meta-objects.

Meta-object hierarchy and terminology

Figure 28 shows the meta-object structure for the adapter for web services product. The meta-objects are named in bold in the illustration and discussed below.

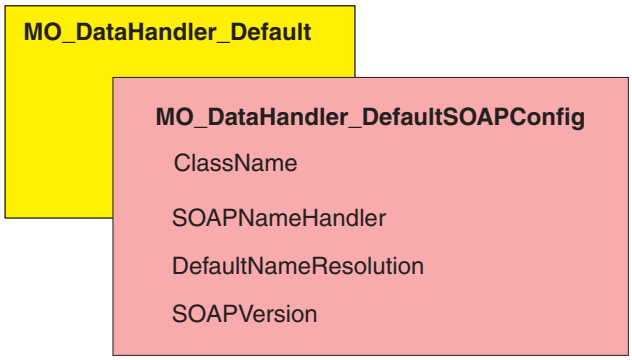


Figure 40. Meta-object structure

The following terminology is used throughout this document when discussing meta-objects:

- `MO_DataHandler_Default` Data handler meta-object used by the connector agent to determine which data handler to instantiate. This is specified in the DataHandlerMetaObjectName property of the connector.
- `MO_DataHandler_DefaultSOAPConfig` Child data handler meta-object specifically for the SOAP data handler.
- SOAP Configuration Meta-Object (SOAP Config MO) A meta-object specified as child of each SOAP business object and that contains the configuration information for a single transformation from business object to SOAP message or vice-versa.

MO_DataHandler_Default

The MO_DataHandler_Default is the top-level meta-object for all data handlers that are called from connectors. The MIME type contained in these meta-objects determines which data handler to use. The connector agent uses this meta-object to create instances of the SOAP data handler. Accordingly, the MO_DataHandler_Default object must include an attribute named `xml_soap` that is of type `MO_DataHandler_DefaultSOAPConfig`.

You can configure the MO_DataHandler_Default object after installing it. You must add `xml_soap` of type `MO_DataHandler_DefaultSOAPConfig`.

MO_DataHandler_DefaultSOAPConfig

The connector agent uses this meta-object to create and configure the SOAP data handler at runtime. The MO_DataHandler_DefaultSOAPConfig has two attributes of type `string` that designate:

- The class name for the SOAP data handler
- The SOAP name handler
- A default name resolution when the custom name handler fails
- The SOAP version (1.1 or 1.2)

These attributes are shown in Table 43.

Unless you wish to implement a custom name handler, which is discussed later in this chapter, you can use the MO_DataHandler_DefaultSOAPConfig as delivered

and installed. No configuration is needed.

Table 43. Meta-object attributes for MO_DataHandler_DefaultSOAPConfig

Name	Type	Default value	Description
ClassName	String	`com.ibm.adapters .dataHandlers.xml. soap`	Standard attribute used by the data handler base class to find the class name based on a MIME type passed into the createHandler method.
SOAPName Handler	String		Name of the SOAP name handler to use.
DefaultName Resolution	String	`false`	Determines whether default name resolution is used if the custom name handler fails.
SOAPVersion	String	1.1	Determines the SOAP standard (1.1 or 1.2) that the data handler uses to read and write SOAP messages.

SOAP configuration meta-object: child of every SOAP business object

A SOAP Config MO defines the data formatting behavior for one data handler transformation — either a SOAP-message-to-business-object or business-object-to-SOAP-message transformation. A SOAP Config MO is a child of a SOAP business object. These child SOAP Config MOs are critical for default business object resolution. When using default business object resolution, all child SOAP Config MOs, whether for a request, response, or fault object, must have unique entries for default values of BodyName and BodyNS. Table 44 shows these and other attributes of a SOAP Config MO.

Table 44. Attributes for SOAP Config MOs

Name	Required	Description
BodyNS	Yes	Namespace to be used for SOAP body.
BodyName	Yes	Name of the body of the SOAP message. For SOAP fault, set the default value to `soap:fault`.
BOVerb	Yes	Verb of the business object that contains the SOAP Config MO.
TypeInfo	No	True or false attribute that dictates whether type information (xsi:type) is written to and read from a SOAP element. Default = `false`
TypeCheck	No	This property is read only if TypeInfo is set to `true`. Possible values are `none` and `strict`. If `none`, type validation is skipped when reading SOAP messages into this business object. If `strict`, the data handler will strictly validate all SOAP type names and namespaces against the business object's application-specific information. Default = `none`
Style	No	This property dictates the SOAP message style and has implications for other attributes such as BodyName and BodyNS. The possible values for this attribute are `rpc` and `document`. Default = `rpc`
Use	No	This property dictates the SOAP message's use and affects how the SOAP body is constructed from a business object. The possible values are `literal` and `encoded`. The default is `literal`.

Figure 41 shows the relationship between a SOAP business object and a SOAP Config MO.

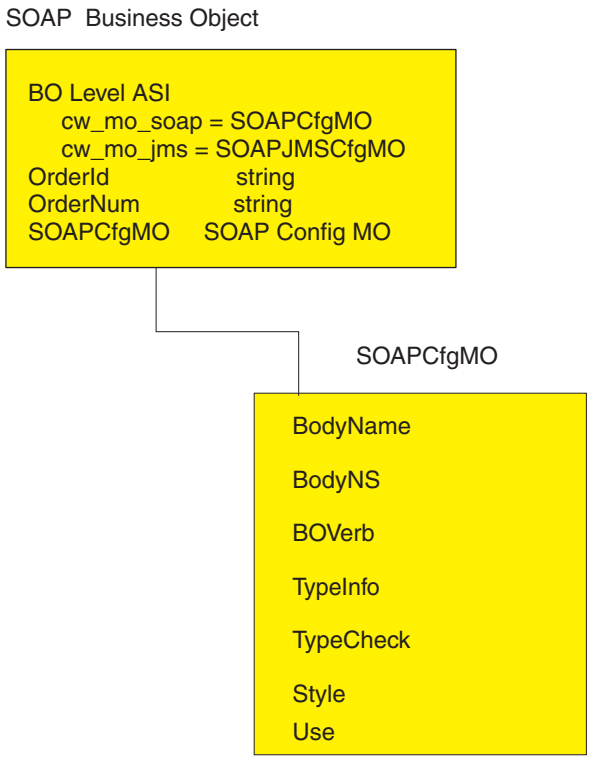


Figure 41. SOAP configuration meta-object

Figure 41 shows a SOAP response business object and its child business object. The child business object, SOAPCfgMO, is a SOAP Config MO that specifies the behavior for the SOAP data handler for a transformation from a business object response to a SOAP response message. The attribute indicating the child SOAP Config MO must use the name-value pair beginning `cw_mo_soap`.

By convention, when reading business object level application-specific information beginning with `cw_mo_`, the data handler recognizes that the child object specified in the name-value pair contains transformation meta-object information and therefore does not include this child as content in the body of the message it is transforming. In the example, the child objects indicated by the name-value pairs `cw_mo_jms` and `cw_mo_soap` are recognized as meta-objects and not written into the SOAP response message. In addition, the SOAP data handler ignores all business object level application-specific information beginning with `cw_mo_` except for `cw_mo_soap`. Accordingly, the SOAP data handler ignores the application-specific information such as `cw_mo_tpi`. But the SOAP data handler reads and uses the SOAP Config MO specified in `cw_mo_soap` to execute the SOAP response transformation from business object to SOAP message.

All SOAP business objects must have child SOAP Config MOs and these must be specified as application-specific information at the business object level. Much of this is automated: when you use the WSDL ODA to generate business objects for SOAP messages, the SOAP Config MOs are automatically generated for you.

Style and Use impact on SOAP messages

The SOAP Config MO optional properties, Style and Use, affect the way that SOAP messages are created. The possible values for Style are `rpc` and `document`, and for Use are `literal` and `encoded`. The sections below discuss how the Style and Use combinations impact SOAP message creation.

rpc/literal: When the Style property is set to `rpc` and the Use property to `literal`, the Body Name and Body Namespace for a SOAP Message are read from the SOAP ConfigMO's BodyName and BodyNS properties, respectively.

The following is an example of an `rpc/literal` style message where the Body Name and Body Namespace have been resolved to getOrderStatus and OrderStatusNS respectively:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV=îhttp://schemas.xmlsoap.org/soap/envelope/ì
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:getOrderStatus xmlns:ns1="http://www.ibm.com/">
         <Part1>
            <ns2:Elem1 xmlns:ns2="http://www.ibm.com/elem1">
               <Child1>1</Child1>
               <Child2>2</Child2>
            </ns2:Elem1>
            <ns3:Elem1 xmlns:ns3="http://www.ibm.com/elem1">
               <Child1>3</Child1>
               <Child2>4</Child2>
            </ns2:Elem1>
            <Elem2>10</Elem2>
         </Part1>
      </ns1:getOrderStatus>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 42 shows the corresponding business object for this `rpc/literal` message.

Name	Type	Key	Card	Default	App Spec Info
⊟ Part1	SOAP_Part1Type	☑	1		
⊟ Elem1	SOAP_MaxType	☑	N		maxoccurs=5;elem_ns=http://www.ibm.com/elem1
Child1	String	☑			
Child2	String	☐			
ObjectEventId	String				
Elem2	String	☐			
ObjectEventId	String				
⊟ SOAPConfigMO	SOAP_Req_Cfg_MO	☐	1		
BodyName	String	☑		getOrderStatus	
BodyNS	String	☐		http://www.ibm.com	
BOVerb	String	☐		Retrieve	
TypeInfo	String	☐		false	
TypeCheck	String	☐		none	
Style	String	☐		rpc	
Use	String	☐		literal	

Figure 42. rpc/literal SOAP Config MO

Note: You must configure these properties and business object attributes appropriately so that a corresponding SOAP message is created.

rpc/encoded: When the Style property is set to rpc and Use is set to encoded, the Body Name and Body Namespace for a SOAP Message are read from the Child ConfigMO's BodyName and BodyNS properties respectively. Also, the SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" attribute is added to the Body tag.

The following is an example of an rpc/encoded message where the Body Name and Body Namespace have been resolved to getOrderStatus and OrderStatusNS respectively.

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body SOAP-ENV:encodingStyle=
    "http://schemas.xmlsoap.org/soap/encoding/">
      <ns1:getOrderStatus xmlns:ns1="http://www.ibm.com/">
         <Part1 xsi:type="ns1:SOAP_Part1Type">
            <ns2:Elem1 SOAP-ENC:arrayType="ns2:SOAP_MaxType[2]"
             xsi:type="SOAP-ENC:Array" xmlns:ns2="http://www.ibm.com/elem1">
               <item>
                  <Child1 xsi:type="xsd:string">1</Child1>
            <Child2 xsi:type="xsd:string">2</Child2>
               </item>
               <item>
                  <Child1 xsi:type="xsd:string">3</Child1>
            <Child2 xsi:type="xsd:string">4</Child2>
               </item>
            </ns2:Elem1>
            <Elem2 xsi:type="xsd:string">10</Elem2>
         </Part1>
      </ns1:getOrderStatus>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 43 shows the corresponding business object for this rpc/encoded message.

Name	Type	Key	Card	Default	App Spec Info
⊟ Part1	SOAP_Part1Type	☑	1		
⊟ Elem1	SOAP_MaxType	☑	N		elem_ns=http://www.ibm.com/elem1
Child1	String	☑			
Child2	String	☐			
ObjectEventId	String				
Elem2	String	☐			
ObjectEventId	String				
⊟ SOAPConfigMO	SOAP_Req_Cfg_MO	☐	1		
BodyName	String	☑		getOrderStatus	
BodyNS	String	☐		http://www.ibm.com	
BOVerb	String	☐		Retrieve	
TypeInfo	String	☐		true	
TypeCheck	String	☐		none	
Style	String	☐		rpc	
Use	String	☐		encoded	

Figure 43. rpc/encoded SOAP Config MO

document/literal: When the Style property is set to document and the Use property is set to literal, an all encompassing Body Name tag will not exist. This is an example of a document style SOAP message based on the above BO:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:Elem1 xmlns:ns1="http://www.ibm.com/elem1">
         <Child1>1</Child1>
         <Child2>2</Child2>
      </ns1:Elem1>
      <ns2:Elem1 xmlns:ns2="http://www.ibm.com/elem1">
         <Child1>3</Child1>
         <Child2>4</Child2>
      </ns2:Elem1>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 44 shows the corresponding business object for this `document/literal` message.

Name	Type	Key	Card	Default	App Spec Info
⊟ Elem1	SOAP_Elem1	☑	1		maxoccurs=3;elem_ns=http://www.ibm.com/elem1
Child1	String	☑			
Child2	String	☐			
ObjectEventId	String				
⊟ SOAPConfigMO	SOAP_Req_Cfg_MO	☐	1		
BodyName	String	☑		getOrderStatus	
BodyNS	String	☐		http://www.ibm.com	
BOVerb	String	☐		Retrieve	
TypeInfo	String	☐		false	
TypeCheck	String	☐		none	
Style	String	☐		document	
Use	String	☐		literal	

Figure 44. document/literal SOAP Config MO

Note that the encodingStyle attribute in the XML code fragment has not been set.

document/encoded: This Style/Use combination is not supported. The data handler fails if it encounters a SOAP ConfigMO with Style set to `document` and Use set to `encoded`.

SOAP data handler processing

The SOAP data handler performs transformations between SOAP messages and business objects in the following ways:

- **SOAP message to business object processing**
  - Request-message-to-SOAP-request-business-object data handling occurs at that stage in event processing when web service clients make calls to collaborations exposed as web services
  - Response-message-to-SOAP-response-business-object data handling occurs during request processing when a web service returns a SOAP response message to a collaboration that had invoked it. Alternatively, fault-message-to-SOAP-business-object data handling may occur at this phase.

  For a detailed description of this processing, see "SOAP-body-message-to-business-object processing" on page 114 later in this section.
- **Business object to SOAP message processing**

– Business-object-to-SOAP-response-message data handling occurs during event processing when a response business object is returned by the collaboration that is exposed as a web service. Alternatively, fault business object-to-SOAP-fault-message data handling may occur at this phase.
– Business-object-to-SOAP-request-message data handling occurs at that phase of request processing when a collaboration makes a service call to the connector to convert a business object to a SOAP request message.

For a detailed description of this processing, see "Business-object-to-SOAP-message-body processing" on page 116 later in this section.

SOAP-body-message-to-business-object processing

This section provides a step-by-step description of the SOAP-body-message-to-business-object transformation.

1. The SOAP data handler receives a SOAP message.
2. Using Apache SOAP APIs, the data handler parses the SOAP message.
3. The data handler extracts the components of the SOAP message: envelope, header, and body.
4. **Header processing** For more, see "SOAP-header-message-to-business-object processing" on page 115.
5. **Body processing** The data handler reads the first element of the SOAP body to determine if it carries a fault or data. If the body content is not a fault, the data handler does the following:
   a. Performs business object resolution to determine which business object will be used in the transformation. If you have configured a custom name handler, the default business object resolution discussed below may not apply. For more on specifying a pluggable name handler, see "Specifying a pluggable name handler" on page 137.
   b. The data handler also resolves the SOAP Config MO (a child of the SOAP business object that the data handler is creating) that will be used for the transformation. If an instance of the SOAP Config MO does not exist, the data handler creates an instance and reads its default values. From the ConfigMO attribute values, the data handler reads the business object verb. The data handler instantiates the SOAP business object and sets the verb accordingly. This is the business object into which the data handler will attempt to write the SOAP message.
   c. The data handler continues parsing the SOAP message one element at a time. For rpc, the data handler expects the first element to be the parent.
   d. The data handler expects that the attributes of the business object (or its application-specific information: for further information, see "ASI in business-object-to-SOAP-message transformations" on page 121) should have the same name as the child elements. If the attribute is not found in the business object, the data handler throws an exception. Child elements may be of simple type or they may be of complex type. Complex elements are those which have child elements.
   e. **Simple element** If a child element is a simple element, by default, the data handler expects a business object attribute with the same name (or ASI) as that of a simple element. The data handler reads the value of the simple element and sets it in the business object.
   f. **Complex element** If a child element is of complex type, the data handler expects the business object to have an attribute with the same name (or ASI) and of type child business object. This attribute may be of single cardinality or of multiple-cardinality depending on if there will be a complex SOAP

element or SOAP array. Next the data handler instantiates the child business object (by default, the type of the attribute gives the name of the child business object) and reads all the child elements of this complex element, setting their values in the child business object. The data handler sets this child business object into the parent business object attribute after verifying the cardinality of this attribute. If the attribute is cardinality n, the data handler appends this business object to the container. The complex element can have either simple or complex child elements. These are also handled in the same way: if it is simple element, the data handler sets the value in the child BO; if it is a complex element, the data handler instantiates a child business object.

6. **Fault processing** The data handler reads the name of the first element of the SOAP body to determine if it is a fault. If the name of the first element is `Fault`, the data handler concludes that this is a fault message. Fault business object resolution occurs to determine into which business object this fault message should be transformed. The data handler then follows the same processing as that for body processing. The data handler expects that the business object specified in the child business object should have the following attributes:
   a. faultcode: Required. String attribute
   b. faultstring: Required. String attribute
   c. faultactor: Not required String attribute
   d. detail: Not required. Child BO
7. If fault processing fails for any reason, the exception thrown will contain the text from the faultcode, faultstring and faultactor elements in the SOAP fault message

Note: According to SOAP specifications for fault messages, faultcode, faultstring, and faultactor are simple elements whereas detail is a complex element (an element with child elements). In addition, faultcode, faultstring, faultactor, and detail belong to the SOAP envelope namespace, whereas detail child elements may belong to user-defined namespaces.

SOAP-header-message-to-business-object processing

This section describes how the data handler converts the header of a SOAP message into a business object.

1. The SOAP data handler processes the body of a SOAP message. Body processing creates a SOAP business object.
2. If the SOAP message has a SOAP header element, the SOAP data handler expects a SOAP header attribute in the business object obtained from body processing. The `SOAPHeader` attribute is the child attribute of a business object and has `soap_location=SOAPHeader` as its application-specific information. If there is no such attribute, the SOAP data handler throws an error.
   The `SOAPHeader` attribute must be of type SOAP Header Container business object. The SOAP data handler creates an instance of this attribute in the SOAP business object obtained in step 1.
3. For each immediate child of the SOAP-Env:Header element:
   a. The data handler expects a child attribute in the SOAP Header Container Business Object. The name of this attribute must be the same as that of the header element and conform to the SOAP Header Child business object. If the data handler cannot find such an attribute, it throws an error. Additionally, the namespace of this element should be the same as specified

in the elem_ns application-specific information of this attribute. If it is not the same, the data handler throws an error.

b. The data handler creates an instance of the SOAP Header Child business object and places it in the instance of SOAP Header Container business object created in step 2.

c. If this header element has an actor attribute, the data handler expects an actor attribute to exist in the child business object created above. If it cannot find an actor attribute, the data handler throws an error.

   Note: If you want to add an actor attribute, see "Specifying SOAP attributes" on page 124.

d. If this header element has a mustUnderstand attribute, the data handler expects a mustUnderstand attribute to exist in the child business object created above. If it cannot find a mustUnderstand attribute, the data handler throws an error.

   Note: If you want to add a mustUnderstand attribute, see "Specifying SOAP attributes" on page 124.

e. For each child element of this header element, the data handler expects an attribute in the child business object with the same name. These elements will be processed in same way as the child elements of SOAP-Env:Body element.

Business-object-to-SOAP-message-body processing

The following is a step-by-step description of the business-object-to SOAP-body-message transformation. For special cases involving application-specific-information, see "ASI in business-object-to-SOAP-message transformations" on page 121

1. The SOAP data handler looks for a SOAP ConfigMO that corresponds to the SOAP business object it is transforming.
2. The data handler composes the envelope and header of the SOAP message.
3. The data handler resolves the SOAP ConfigMO. If an instance of the SOAP ConfigMO does not exist, the data handler will create an instance and read from the default values. By default, the data handler reads the value of the BodyName attribute in the SOAP ConfigMO to determine whether it is processing a fault business object. If it is set to soap:fault the business object is considered a SOAP fault business object. If it is not a fault business object, the data handler performs the processing described under composing body below, else that described under composing fault.
4. **Composing body** The following steps detail the processing performed by the data handler to compose the body of the SOAP message from a business object:
   - The data handler obtains the BodyName and BodyNS from the SOAP ConfigMO attributes and then composes the first (parent) element of the body of the SOAP message. The name of first element is, by default, the value for the BodyName. In this document, it is also referred to as the body element. The namespace of the body element is, by default, the value determined for BodyNS. If the Style attribute of the SOAP ConfigMO is set to document, this step (creating the first body element) is skipped.
   - The data handler then reads the attributes of the business object and processes them by type. The processing for each type of attribute is described below.
     – **Simple attributes** If the attribute is of type simple, the data handler creates a child element from the body element, with the same name as the

attribute (unless otherwise specified by special application-specific information). The data handler sets the value of this element to the value of the attribute in the business object.

- **Cardinality 1 child business object attributes**

  If the attribute is a single cardinality child business object, the data handler creates a child element of the body element. This is referred to as a child business object element. The name of the child element created is the same as that of the attribute (unless otherwise specified by special ASI properties). The data handler then traverses the attributes of the child business object, creating the child elements for the attributes in the same way it processes the attributes of the incoming business object. However, the child elements are made children not of the body element but of the child business object element

- **Cardinality n child business object attributes** If an attribute is a cardinality n child business object, the data handler creates a SOAP array. Each attribute is handled the same way that a single cardinality child business object is handled.

5. **Composing fault** The following section walks through the process by which the data handler composes a fault message.
   - The data handler expects the following attributes in the business object:
     - faultcode: Required, String attribute
     - faultstring: Required, String attribute
     - faultactor: Not required. String attribute
     - detail: Not required. Child BO attribute.

     If any required attributes are missing, the data handler errors out.
   - The data handler creates an element for faultcode. It sets the value given by the faultcode attribute of the business object.
   - The data handler creates an element for faultstring. It sets the value given by the faultstring attribute of the business object.
   - The data handler creates the faultactor. It sets the value given by the faultactor attribute of the business object.
   - If the detail attribute is present in the business object, the attribute should be of child business object type. Otherwise the data handler errors out. It handles the attributes of each detail business object as highlighted in the section on **Composing body** above.
6. **CxIgnore processing** If the data handler finds out that the value of an attribute is set to CxIgnore, the data handler does not create an element for this attribute.
7. **CxBlank processing** If the data handler determines that the value of an attribute is set to CxBlank, the data handler creates an element for this attribute but does not set its value.

Business-object-to-SOAP-message-header processing

This section describes the processing of the SOAP header attribute only. All other attributes are processed as described in "Business-object-to-SOAP-message-body processing" on page 116.

1. From the business object, the SOAP data handler obtains the `SOAPHeader` attribute. This attribute has `soap_location=SOAPHeader` as its application-specific information. The SOAP data handler creates a `SOAP-Env:Header` element if and

only if the value of this attribute is not null. If a business object contains more than one SOAPHeader attribute, the first one is processed and the rest are treated as part of the body.

2. The SOAP data handler expects that the SOAPHeader attribute is a single cardinality child representing a SOAP Header Container business object. The data handler processes the child attributes of the SOAP Header Container business object that are of type SOAP Header Child business object.
3. For each attribute of the SOAP Header Container business object, the data handler does the following:
   a. Checks the cardinality: if this attribute is NOT a 1 or n cardinality child object, it is ignored.
   b. Checks the value: if the value of this attribute is NULL, it will be ignored.
   c. If the attribute is a 1 or n cardinality child object, the SOAP data handler creates a header element that is the immediate child of the SOAP-Env:Header element created in step 1. The name of this header element is same as that of the attribute. The namespace of this element is given by the elem_ns application-specific information of this attribute.
   d. If the attribute is a SOAP Header Child business object, all of the attributes of this business object are processed. This attribute may have an actor and a mustUnderstand attribute.

      Note: If you want to add a mustUnderstand or actor attribute, see "Specifying SOAP attributes" on page 124.
   e. If a SOAP Header Child business object has a non-null actor attribute, the data handler creates an actor attribute in the header element that was created in step c.
   f. If a SOAP Header Child business object has a non-null mustUnderstand attribute, the data handler will create a mustUnderstand attribute in the header element created in step c.
   g. All other non-null attributes of the SOAP Header Child business object become child elements of this header element. They are composed in the same manner as the child elements of the SOAP-Env:Body element.

Header fault processing

The SOAP specification states that errors pertaining to headers must be returned in headers. These headers are returned in the SOAP fault message. Just as message headers are specified in the SOAPHeader attribute of request and response business objects, fault headers are specified in the SOAPHeader attribute of fault business objects.

Each of the possible headers of request or response business objects may cause an error. Such errors are reported in the headers of the fault message.

WSDL documents have a SOAP binding header fault element that allows you to specify the fault header. For more information, see the SOAP and WSDL specifications listed in Chapter 1.

The application-specific information of headerfault allows you to specify header faults for each of your headers. You may specify headerfault application-specific information for each of the attributes of the SOAP Header Container business object. The list of attributes in the SOAP Header Container business object for the fault business object is as follows:

```
headerfault=attr1, attr2, attr3...
```

If the WSDL Configuration Wizard finds `headerfault` application-specific information in the SOAP Header Child business objects of request or response objects, the utility creates `headerfault` elements in the WSDL generated for these headers. Note that WSDL allows you to specify multiple header faults for each of your request (input) and response (output) headers. Therefore the value of this application-specific information is a comma-delimited list of attributes.

Using application-specific information functionality

You can specify object- and attribute-level application-specific information (ASI) to extend and enhance SOAP data handler functionality. Table 45 shows these attributes, which are discussed in the sections below. All of the entries in the table are attribute-level ASI unless otherwise noted.

Table 45. SOAP object ASI summary

ASI	Possible values	Description
soap_location	`SOAPHeader`	Specifies this business object attribute as the header attribute
headerfault	`String`	Identifies the BO attribute name of the corresponding SOAP header in the fault BO
elem_name	`String`	Specifies the name for the SOAP element corresponding to this BO attribute
elem_ns	`String`	Specifies the namespace for the SOAP element corresponding to this BO attribute
type_name	`String`	Specifies the type for the SOAP element corresponding to this BO attribute
type_ns	`String`	Specifies the type namespace for the element corresponding to this BO attribute
xsdtype	`true`	Specifies xsd as the namespace for the element corresponding to this BO attribute, overriding older xsd versions (such as 1999, 2000, etc.) with the latest version of xsd (for example, 2001).
attr_name	`String`	Specifies the name for the SOAP attribute corresponding to this BO attribute
attr_ns	`String`	Specifies the namespace for the SOAP attribute corresponding to this BO attribute

Table 45. SOAP object ASI summary (continued)

ASI	Possible values	Description
arrayof	`String`	Specifies the name of the `n` cardinality child business object attribute that must be used as a placeholder for the simple type array items
dh_mimetype	`String`	Specifies the mimeType of the data handler that will be used to transform this attribute of complex type
cw_mo_*	`String`	This business object level ASI specifies the name of a child config MO that is interpreted as meta-data, not content, by the SOAP data handler. Only cw_mo_soap specifies a child config MO that is processed as meta-data; all other cw_mo_* indicate a different component and are therefore excluded from SOAP data handler processing. All other cw_mo* is ignored.
cw_mo_soap	`String`	This business object level ASI specifies the name of the Child Config MO attribute that should be used when transforming this business object
cw_mo_jms	`String`	This business-object level ASI specifies the name of the JMS Protocol Config MO to use
cw_mo_http	`String`	This business-object level ASI specifies the name of the HTTP Protocol Config MO to use
wrapper	`true`	Specifies the attribute name of the wrapper object within this business object. Wrapper objects are used for certain schema indicators, and must not be serialized
maxoccurs	`Integer`	Specifies this business object attribute's maximum occurrence possibility. Depending on the value of maxoccurs, the business object may or may not have a wrapper.
minoccurs	`Integer`	Specifies this business object attribute's minimum occurrence possibility. Depending on the value of minoccurs, the object may or may not have a wrapper.

Table 45. SOAP object ASI summary (continued)

ASI	Possible values	Description
all	String	Specifies the child attribute that represents the all indicator in the schema.
choice	String	Specifies the child attribute that represents the choice indicator in the schema.

ASI in business-object-to-SOAP-message transformations

The SOAP data handler uses a business object's ASI to determine how to construct a SOAP message. Unless otherwise stated, all ASI discussed in the sections below refers to attribute level ASI and all string-based comparisons are performed without regard to case.

elem_name and elem_ns processing

The examples discussed in this section assume that the attribute name is OrderId and the SOAP element namespace prefix ns0.

1. When neither elem_name nor elem_ns are specified, the elem_name defaults to the attribute name, and the elem_ns defaults to the namespace of the element's parent.The ASI is not specified.

   ```
   <OrderId>1</OrderId>
   ```

2. When the elem_name is specified and the elem_ns is not specified, the elem_name will be set to the ASI elem_name value, and the elem_ns will be defaulted to the namespace of the SOAP Body. The ASI is as follows:

   ```
   elem_name=CustOrderId
   <CustOrderId>2</CustOrderId>
   ```

3. When elem_ns is specified and elem_name is not, elem_name defaults to the attribute name and elem_ns is set to the ASI elem_ns value. The xmlns attribute is explicitly written if and only if the element namespace is not found elsewhere in the scope of this element. If the element namespace is found, the already defined namespace prefix is used. Otherwise (if the element namespace is no found), a unique prefix for the elem_ns is generated. Consider the following example, which presumes that a prefix is already defined in scope (ns1 represents a prefix corresponding to a namespace already defined in the scope of this element). The ASI is as follows:

   ```
   elem_ns= http://www.w3.org/2001/XMLSchema
   <ns1:OrderId>3</ns1:OrderId>
   ```

   The following example presumes that prefix is not found (ns2 represents a unique prefix). The ASI is as follows:

   ```
   elem_ns=CustOrderIdNamespace
   <ns2:OrderId xmlns:ns2="CustOrderIdNamespace">3</ns2:OrderId>
   ```

4. When both elem_name and elem_ns are specified, elem_name and elem_ns are set to the ASI values. The same check that is performed in case 3 above regarding already defined namespaces applies. Just as in case 3, if the namespace is not already defined, a unique prefix for the elem_ns is generated. The ASI is as follows:

   ```
   elem_name=CustOrderId;elem_ns=CustOrderIdNamespace
   <ns2:CustOrderId xmlns:ns2="CustOrderIdNamespace">1</ns2:OrderId>
   ```

type_name and type_ns processing for simple attributes

For the examples in this section, the attribute name is `OrderId`, the SOAP element namespace prefix is `ns0`, and the attribute type is `String`.

Note: type_name and type_ns processing takes place only when the Config MO attribute TypeInfo is `true`.

1. When neither type_name nor type_ns are specified, type_name defaults to the simple type and the type_ns defaults to the xml schema-defined namespace (xsd). The ASI is not specified

   ```
   <OrderId xsi:type="xsd:string">1</OrderId>
   ```

2. When type_name is specified and type_ns is not, type_name is set to the ASI type_name value and type_ns defaults to the namespace of the element. The ASI is as follows:

   ```
   type_name=CustString
   <OrderId xsi:type="ns0:CustString">2</OrderId>
   ```

3. When type_ns is specified and type_name is not, the type_ns defaults to the simple type name and type_name is set to the ASI type_ns value. The prefix is handled in a way that is comparable to elem_ns creation. A unique prefix for the type namespace is generated unless the namespace already exists in the element scope. The ASI is as follows:

   ```
   type_ns=CustStringNamespace
   <OrderId xmlns:ns2="CustStringNamespace" xsi:type=
   "ns2:String">3</OrderId>
   ```

4. When both type_name and type_ns are specified, they are set to the assigned ASI values. A unique prefix for the type namespace is generated. The ASI is as follows:

   ```
   type_name=CustString;type_ns=CustStringNamespace
   <OrderId xmlns:ns2="CustStringNamespace" xsi:type=
   "ns2:CustString">1</OrderId>
   ```

type_name and type_ns processing for single cardinality attributes

For the examples in this section, the attribute name is `OrderStaus`, the SOAP element namespace prefix is `ns0`, and the attribute type is `OrderStatus`.

Note: type_name and type_ns processing takes place only when the Config MO attribute TypeInfo is `true`.

1. When neither type_name nor type_ns are specified, type_name defaults to the business object name and the type namespace defaults to the namespace of the element. The ASI is not specified:

   ```
   <OrderStatus  xsi:type="ns0:OrderStatus">1</OrderStatus>
   ```

2. When type_name is specified and type_ns is not, the type_name is set to the assigned ASI value and type_ns defaults to the namespace of the element. The ASI is as follows:

   ```
   type_name=CustOrderStatus
   <OrderStatus  xsi:type="ns0:CustOrderStatus">1</OrderStatus>
   ```

3. When type_ns is specified and type_name is not, type_name defaults to the business object name and type_ns is set to the assigned type_ns value. A unique prefix for the type namespace is generated. The ASI is as follows:

   ```
   type_ns=CustTypeNS
   <OrderStatus  xsi:type="ns2:SOAP_OrderStatusLine
   " xmlns:ns2="CustTypeNS">1</OrderStatus>
   ```

4. When both type_name and type_ns are specified, they are set to the assigned ASI values. A unique prefix for the type namespace is generated. The ASI is as follows:

```
type_name=CustOrderStatus;type_ns=CustTypeNS
<OrderStatus
xsi:type="ns2:CustOrderStatus" xmlns:ns2="CustTypeNS">1</OrderStatus>
```

type_name and type_ns processing for multiple cardinality attributes

For all the examples given in this section assume the attribute name to be MultiLines and the SOAP element namespace prefix to be ns0. Assume the attribute type to be OrderStatus.

Note: type_name and type_ns processing takes place only when the Config MO attribute TypeInfo is `true`.

1. When neither type_name nor type_ns are specified, type_name defaults to the business object name and type_ns defaults to the namespace of the element. The ASI is as follows:

```
<MultiLines SOAP-ENC:arrayType="ns0:OrderStatus[2]"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="SOAP-ENC:Array">
```

2. When type_name is specified and type_ns is not, type_name is set to the assigned ASI type_name value and type_ns defaults to the namespace of the element. The ASI is as follows:

```
type_name=CustOrderStatus
<MultiLines SOAP-ENC:arrayType="ns0:CustOrderStatus[2]"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="SOAP-ENC:Array">
```

3. When type_ns is specified and type_name is not, type_name defaults to the business object name, and the type_ns is set to the assigned ASI type_ns value. A unique prefix for the type namespace is generated. The ASI is as follows:

```
type_ns=CustTypeNS
<MultiLines SOAP-ENC:arrayType="ns2:OrderStatus[2]"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/
" xmlns:ns2="CustTypeNS" xsi:type="SOAP-ENC:Array">
```

4. When both type_name and type_ns are specified, they are set to the assigned ASI values. A unique prefix for the type namespace is generated. The ASI is as follows:

```
type_name=CustOrderStatus;type_ns=CustTypeNS
<MultiLines SOAP-ENC:arrayType="ns2:CustOrderStatus[2
]" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns2="CustTypeNS" xsi:type="SOAP-ENC:Array">
```

Note: The item element representing the parent for each Array element has the same type and namespace as the arrayType.

xsdtype for simple, single, and multiple cardinality types

For simple, single. and multiple cardinality types, set the xsdtype ASI attribute to `true` for the type name to adhere to the current XSD for the SOAP message. The xsdtype property is read only when both the type_name and type_ns properties are set. Given the type_name and type_ns, the SOAP data handler first attempts to map the pair to a Java type using the SOAP API. Then the data handler attempts to convert the Java type back to a SOAP element type using the current XSD for the SOAP Message. For example, if the current XSD is

```
http://www.w3.org/2001/XMLSchema
```

and the following ASI:

```
type_name=timeInstant;type_ns=http://www.w3.org/1999/XMLSchema;xsdtype=true
```

The SOAP message type name is written as:

```
<OrderDate xsi:type="xsd:dateTime">
```

because `dateTime` is the 2001 XSD equivalent of the `timeInstant` in the 1999 XSD.

xsdtype and simple type arrays

For multiple cardinality objects, you can create a simple type array such as the following:

```
<MultiLines SOAP-ENC:arrayType="xsd:string[4]"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="SOAP-ENC:Array">
```

To achieve this, set the type_name property to the desired simple type (for example, `string`) and set the type_ns property to the appropriate XSD specification. Then, set the xsdtype property to `true` so that the type is converted to the current XSD type. Finally, the arrayOf property should be set to the name of the attribute in the container which should hold the simple type value. This is an example of what the ASI would look like for a string array:

```
arrayof=size;type_name=string;type_ns=http://www.w3.org/2001/XMLSchema;xsdtype=true
```

ASI effects on fault processing

The faultcode, faultactor, faultstring, and detail elements adhere to the following rules:

1. Any elem_name, elem_ns, type_name and type_ns ASI in these attributes is ignored.
2. All children of the detail elements are written exactly as described in body processing.

ASI effects on header processing

You can use all ASI properties (see Table 45) at the header child object level and below.

Specifying SOAP attributes

attr_name processing for simple types

There is an XML schema case in which complexTypes with simpleContent extensions or restrictions have have both values and attributes. For example, consider the following SOAP tag:

```
<size system="us">10</size>
```

It is based on the following schema:

```
<complexType name="SizeType">
   <simpleContent>
       <extension base="int">
          <attribute name="system" type="string"/>
       </extension>
   </simpleContent>
</complexType>
<element name="size" type="ns:SizeType"/>
```

The business object corresponding to the complex type, with simple content extension or restriction, must contain one additional attribute besides other

attributes that correspond to the complex type attributes. The additional attribute must contain the simple content value (in the example above, 10— the value of element size). The business object attribute, having the business object corresponding to such a complex type as its type, will have `elem_value=simpleContentValue` as its attribute-level ASI.

Figure 45 shows the corresponding business object.

Name	Type	Key	Card	Application Specific Informatio
⊟ Request	SOAP_getQuote_N09218329332_Request	☐	1	ws_botype=request
⊟ size	SOAP_getQuote_C09218329332_SizeType	☐	1	elem_value=simpleContentValue; type_name=SizeType
simpleContentValue	String	☑		
system	String	☐		attr_name=system

Figure 45. attr_name business object for simple types

attr_name processing for single and multiple cardinality types

You can specify ASI that translates business object attributes into soap attributes instead of into soap elements. The data handler supports adding SOAP attributes to complex single and n-card types only. Consider the following sample:

```
<CustInfo City="4" State="5" Street="2" Zip="6">
   <Name xsi:type="xsd:string">1</Name>
   <Street2 xsi:type="xsd:string">3</Street2>
</CustInfo>
```

Given this business object definition structure (with the attribute level ASI specified to the right of each attribute in Figure 46), the data handler follows these processing steps:

Name	Type	App Spec Info
⊟ CustInfo	CustomerInfo	
Name	String	
Street1	String	attr_name=Street
Street2	String	
City	String	attr_name=City
State	String	attr_name=State
Zip	String	attr_name=Zip

Figure 46. attr_name business object

1. When traversing a complex attribute, the data handler first generates a corresponding tag for this complex business object attribute. In this example, CustInfo represents the complex business object attribute.
2. The data handler iterates through the children of the complex business object. Only simple type attributes are considered for attribute creation. If a simple type has an ASI property named `attr_name`, the data handler writes this simple type as an attribute to the SOAP element. In this example, the element (CustInfo) will have four attributes; Street, City, State and Zip.
3. The rest of the attributes of the business object are written using standard BODY processing. This means that all relevant ASI will also be evaluated for the business object attributes that do not have `attr_name` ASI.

The logic for processing multiple cardinality types is identical to that for processing single cardinality types. Specifically, each <item> tag corresponds to each business object instance in the multiple cardinality object, and will be processed using ASI. For example, given this multiple cardinality business object definition structure with corresponding ASI:

Name	Type	Card	App Spec Info
⊟ CustInfo	CustomerInfo	N	
Name	String		
Street1	String		attr_name=Street
Street2	String		
City	String		attr_name=City
State	String		attr_name=State
Zip	String		attr_name=Zip

Figure 47. attr_name multiple cardinality business object

If the event sent to the data handler had two instances of this multiple cardinality object, the SOAP message created may look like this:

```
<CustInfo>
   <item City="Armonk" Street="Main Street">
      <Name>IBM</Name>
      <Street2>None</Street2>
   </item>
   <item City="Burlingame" State="Ca" Street="577 Airport Blvd" Zip="94010">
      <Name>Burlingame Labs</Name>
      <Street2>Suite 600</Street2>
   </item>
</CustInfo>
```

Notice that the item tags are treated as the complex element type. Any attributes in the BO definition will become SOAP attributes of the corresponding item tag.

arrayof processing for simple type arrays

The arrayof ASI property should only be used in the case of SOAP encoded simple type arrays. For example, a serialization such as the following:

```
<CustomerNames SOAP-ENC:arrayType="xsd:string[4]" xmlns:SOAP-ENC=
"http://schemas.xmlsoap.org/soap/encoding/" xsi:type="SOAP-ENC:Array">
<item xsi:type="xsd:string">value1</item>
<item xsi:type="xsd:string">value2</item>
<item xsi:type="xsd:string">value3</item>
<item xsi:type="xsd:string">value4</item>
</CustomerNames>
```

would require a business object definition such as that shown inFigure 48 :

⊟ Request	SOAP_echoStringArray_N0562488530_Request	1	ws_botype=request
⊟ CustomerNames	SOAP_echoStringArray_N0562488530_N1185926546_ArrayOfstring	n	arrayof=Item;type_name=string;type_ns=http://www.w3.org/2001/XMLSchema
Item	String		type_name=string;type_ns=http://www.w3.org/2001/XMLSchema

Figure 48. arrayof business object

(The business object is shown from the Request level for clarity.)

Note: Although not shown, the SOAP Config MO's TypeInfo property must be set to `true` in this example to derive the above SOAP serialization from the business object structure.

Also, the `arrayof` property can be used to create array items with a name other than `item`. Using the example above, the <item> tags can be replaced with <name> tags if both the BO attribute name and the ″arrayof″ asi property value is name. This would be the serialization:

```
<CustomerNames SOAP-ENC:arrayType="xsd:string[4]"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="SOAP-ENC:Array">
<name xsi:type="xsd:string">value1</name>
<name xsi:type="xsd:string">value2</name>
<name xsi:type="xsd:string">value3</name>
<name xsi:type="xsd:string">value4</name>
</CustomerNames>
```

attr_name and attr_ns processing

You may need to provide a namespace that corresponds to the SOAP attribute created. You do this by specifying the attr_ns ASI property for a simple type. The data handler processes the attr_ns property if and only if attr_name exists in the same attribute's ASI. The following rules are followed with attr_name and attr_ns:

1. When neither attr_name nor attr_ns exist, the business object attribute is translated to a SOAP element.
2. When only attr_name is set, the SOAP attribute's namespace defaults to the element's namespace:

   ```
   <CustInfo Street="577 Airport"></CustomerInfo>
   ```

3. When only attr_ns is set, the property is ignored and the business object attribute is translated to a SOAP element.
4. When both attr_name and attr_ns exist, the SOAP attribute is created like the following:

   ```
   <CustInfo ns2:Street="577 Airport" xmlns:ns2=
   "AttrNS"></CustomerInfo>
   ```

dh_mimetype: calling a data handler

The SOAP data handler can call another data handler to write business objects into any format for which a data handler exists. You do this by adding encoded text to a SOAP message when transferring a SOAP child business object into a SOAP String.

An RNIF document is one of the formats in which a SOAP element's value may be encoded. To make use of this functionality, add an RNIF BO at any level of a SOAP child business object. To signal the SOAP data handler to call another data handler when transforming this RNIF business object to a string, add the dh_mimetype property to the attribute's ASI. The value of the dh_mimetype ASI property must be a legal mimeType specified in the MO_DataHandler_Default meta-object. The mimeType is used to determine which data handler is called to process the business object.

Figure 49 shows a SOAP child business object in which `CustomerInfo` is a complex child and `RNET_Pip3A2PriceAndAvailabilityQuery` is an RNIF business object:

Name	Type	App Spec Info
⊟ CustomerInfo	CustomerInfo	
Name	String	
CustID	String	
⊞ RNIFMsg	RNET_Pip3A2PriceAndAvailabilityQuery	elem_name=RNIFexample;dh_mimetype=application/x_rosettanet_agent;type_name=base64Binary;type_ns=http://www.w3.org/2001/XMLSchema;xsdtype=true

Figure 49. RNIF business object with dh_mimetype

The SOAP message created from this business object may look like this:

```
<CustomerInfo>
<Name>IBM Corporation</Name>
<CustID>95626</CustID>
<RNIFexample
xsi:type="xsd:base64Binary">1AWERYER238W98EYR923872837487189278 7ASRJK23423
JKAWERJ234AWERIJHI423488R4HASF1AWERYER238W98EYR923872837487189278 7ASRJK234
34JKAWERJ234AWERIJHI423488R4HASF1AWERYER238W98EYR923872837487189278 7ASRJK2
4234JKAWERJ234AWERIJHI423488R4HASF1AWERYER238W98EYR923872837487189278 7ASRJ
234234JKAWERJ234AWERIJHI423488R4HASFWR234
</RNIFexample>
</CustomerInfo>
```

Note that the RNIF example element contains an RNIF encoded string that has been base64 binary encoded as its element value. Also, note that elem_name, elem_ns, type_name, type_ns, and xsdtype ASI properties remain relevant for this business object attribute. In this example, the specified elem_name dictates the name of the SOAP element upon message creation.

Note: If the element value returned by the called data handler is encoded text, the type_name property must be set to `base64Binary`, the type_ns must correspond to an xsd namespace, and xsdtype must be set to `true`.

xsd:base64Binary: When you set the type_name and type_ns to resolve to xsd:base64Binary, the SOAP data handler encodes the value from the business object before setting the value for the corresponding element. Using the Apache API, the data handler queries the registry for a base64Binary serializer, serializes the string returned from the called data handler, and sets the element's value.

Schema complexType indicators

The following sections discuss the effects of schema complexType Indicators on business objects. The indicators include:

- maxOccurs
- minOccurs
- all
- sequence
- choice

maxOccurs and minOccurs indicators for simple types: The maxOccurs indicator specifies the maximum number of times an element can occur within a complex type. The minOccurs indicator specifies the minimum number of times an element should occur within a complexType.

Consider this Schema:

```
<xs:element name="Address" type="Address">
<xs:complexType name="Address">
     <xs:sequence>
```

```
        <xs:element name="AddressLine" type="xsd:string" maxOccurs="10"/>
        <xs:element name="SuiteNumber" type="xsd:string" minOccurs="3"
                              maxoccurs="unbounded"/>
        <xs:element name="City" type="xsd:string"/>
     </xs:sequence>
     </xs:complexType>
</xs:element>
```

The example above indicates that the `AddressLine` element can occur at most ten times in an `Address` element, while the `SuiteNumber` element must occur at least three times. The business object that corresponds to this schema must have an N cardinality wrapper object for each maxoccurs/minoccurs indicator that has the following ASI:

```
maxOccurs=N;wrapper=true
```

or

```
minOccurs=3;wrapper=true;
```

The `wrapper=true` ASI indicates that this object is a wrapper, and therefore not explicitly written to the SOAP message. Instead, there must be one child of simple type in this wrapper object. At runtime, for SOAP to business object transformations, the data handler reads the `N` child objects of the wrapper and creates a corresponding element for each one. When performing business-object-to-SOAP-message transformations, the data handler creates child objects in the `N` cardinality wrapper for every element it encounters.

The corresponding SOAP business object resembles that shown in Figure 50.

Pos	Name	Type	Key	Card	App Spec Info
1	⊟ Address	Address	☑	1	
1.1	⊟ AddressLine	AddressLine_wrap	☑	N	maxoccurs=10;wrapper=true
1.1.1	AddressLine	String	☑		
1.1.2	ObjectEventId	String			
1.2	⊟ SuiteNumber	SuiteNumber_wrap	☑	N	minoccurs=3;wrapper=true
1.2.1	SuiteNumber	String	☑		
1.2.2	ObjectEventId	String			
1.3	City	String	☐		
1.4	ObjectEventId	String			
2	ObjectEventId	String			

Figure 50. minOccurs and maxOccurs of simple type ASI in a SOAP business object

The SOAP message that corresponds to the business object shown in Figure 50 is as follows:

```
<Address xsi:type="ns0:Address">
    <AddressLine xsi:type="xsd:string">Line1</AddressLine>
    <AddressLine xsi:type="xsd:string">Line2</AddressLine>
    <SuiteNumber xsi:type="xsd:string">600</SuiteNumber>
    <SuiteNumber xsi:type="xsd:string">650</SuiteNumber>
    <SuiteNumber xsi:type="xsd:string">700</SuiteNumber>
     <City xsi:type="xsd:string">San Francisco</City>
</Address>
```

Note: The SOAP data handler processes maxOccurs and minOccurs indicators in the same way, without validating the maximum or minimum occurrences of

elements. The data handler simply provides a container structure to hold multiple instances of a particular element with the maxOccurs and minOccurs indicators. This applies to simple and complex types.

maxOccurs and minOccurs indicators for complex types: The <maxOccurs> indicator specifies the maximum number of times an element can occur within a complex type. The <minOccurs> indicator specifies the minimum number of times an element should occur within a complexType. Consider the maxOccurs indicator in the following schema:

```
<xs:element name="Address" type="Address">
<xs:complexType name="Address">
     <xs:sequence>
     <xs:element name="AddressInfo" type="AddressInfo" maxOccurs="3"/>
     <xs:element name="City" type="xsd:string"/>
  </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:complexType name="AddressInfo">
     <xs:sequence>
     <xs:element name="StreetLine" type="xsd:string"/>
     </xs:sequence>
  </xs:complexType>
</xs:element>
```

The example above indicates that the `AddressInfo` complex type element can occur at most three times in an `Address` element. The corresponding business object for this schema will not have a wrapper object, since the complexType `AddressInfo` itself can be of `N` cardinality. The following ASI will be placed at the `N` cardinality attribute: `maxoccurs=3`

Figure 51 shows the corresponding SOAP business object.

Pos	Name	Type	Key	Card	App Spec Info
1	⊟ Address	Address	☑	1	
1.1	⊟ AddressInfo	AddressInfo	☑	N	maxoccurs=3
1.1.1	StreetLine	String	☑		
1.1.2	ObjectEventId	String			
1.2	City	String	☐		
1.3	ObjectEventId	String			
2	ObjectEventId	String			

Figure 51. minOccurs and maxOccurs of complex type ASI in a SOAP business object

The SOAP message that corresponds to the business object shown in Figure 51 is as follows:

```
<Address xsi:type="ns0:Address">
     <AddressInfo xsi:type="ns0:AddressInfo">
        <StreetLine xsi:type="xsd:string">100 Market St.</ StreetLine>
        <StreetLine xsi:type="xsd:string">Apt 15</ StreetLine>
     </AddressInfo>
  <City xsi:type="xsd:string">San Francisco</City>
</Address>
```

all indicator: The `all` indicator specifies by default that the child elements for this complexType can appear in any order and that each child element must occur zero or one times. Consider the following Schema:

```
<complexType name="Item">
     <all>
        <element name="quantity" type="xsd:int"/>
        <element name="product" type="xsd:string"/>
     </all>
</complexType>
```

The example above indicates that the elements `quantity` and `product`, can occur in any order in the SOAP message. The `quantity` element may occur first and the `product` element second, or vice versa.

Figure 52 shows the business object that corresponds to this schema fragment.

Pos	Name	Type	Card	App Spec Info
1	⊟ Item	Item	1	all=Item_wrapper
1.1	⊟ Item_wrapper	Item_wrapper	N	wrapper=true
1.1.1	quantity	String		
1.1.2	product	String		
1.1.3	ObjectEventId	String		
1.2	ObjectEventId	String		
2	ObjectEventId	String		

Figure 52. all indicator ASI in a SOAP business object

The corresponding SOAP message fragment is as follows:

```
<Item xsi:type="ns0:Item">
    <quantity xsi:type="xsd:string">12</quantity>
    <product xsi:type="xsd:string">2</product>
</Item>
```

Handling array content with 'all' content model: The SOAP data handler processes complex-type array content with the 'all' content model as described in this section. In the example, `ArrayOfSOAPStruct` contains `SOAPStruct`, which has the 'all' content model.

```
<complexType name="SOAPStruct">
       <all>
       <element name="varString" type="string" />
       <element name="varInt" type="int" />
       <element name="varFloat" type="float" />
       </all>
</complexType>
<complexType   name ="ArrayOfSOAPStruct'">
  <complexContent>
   <restriction base='SOAP-ENC:Array'>
    <attribute ref='SOAP-ENC:arrayType'
     wsdl:arrayType='typens:SOAPStruct[]'/>
   </restriction>
  </complexContent>
 </complexType>
```

The SOAP data handler must generate the following SOAP data on serialization:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
        xmlns:SOAP-ENV = "http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
 <ns0:echoStructArray xmlns:ns0="http://soapinterop.org/">
```

```
<inputStructArray SOAP-ENC:arrayType="ns1:SOAPStruct[2]"
         xmlns:ns1="http://soapinterop.org/xsd" xsi:type="SOAP-ENC:Array">
    <item>
                <ns1:varFloat xsi:type="xsd:string">1.1</ns1:varFloat>
                <ns1:varInt xsi:type="xsd:string">1</ns1:varInt>
         <ns1:varString xsi:type="xsd:string">hi</ns1:varString>
    <item>
    <item>
         <ns1:varString xsi:type="xsd:string">hello</ns1:varString>
         <ns1:varInt xsi:type="xsd:string">1</ns1:varInt>
         <ns1:varFloat xsi:type="xsd:string">1.1</ns1:varFloat>
       </item>
   </inputStructArray>
  </ns0:echoStructArray>
</SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

In this example, `echoStructArray` is the name of the operation, and `inputStructArray` is the parameter name with type `ArrayOfSOAPStruct`.

sequence indicator: The `sequence` indicator specifies that child elements must appear in the order specified in the complexType.

```
<complexType name="Item">
     <sequence>
        <element name="quantity" type="int"/>
        <element name="product" type="string"/>
     </sequence>
</complexType>
```

The SOAP data handler does not require special ASI or wrapper objects for this indicator. By default, the data handler reads and writes SOAP elements in the order specified in the business object.

choice indicator: The `choice` indicator specifies that one and only one of the elements in a complexType can appear in the SOAP message. Consider the following schema:

```
<complexType name="Item">
    <choice>
        <element name="quantity" type="int"/>
        <element name="product" type="string"/>
    </choice>
</complexType>
```

The SOAP data handler does not require special ASI or wrapper objects for this indicator. When converting a business object to a SOAP message, the data handler defers to your choice of which elements should appear in the SOAP message. When converting a SOAP message to a business object, the data handler reads the existing element and populates the attribute to which it corresponds.

maxOccurs indicator on sequence, choice, group and all: Model Groups (sequence, choice, group, and all) have minOccurs and maxOccurs attributes. The default value for minOccurs and maxOccurs is `one`. For the all group, the maxOccurs can take a value of `one` only. The WSDL ODA and SOAP data handler support all possible values for maxOccurs on sequence, choice and group.

ASI in SOAP-to-business object transformations

The SOAP data handler uses a business object's ASI to read and validate an incoming SOAP message. The following rules apply to ASI validation by the SOAP data handler:

- Header and body processing are the same.
- The SOAP ConfigMO property, TypeCheck, must be set to `strict` and TypeInfo set to `true` for the data handler to perform the validation described in the sections below.
- type_name and type_ns validation are performed concurrently since type validation is generally dependent on both properties.

Note: Unless otherwise stated, all ASI discussed in the following sections is attribute-level ASI

elem_name validation

The following rules apply to validation for simple, cardinality `1` and cardinality `n` attributes:

1. When encountering an element while parsing a SOAP message, the data handler first searches all of the ASI at the business object level, attempting to match the element's name against the elem_name value.
2. If a match is not found, the data handler attempts to match the element's name against each of the attribute names at that business object level.
3. If neither search succeeds, the data handler fails.

elem_ns validation

The following cases apply to validation for simple, cardinality `1` and cardinality `n` attributes:

1. When neither elem_ns ASI nor xmlns from the SOAP message for this element exist, the element is properly validated.
2. When elem_ns ASI does not exist and the corresponding element from the SOAP message does have an xmlns specified, the data handler defaults the elem_ns to the last elem_ns read from the business object that was in the scope. The data handler compares this value with the xmlns value from the SOAP message. If there is no match, validation fails.
3. When elem_ns ASI does exist and the corresponding element from the SOAP message does not have xmlns specified, the data handler verifies that the elem_ns specified in ASI matches one of the namespaces in the current scope of the SOAP message. If there is no match, validation fails.

type_name and type_ns validation

The sections below discuss type_name and type_ns validation.

Simple attributes: The following rules apply to type_name and type_ns validation when xsdType is `true`:

- **Both type_name and type_ns are specified** Using the type_name and type_ns pair, the data handler creates a corresponding java Class object. Using the incoming SOAP message typename and typenamespace, another java Class object is queried. It the two java Class objects match, validation succeeds. Otherwise, validation fails.
- **Neither type_name nor type_ns are specified** The data handler maps the simple business object attribute to a java Class object. Using the incoming SOAP message typename and typenamespace, another java Class object is queried. If the two java Class objects match, validation succeeds. Otherwise, validation fails.
- **type_name only is specified** Simple Type Validation fails. Both type_name and type_ns or neither should be specified when xsdType is `true`.
- **type_ns only is specified** Simple Type Validation fails. Both type_name and type_ns or neither should be specified when xsdType is `true`

The following rules apply to type_name and type_ns validation when xsdType is `false`:

- **Both type_name and type_ns are specified** The data handler performs a direct comparison between the SOAP message typename and typenamespace pair and the type_name and type_ns values specified in ASI. If the pairs are exactly alike, validation succeeds. Otherwise, validation fails.
- **Neither type_name nor type_ns are specified** The data handler maps the simple business object attribute to a java Class object. Using the incoming SOAP message typename and typenamespace, another java Class object is queried. If the two java Class objects match, validation succeeds. Otherwise, validation fails.
- **type_name only is specified** The type_ns value defaults to the element namespace found in the business object ASI. Using this default type_ns and the type_name specified in ASI, the data handler performs a direct comparison between these values and the SOAP message typename and typenamespace. If the pairs are exactly alike, validation succeeds. Otherwise, validation fails.
- **type_ns only is specified** The type_name value defaults to the business object attribute type. Using this default type_name and the type_ns specified in ASI, the data handler performs a direct comparison between these values and the SOAP message typename and typenamespace. If the pairs are exactly alike, validation succeeds. Otherwise, validation fails.

Complex attributes (cardinality 1 and n): The following rules apply to type_name and type_ns validation when xsdType is `true`:

- **Both type_name and type_ns are specified** xsdType is ignored. The data handler processes as if xsdType is `false`.
- **Neither type_name nor type_ns are specified** xsdType is ignored. The data handler processes as if xsdType is `false`.
- **type_name only is specified** xsdType is ignored. The data handler processes as if xsdType is `false`.
- **type_ns only is specified** xsdType is ignored. The data handler processes as if xsdType is `false`.

The following rules apply to type_name and type_ns validation when xsdType is `false`:

- **Both type_name and type_ns are specified** The data handler performs a direct comparison between the SOAP message typename and typenamespace pair and the type_name and type_ns values specified in ASI. If the pairs are exactly alike, validation succeeds. Otherwise, validation fails.
- **Neither type_name nor type_ns are specified** The type_name value defaults to the business attribute type. The type_ns value defaults to the element namespace found in the business object ASI. Using this default behavior, the data handler performs a direct comparison between these values and the SOAP message typename and typenamespace pair. If the pairs are exactly alike, validation succeeds. Otherwise, validation fails.
- **type_name only is specified** The type_ns value defaults to the element namespace found in the business object ASI. Using this default type_ns and the type_name specified in ASI, the data handler performs a direct comparison between these values and the SOAP message typename and typenamespace. If the pairs are exactly alike, validation succeeds. Otherwise, validation fails.
- **type_ns only is specified** The type_name value defaults to the business object attribute type. Using this default type_name and the type_ns specified in ASI, the data handler performs a direct comparison between these values and the

SOAP message typename and typenamespace. If the pairs are exactly alike, validation succeeds. Otherwise, validation fails.

attr_name and attr_ns validation

While reading SOAP message into a business object, each SOAP element is searched for SOAP attributes. If found, these attributes are compared to the attr_name property values from the corresponding BO. For example, consider this SOAP message:

```
<CustInfo City="4" State="5" Street="2" Zip="6">
   <Name xsi:type="xsd:string">1</Name>
   <Street2 xsi:type="xsd:string">3</Street2>
</CustInfo>
```

Now consider the business object definition structure (with the attribute level ASI specified to the right of each attribute) shown inFigure 53.

Name	Type	App Spec Info
⊟ CustInfo	CustomerInfo	
Name	String	
Street1	String	attr_name=Street
Street2	String	
City	String	attr_name=City
State	String	attr_name=State
Zip	String	attr_name=Zip

Figure 53. attr_name and attr_ns validation

The data handler would follow these processing steps:

1. Read the element name CustInfo.
2. Resolve the business object attribute that corresponds to this element name.
3. Read the attributes of the SOAP element and attempt to match them against the ASI of the child attributes. In this case, the SOAP message Street matches the business object attribute Street1, City matches the business object attribute City and so on.
4. The child elements for CustInfo are read and processed in the same manner as the rest of the body.

Note: attr_ns is not validated.

The data handler loops through the SOAP attributes for a given element. For each attribute encountered, the data handler searches the business object for a corresponding attribute. If found, the business object attribute is populated with the value of the SOAP attribute. If a corresponding business object attribute is not found, the data handler continues to the next SOAP attribute.

Calling a data handler from within the SOAP data handler

The SOAP data handler can read an encoded element value from a SOAP message into a business object using another data handler. For example, an RNIF document may be one of the formats in which a SOAP element value is encoded. To make use of this functionality, an RNIF business object can be added at any level of a SOAP Child business object. To signify to the SOAP data handler that another data handler must be used when transforming this RNIF encoded String to an RNIF business object, you must add the `dh_mimetype` property to the attribute's ASI. The

value of the dh_mimetype ASI should be a legal mimeType specified in the MO_DataHandler_Default business object. The mimeType is used to determine which data handler to use on the String. For example, given the following SOAP message where RNIFExample is the SOAP element that contains an RNIF encoded String:

```
<CustInfo>
<Name>IBM Corporation</Name>
<CustID>95626</CustID>
<RNIFexample xsi:type="xsd:base64Binary">
1AWERYER238W98EYR923872837487189278 7ASRJK234234JKAWER
J234AWERIJHI423488R4HASF1AWERYER238W98EYR923872837487
1892787ASRJK234234JKAWERJ234AWERIJHI423488R4HASF1AWER
YER238W98EYR923872837487189278 7ASRJK234234JKAWERJ234A
WERIJHI423488R4HASF1AWERYER238W98EYR92387283748718927
87ASRJK234234JKAWERJ234AWERIJHI423488R4HASFWR234
</RNIFexample>
</CustomerInfo>
```

The SOAP business object would look like that shown in Figure 54.

Name	Type	App Spec Info
⊟ CustInfo	CustomerInfo	
Name	String	
Street1	String	attr_name=Street
Street2	String	
City	String	attr_name=City
State	String	attr_name=State
Zip	String	attr_name=Zip

Figure 54. RNIFExample business object

Note that the RNIFExample element contains an RNIF encoded String as its element value. Also, note that elem_name, elem_ns, type_name, type_ns and xsdtype ASI properties still remain relevant for this business object attribute.

Note: If the element value returned by the called data handler is encoded text, the type_name property must be set to base64Binary, the type_ns must correspond to an xsd namespace, and xsdtype must be set to true.

Default business object resolution

For SOAP to business object transformations, the SOAP data handler and web services connector adhere to a special contract of exchanging information to resolve business object names. The connector provides the SOAP data handler with a list of business object names mapped to BodyName and BodyNamespace pairs. In addition, if there is a defaultfault business object set in the TLO, this information is passed to the data handler. Given this information, the SOAP data handler processes using the following steps:

1. The data handler receives a SOAP message
2. The data handler determines if this is a SOAP request, response or fault message.
   a. If a SOAP request or response message, the data handler reads the BodyName and BodyNamespace from the first child element of the SOAP-ENV:Body element.
   b. If a SOAP fault message, the data handler reads the BodyName and BodyNamespace from the first child element of the detail element in the

fault message. If there is no detail element in the fault message, the data handler uses the defaultfault business object for this transformation

3. If a defaultfault business object has not already been chosen, the data handler attempts to match the BodyName and BodyNamespace found in step 2 to the pairs found in the list provided by the connector. If a match is made, business object resolution is successful. If no match is made, the data handler fails with a meaningful error message.

Specifying a pluggable name handler

With default business object resolution, you can specify a pluggable name handler to determine the business object to be used in SOAP-message-to-business-object transformations. You do this by changing an MO_DataHandler_DefaultSOAPConfig attribute.

The MO_DataHandler_DefaultSOAPConfig has, among others, two attributes of type `string` that designate:

- **ClassName** The class name for the SOAP data handler base class. You do not change this attribute value when specifying a pluggable name handler.
- **SOAPNameHandler** The SOAPNameHandler attribute dictates which name handler is called. You can specify a value for a pluggable name handler. The value of this property should be a class name. The SOAPNameHandler class is an abstract class with the following signature:

  ```
  public abstract String getBOName(Envelope msgEnv, SOAPProperty prop)
  ```

If the SOAPNameHandler attribute has a value, the SOAP data handler calls the specified name handler. If the value does not exist, or if the specified name handler fails to get a business object name, the SOAP data handler is called by default to perform default business object resolution.

The SOAP DataHandler uses the SOAPNameHandler property specified in the MO to instantiate the custom-name-handler class. It then calls the getBOName to resolve the business object name. The SOAP DataHanlder passes the SOAPProperty object it received from the connector to the custom-name-handler implementation class.

This SOAPProperty object contains a structured list of potential candidate BOs for resolution. Contained in the list are BodyName, BodyNamespace and BOName triplets. These triplets are based on the SOAP Config MO configuration information. The Default Name Handler uses this object to resolve the BO. A custom name handler developer may use this object at their discretion.

Using the SOAPProperty object

You use the SOAPPropertyUtils class to extract the business object name from the SOAPProperty. To do so, use the following method:

```
/**
 * Retrieve the business object name based on the body name and the body
* namespace
*.
* @param soapProp top level SOAPProperty object that is passed by the
* connector
* @param name body name from the SOAP message
* @param uri body namespace from the SOAP message
* @return business object name from the SOAPProperty object with the body
```

```
* name and body namespace.
*/
java.lang.String findBOName(SOAPProperty soapProp, String name, String uri);
```

Sample NameHandler

The following is a sample NameHandler:package

```
com.ibm.adapters.datahandlers.soap.namehandlers;
// DOM and Parsers
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.xml.sax.InputSource;
// Apache Xerces and SOAP
import org.apache.soap.Envelope;
import org.apache.soap.Header;
import org.apache.soap.Body;
import org.apache.soap.Constants;
import org.apache.soap.util.xml.DOMUtils;
import org.apache.soap.util.xml.XMLParserUtils;
import org.apache.soap.util.xml.QName;
import org.apache.soap.encoding.soapenc.SoapEncUtils;
import org.apache.soap.encoding.soapenc.Base64;
// java
import java.util.Vector;
// SOAP data handler
import com.ibm.adapters.datahandlers.soap.*;
import com.ibm.adapters.datahandlers.soap.exceptions.*;
public class MyCustomNameHandler extends SOAPNameHandler {
  private static final String BOPREFIX = "MyCustomBOPrefix";
  private static final char UNDERSCORE = '_';
  private static final char EMPTY_STRING ="";

  public String getBOName(Envelope msgEnv, SOAPProperty prop)
    throws SOAPNameHandlerException
  {
    // Initialize a String Buffer
    StringBuffer boName = new StringBuffer();
    // Determine the "MyCustomBOPrefix" SOAP data handler
    // MO property. If it exists, and is populated append
    // this prefix to the front of the BOName.
    String pref = dh.getOption(BOPREFIX);
    if (pref != null) {
       boName.append(pref.equals(EMPTY_STRING)
       ? EMPTY_STRING : pref + UNDERSCORE);
  }
    // Begin parsing the SOAP msg envelope.
    Element bodyEl, requestEl;
    Body msgBody = msgEnv.getBody();
    Vector bodyEntries = msgBody.getBodyEntries();
    if((bodyEntries == null) || (bodyEntries.size() <= 0))
       throw new SOAPNameHandlerException("No Body Entries exist
        for this SOAP message. Cannot determine BOName to use.");
    // Grab the first <SOAP-ENV:Body> Element
    bodyEl = (Element) bodyEntries.elementAt(0);
    // Grab the first Child Element of the <SOAP-ENV:Body>
    // Element
    requestEl = (Element) DOMUtils.getFirstChildElement(bodyEl);
    // Read the name and namespace of this first child
    String name = bodyEl.getLocalName();
    String uri = bodyEl.getNamespaceURI();
    if (uri == null)
       uri = Constants.NS_URI_SOAP_ENV;
    // Use the SOAPPropertyUtils findBOName() method to search
    // the SOAPProperty object for this messages first element
    // name and namespace. If no match is found, a
    // SOAPDataHandlerException will be thrown. If a match is
```

```
    // found, and it's not an empty string, append to the boname.
String returnedBOName = SOAPPropertyUtils.findBOName(prop, name, uri);
if (returnedBOName != null &&
       !returnedBOName.equals(EMPTY_STRING))
  boName.append(returnedBOName);
       return boName.toString()
   }
}
```

Limitations

The sections below discuss data handler limitations.

SOAP style and use guidelines

SOAP messages are created using a style and use defined by the web service. The SOAP data handler provides the levels of support shown in Table 46.

Table 46. Style and use guidelines

Style	Use	Parts defined using	Data handler support
document	literal	element	full
document	literal	type	limited (see below)
document	encoded	element	none
document	encoded	type	limited (see below)
rpc	literal	element	none
rpc	literal	type	full
rpc	encoded	element	none
rpc	encoded	type	full

Part and part element order

When the SOAP data handler is transforming a SOAP message into a business object and the SOAP message follows either the document/literal/type or document/encoded/type formats, the message parts must be in the order described in the WSDL. For example, consider the following WSDL:

```
<operation name="GetQuote"
     style="document" ...>
<input>
   <soap:body parts="Part1 Part2 Part3 Part4" use="literal">
</input>
</operation>

<definitions
 xmlns:stns="(SchemaTNS)"
xmlns:wtns="(WsdlTNS)"
targetNamespace="(WsdlTNS)">

<schema targetNamespace="(SchemaTNS)"
                elementFormDefault="qualified">
<element name="SimpleElement" type="xsd:int"/>
<element name="CompositElement" type="stns:CompositeType"/>
<complexType name="CompositeType">
<all>
      <element name='elem_a' type="xsd:int"/>
      <element name='elem_b' type="xsd:string"/>
</all>
</complexType>
</schema>
```

```
<message...>
<part name='Part1' type="stns:CompositeType"/>
<part name='Part2' type="xsd:int"/>
<part name='Part3' element="stns:SimpleElement"/>
<part name='Part4' element="stns:CompositeElement"/>
</message>
Ö
</definitions>
```

The SOAP message must adhere to the order defined by the parts. In the SOAP example below, notice that Part1 elements precede Part2, Part3, and Part4 elements. This order must be maintained for proper BO resolution.

```
<soapenv:body... xmlns:mns="(MessageNS)"
   xmlns:stns="(SchemaTNS)">
   <stns:elem_a>123</stns:elem_a>
   <stns:elem_b>hello</stns:elem_b>
   <soapenc:int>123</soapenc:int>123</soapenc:int>123</soapenc:int>
   <stns:SimpleElement>123</stns:SimpleElement>
   <stns:CompositeElement>
      <stns:elem_a>123</stns:elem_a>
      <stns:elem_b>hello</stns:elem_b>
   </stns:CompositeElement>
</soapenv:body>
```

When the SOAP message follows either the document/literal/type or document/encoded/type formats, part elements must be in order, too. In Part1 of the example above, the `elem_a` tag must precede the `elem_b` tag. This limitation is dictated by the data handler's business object resolution process. Since default business object resolution for document style makes use of the first element's body name and namespace, these must be the same element in all SOAP messages of this particular request, response, or fault so that the same business object is resolved in each case.

Note: When the SOAP message follows either the document/literal/type or document/encoded/type formats, elements must not be optional.

XML limitations

The following XML structures, features, and notation are not supported:

- Multi-dimensional arrays
- Partially transmitted arrays
- Sparse arrays
- Mixed content
- Sequence, group, and choice model group components with maxOccurs greater than one

Chapter 6. Enabling collaborations for request processing

This chapter describes the steps you must follow to enable collaborations for request processing. Collaborations use the connector to invoke web services.

Request processing collaboration checklist

Using Business Object Designer to generate business objects is part of the process of developing collaborations. You must perform the following tasks, described in sections below, to generate business objects that a collaboration can use to invoke web services:

1. Identify the WSDL document either from a URL, UDDI or a file system. You use third-party tools for this task—the web services connector provides no tools for this task.
2. Open Business Object Designer and launch the WSDL ODA. For further information, see "Starting the WSDL ODA" on page 155.
3. Configure the ODA.
4. Confirm your selections.
5. Generate a top-level business object that includes Request and (for synchronous requests) Response and Fault business objects as well as SOAP Config MOs, Protocol Config MOs, header container and child objects and application-specific information appropriate to each object and attribute. The WSDL ODA automates this process.

After you generate business objects, you must perform tasks to enable a collaboration to invoke a web service using the connector and the SOAP data handler. For steps on developing a collaboration, including creating a collaboration template and object and binding its ports, see *IBM WebSphere InterChange Server Collaboration Development Guide*. For further information on creating maps between generic business objects and the application-specific business objects generated by the WSDL ODA, see *IBM WebSphere InterChange Server Map Development Guide*.

Chapter 7. Exposing collaborations as web services

This chapter describes the design-time procedure of exposing a collaboration as a web service. This enables the connector to process events when a web service client invokes a collaboration.

Integrated design tools simplify the task of exposing a collaboration as a web service. After configuring the collaboration and business objects for web services, you use the WSDL Configuration Wizard. The wizard creates a WSDL document and XML schema that represent the collaboration as a web service. The WSDL outputs not only describe the collaboration but form the basis for its invocation by a web service client.

Procedure checklist

You must perform the following tasks, described in the sections below, to expose a collaboration as a web service:

1. Identify or, as needed, develop the business objects for use as request and optionally (for synchronous event processing) response and fault SOAP messages. There are two ways to generate these objects: 1) manually, using Business Object Designer, or 2) if a WSDL interface file exists for your web service, you can use the WSDL ODA to generate the Request and other (Response or Fault) business objects. If you are following the second approach:
   a. Specify the name of the collaboration in the Collaboration WSDL ODA configuration property. This value dictates the ws_collab ASI in the TLO.
   b. Specify either a WSDL_URL or UDDI_InquiryAP I_URL WSDL ODA configuration property for the WSDL interface file (you can also specify a directory path to this file, if it resides on your network or locally).

   For further information, see ."Starting the WSDL ODA" on page 155.
2. Develop a collaboration template or choose an existing one to use the business objects.
3. Create the collaboration object and its ports for the web service.

   You first must ensure that the collaboration object properly populates business objects. For more information and a step-by-step procedure for creating a collaboration object, see the *Implementation Guide for WebSphere InterChange Server*.

   Note: The collaboration object must have its maps configured for the appropriate transformations. Maps convert the business object received in the SOAP request message to the business object used by the

collaboration. Maps also convert the business object returned by the collaboration to the business object that is embedded in the SOAP response message. For more information about mapping and mapping procedures, see the *Map Development Guide*.

4. Use the WSDL Configuration Wizard to create the WSDL document. The utility also configures the web services connector.

   Note: The WSDL Configuration Wizard creates implementation, interface, and one or more schema files. This document refers to these outputs collectively as the WSDL document.

5. Publish the WSDL document as required.

   Note: The connector provides neither tools nor support for publishing WSDL documents.

Identifying or Developing Business Objects

You use Business Object Designer to create business objects and Connector Configurator to configure the connector to support them.

For more information on Business Object Designer, see the *Business Object Designer*. For detailed information on web services business objects, see Chapter 3, "Business object requirements," on page 21.

Choosing or developing a collaboration template

The collaboration template you choose or develop must have one or more scenarios to expose as a web service. For further information on collaboration templates, see *Collaboration Development Guide*.

Binding the port of a new collaboration object

After you have configured the port of a collaboration template for a business object type you must create the collaboration object and bind its port to an instance of a web services connector.

To create a new collaboration object and bind its port to an instance of the web services connector:

1. Right click the Collaboration Objects folder and select Create New Collaboration Object. This displays the Create New Collaboration window, which displays the list of templates (as shown in Figure 55).

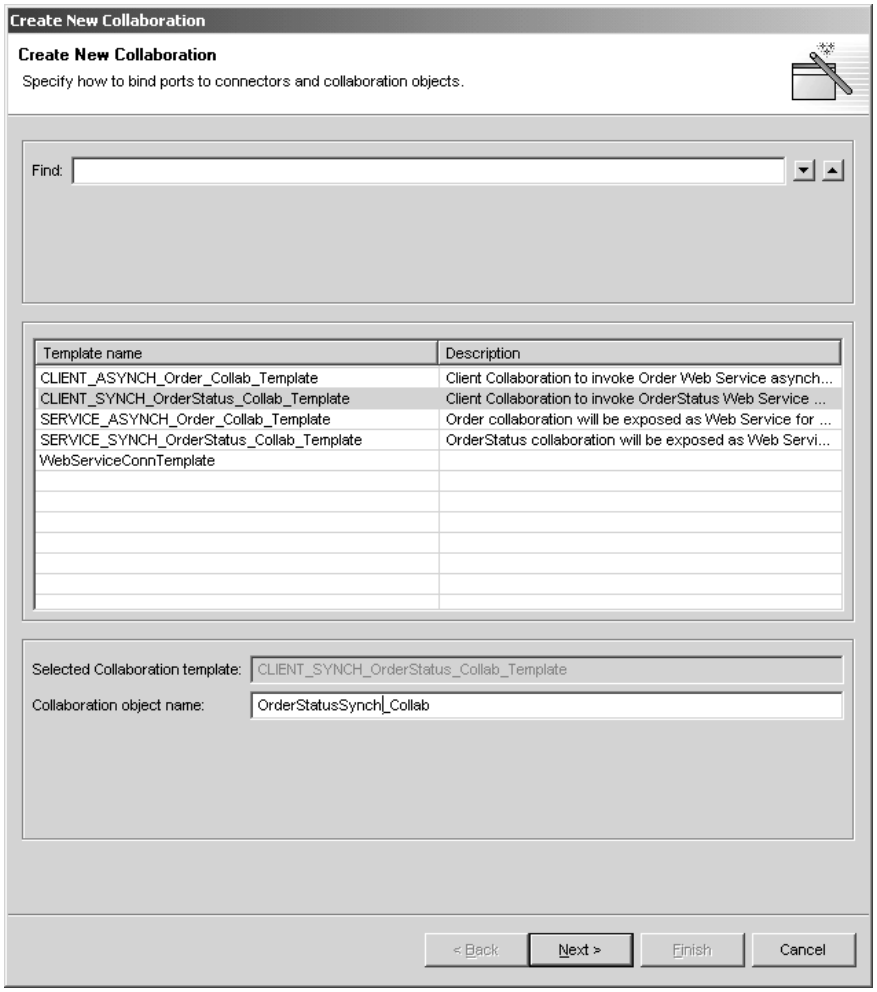


Figure 55. Create New Collaboration window

2. Select a collaboration template from the Template Name and enter a name for the collaboration object in Collaboration object name field. This displays the Bind Ports window as shown in Figure 56.

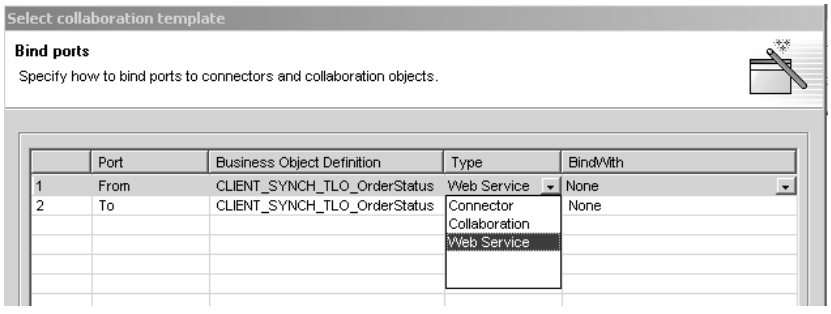


Figure 56. Bind Ports window

3. Select a port, click the Type arrow to display the pull down menu for the port and choose WebSerivce (as shown in Figure 56)

   All instances of the web services connector have a ConnectorType application-specific property. By default, this property is set to `WebService`. The

Bind Collaborations Port window in System Manager uses the value of the ConnectorType property to determine which connectors are web service connectors.

4. Click the BindWith arrow to display a list of connector instances. System Manager displays instances of connectors whose ConnectorType properties have values set to `WebService`. Choose an instance of the web services connector. (An example is shown in Figure 57).

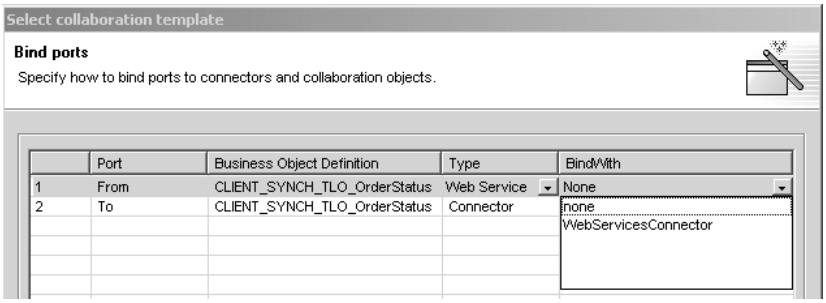


Figure 57. Selecting an instance of the web services connector

5. Click Finish.

You are now ready to run the WSDL Configuration Wizard.

WSDL Configuration Wizard

After you have created the collaboration object and bound its triggering port to an instance of a web services connector, you are ready to use the WSDL Configuration Wizard. Using binding, port name, operation and other data you specified for the collaboration, business object definition, and connector, the utility produces the a WSDL implementation file (`*.impl.wsdl`), a WSDL interface file (`*.wsdl`), and an xml schema file (`*.xsd`). These files are a composite of the collaboration exposed as a web service, and the utility allows you to specify whether to generate these as separate files or as one file. The utility supports SOAP over HTTP, HTTPS, and JMS protocols. Configuration information for the protocol listener framework is retrieved from the connector-specific property ProtocolListenerFramework. This property also makes the list of listeners available.

Running the wizard

To run the WSDL Configuration Wizard:

1. Right-click a collaboration object that you have configured for web services and choose Expose as a web service in the popup menu. The WSDL Configuration Wizard displays as shown in Figure 58

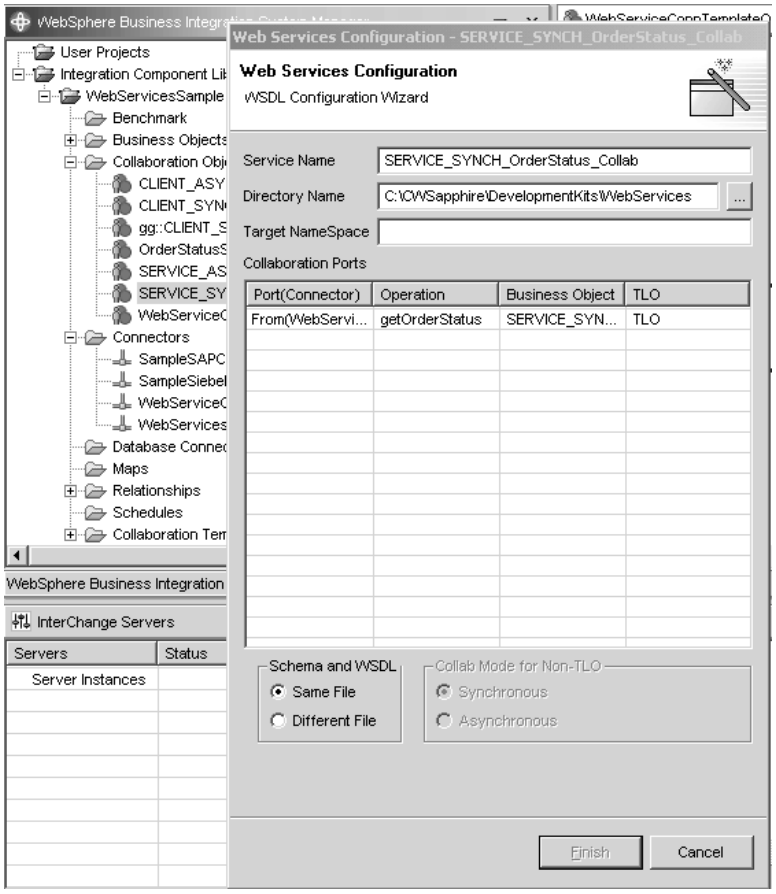


Figure 58. WSDL Configuration Wizard

As shown in Figure 58, the columns are as follows:

- **Port (Connector)** The triggering port on the collaboration object that is bound to a web services connector. The wizard gets this information from the collaboration object.
- **Operation** If the business object is a TLO, the wizard gets this information from the Request business object's SOAP Config Mo BodyName attribute. If the business object is a non-TLO, then the wizard combines the business object name and the port name.
- **Business Object** Used to create the schema. The wizard gets this information from the connector's supported business objects for this triggering port.

2. Enter the following as needed:
   - **Service Name** By default, the name you used to describe the collaboration object
   - **Directory Name** Where the adapter for web services and collaboration templates and objects reside
   - **Target NameSpace** The URL for the collaboration being exposed as a web service.
   - **Collaboration Ports** The information in these fields are as specified in the Bind Ports window of the collaboration object configuration procedure.
   - **Collaboration Mode for Non-TLO** This does not apply if you are using TLOs. Otherwise, if you using a non-TLO object as input, you must specify synchronous or asynchronous.

- **Schema and WSDL** Specify whether you want these outputs in a single file or in separate files.

3. Click Finish. The utility generates outputs based on the inputs and specifications you entered, all of which are summarized in the next section.

WSDL Configuration Wizard processing of business objects in TLO format

The configuration wizard creates a WSDL operation for each triggering port of a collaboration object that is bound to a web services connector. The creation of the operation is based on the business objects that are associated with the invocation of this collaboration.

The configuration wizard determines that a business object is in the TLO format by reading the object-level ASI `ws_eventtlo`. If the ASI property is set to `true`, the business object is a TLO. Using the TLO, the following WSDL properties are found:

- **Operation Name and BodyNS** When the wizard finds business objects in TLO format, it creates an operation name using the BodyName property of the SOAP Config MO within the SOAP Request business object of the TLO. Similarly, the wizard determines the message namespace to be the BodyNS property in the same SOAP Config MO
- **Execution Mode** By inspecting the `ws_mode` property from the business object level ASI of the TLO, the wizard determines that the mode is either synchronous or asynchronous, and creates a REQUEST_RESPONSE or ONE_WAY WSDL, respectively.

To create WSDL operations based on TLOs, a collaboration can be configured in two ways, with and without maps.

TLOs with maps: A collaboration is generally configured to accept Generic Business Object (GBO) requests. That is, the collaboration template triggering ports subscribe to GBOs. To use TLOs in this case, the collaboration must be bound to a web services connector, and the connector must support the transformation of the GBO to TLOs via maps. Figure 59 shows this scenario.

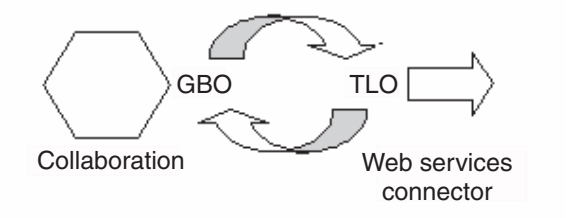


Figure 59. TLO with map

When the collaboration and connector are configured in this way, the wizard determines that the TLO business object will be used to create the operations described in the WSDL document. This determination is made by inspecting the connector-supported business objects and associated maps. It is important for the run-time processing of the web services connector that the configured maps always transform the collaboration's GBO to one and only one TLO. Also, it is important that the source and destination business objects of the inbound map translate to the destination and source business objects of the outbound map, respectively.

TLOs without maps: The wizard also supports processing TLOs without maps. In this case, the collaboration template's triggering ports subscribe to TLOs directly.

Because the web services connector supports the TLOs, maps are not required. Figure 60 illustrates this scenario.

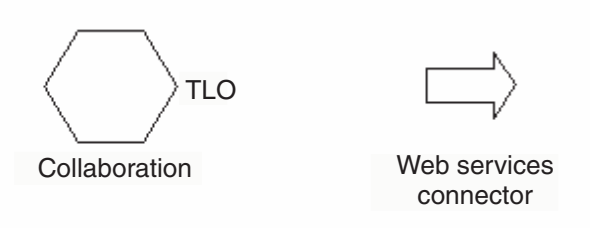


Figure 60. TLO without map

When the collaboration and connector have been configured in this way, the wizard uses the TLO business object found in the collaboration to create the operations described in the WSDL document. The wizard determines that no maps are configured for this port.

WSDL Configuration Wizard processing of business objects in non-TLO format

Support for non-TLO business objects allows you to use pre-existing collaborations and maps for exposing as web services. For this reason the wizard also supports creating WSDL operations using business objects that are not in TLO format.

Similar to the TLO process, the wizard determines that a business object is in non-TLO format by reading the object-level ASI `ws_eventtlo`. If the ASI property does not exist or exists but is set to something other than `true`, this business object is a non-TLO. A non-TLO is any business object that does not adhere to the web services TLO structure. Using the non-TLO, the wizard discovers the following properties:

- **Operation Name and BodyNS** When the wizard finds business objects in non-TLO format, it creates an operation name using a combination of the collaboration name, the business object name, and the port name. The Body Namespace for the WSDL operation is configured using the Target Namespace entry in the WSDL Configuration Wizard.
- **WSCollaborations** The wizard creates a hierarchy of properties in the web services connector that includes a BO Name, a SOAP Body Name, a SOAP Body Namespace, and a Mode for each WSDL operation in a port of a collaboration that is exposed as a web service. Figure 61 shows a sample WSCollaborations property:

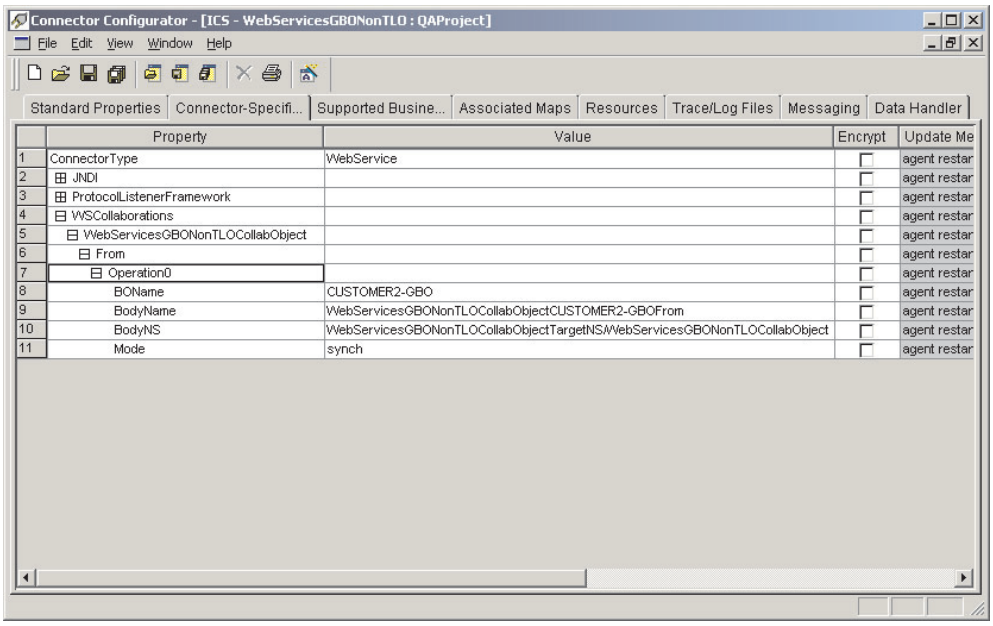


Figure 61. WSCollaborations

- **Execution Mode** The Execution mode for the WSDL operation is configured using the Collab Mode for Non-TLO selection button in the WSDL Configuration Wizard.

To create WSDL operations based on non-TLOs, a collaboration can be configured in two ways, with and without maps.

Non-TLOs with maps: Collaborations are generally configured to accept Generic Business Object (GBO) requests. At the same time, there may be pre-existing maps that transform the GBO from the collaboration to a non-TLO business object. Figure 62 shows this scenario.

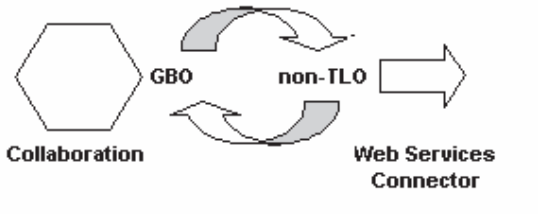


Figure 62. Non-TLO with map

In this case, the wizard uses the non-TLO business object to create WSDL operations described in the WSDL document. It is important for the run-time processing of the web services connector that the configured maps always transform the collaboration's GBO to one and only one non-TLO. Also, it is important that the source and destination business objects of the inbound map translate exactly to the destination and source business objects of the outbound map respectively.

Non-TLOs without maps: In highly specialized cases, collaborations may be configured to accept requests from business objects other than GBOs. In this case,

the non-TLO is a direct business object for the collaboration, and no maps exist. Figure 63 shows this scenario.

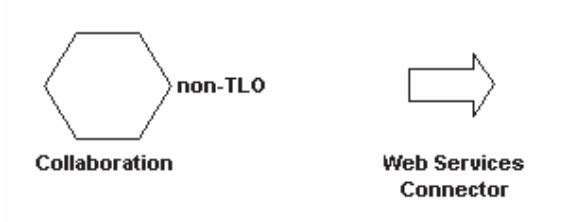


Figure 63. Non-TLO without map

In this case, the wizard determines that no maps are configured for this port, so it uses the non-TLO business object to create WSDL operations described in the WSDL document.

Processing requirements and exceptions

The sections below discuss requirements of the WSDL Configuration Wizard that apply to all types of objects (TLOs and non-TLOs) unless otherwise explicitly mentioned. For further information on business object requirements for web services TLOs, see Chapter 3, "Business object requirements," on page 21.

Note: Among the business object ASI that the WSDL tool reads, only the following can have internationalized characters:

- elem_name
- elem_ns
- attr_name
- attr_ns
- BodyName
- BodyNS
- type_name
- type_ns

Support for Use property in SOAP Config MO: The WSDL Configuration Wizard supports the Use property in SOAP Config MOs, but throws an error if the Use value in a SOAP Request BO and the corresponding SOAP Response BO are different. You can set the Use value to literal or encoded to generate a WSDL document. For more information on the Use property and its values, see "Style and Use impact on SOAP messages" on page 111.

Support for Style in SOAP Config MO: Only rpc style is supported for exposing collaborations as web services. If the Style is specified as document in the SOAP Config MO, the wizard will throw an error.

Fault processing: The details attribute inside a SOAP Fault business object can have one child attribute only. Otherwise, the utility generates an error.

The utility accepts Fault business objects. If it encounters multiple Fault business objects, the utility processes the header container of the first or default fault business object. Processing is as follows:

- No Namespace is specified for the soap:fault element inside the binding section.
- Fault is always specified using the document style and use literal.

- Message parts are specified using the element attribute.

Header fault processing: A header fault is processed as soap:headerfault, a child element of soap:header inside the WSDL document binding section. The header fault is processed using the `headerfault` ASI specified in the header child business object as follows:

- No Namespace is specified for the soap:headerfault element.
- A header fault is always specified using the document style and use literal.
- Message parts are specified using the element attribute instead of the type attribute.

Header Processing: Multiple header attributes are specified as SOAP header child business objects inside a SOAP header container business object. A Header container business object is identified by its ASI: `soap_location=SOAPHeader`. During utility processing, a soap:header element is created inside binding section for each of the attributes inside the header container business object and the following rules apply:

- The header is always specified using document style and use literal.
- Message parts are specified using the element attribute instead of the type attribute.
- If no elem_ns is specified, headers are written to the Body Namespace.

Note: The header container business object can be a child of SOAP Request, Response or Fault business objects. The namespace attribute is not specified for the soap:header element.

elem_ns ASI processing: The utility ignores elem_ns ASI at the message part level. Instead, elem_ns is used in second- and lower-level attributes. Second- level business object attributes can be defined in a separate namespace if elem_ns is specified.

JMS protocol processing: SOAP/JMS binding in the port section of the WSDL document contains the jms:address element. The following is an example of jms:address element. (Attributes suffixed with ″?″ are optional).

```
    <jms:address
                    destinationStyle = "queue"
jmsVendorURI = "http://ibm.com/ns/mqseries"?
initialContextFactory = "com.ibm.NamingFactory"?
jndiProviderURL = "iiop://something:900/wherever"?
        jndiConnectionFactoryName = "orange"
        jndiDestinationName = "fred"
        jmsProviderDestinationName="trash" />
```

If the LookupQueuesUsingJNDI connector property is set to `true`, the value of InputQueue property corresponds to the jndiDestinationName attribute of the jms:address element of the SOAP/JMS binding. The jms:address element is specified in the wsdl:port section. If LookupQueueUsingJNDI is set to `false`, then the jmsProviderDestinationName attribute is set to InputQueue. InputQueue is the connector property available under the Listener_JMS hierarchical property. The initialContextFactory, jndiProviderURL and jndiConnectionFactoryName properties will be specified only for synchronous processing.

HTTP protocol processing: A sample port section from a WSDL document is shown below:

```
<service name="StockQuoteWebService">
<port name="StockQuoteWebServicePort" binding="intf:StockQuoteBinding">
<soap:address location="http://localhost:8080/wbia/webservices/stockquoteservice"/>
</port>
</service>
```

The WSDL Configuration Wizard uses the value of host name and the port from the context path. If the context path contains only the relative path without the host name and port, then the value of host name and port property located under the Listener_HTTP configuration property will be used to specify the location attribute in soap:address xml element.

Chapter 8. Using the WSDL ODA

Note: The Web Services Description Language (WSDL) Object Discovery Agent (ODA) is used for generating business objects for request processing and, when a WSDL Interface file is available, for event processing.

Collaborations use the connector to invoke web services. Or you can expose collaborations as web services. Web services are described using WSDL (Web Services Description Language). This chapter describes how to use the Web Services Description Language (WSDL) Object Discovery Agent (ODA) to generate business objects. The connector and SOAP data handler use these business objects when collaborations invoke a web service and when exposing collaborations as web services.

You use the WSDL ODA to generate business objects for two purposes:

1. The WSDL ODA can take a WSDL implementation file and generate business objects for a collaboration to invoke an external web service.
2. The WSDL ODA can take a WSDL interface file and generate business objects for a collaboration that is exposed as a web service.

You can launch the WSDL ODA when you use the Business Object Designer. The WSDL ODA reads a WSDL document and creates the business objects required by the connector and SOAP data handler. The WSDL ODA simplifies the job of business object development.

Note: The WSDL ODA handles SOAP/HTTP and SOAP/JMS bindings in a WSDL.

Starting the WSDL ODA

You can start the WSDL ODA using one of the following scripts:

- Windows
  - start_WSDLODA.bat

    Note: You can also start the WSDL ODA using the shortcut that the Installer automatically creates for Windows environments.
- UNIX
  - start_WSDLODA.sh

You select, configure, and run the WSDL ODA using Business Object Designer. Business Object Designer locates each ODA by the name specified in the AGENTNAME variable of each script or batch file.

Running the WSDL ODA

An Object Discovery Agent (ODA) simplifies the work of building business objects for request processing. Business Object Designer provides a graphical interface to all available ODAs, and helps you find the agent you need. The WSDL ODA is named, by default, WSDLODA. The name as it appears in the WSDL Wizard depends on the value of the `AGENTNAME` variable in the `start_WSDLODA.bat` or `start_WSDLODA.sh file`. For more on ODAs and business object definitions and how to configure, start and use ODAs, see the *IBM WebSphere Business Object Development Guide*. You are encouraged to consult that document as needed while following the procedures below.

After starting the Object Discovery Agent, follow these steps to launch the WSDL ODA:

1. Open Business Object Designer.
2. From the File menu, select the New Using ODA... submenu. Business Object Designer displays the Select Agent dialog box in the Business Object Wizard. Figure 64 illustrates this window.
3. Click the Find Agents button to display all running agents and select the WSDL ODA.

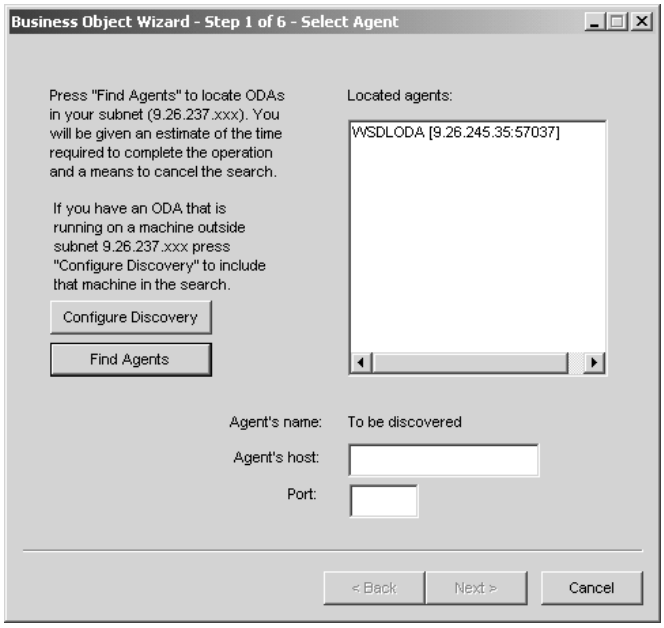


Figure 64. Select Agent window

If Business Object Designer does not locate your WSDL ODA, check the setup of the ODA.

4. Select the WSDL ODA in the Located Agents pane list and click Next.

   This displays the Configure Agent wizard window, which shows the configuration properties you need to specify.

Configuring the agent

Figure 65 shows the Configure Agent window of the WSDL ODA Business Object Wizard.

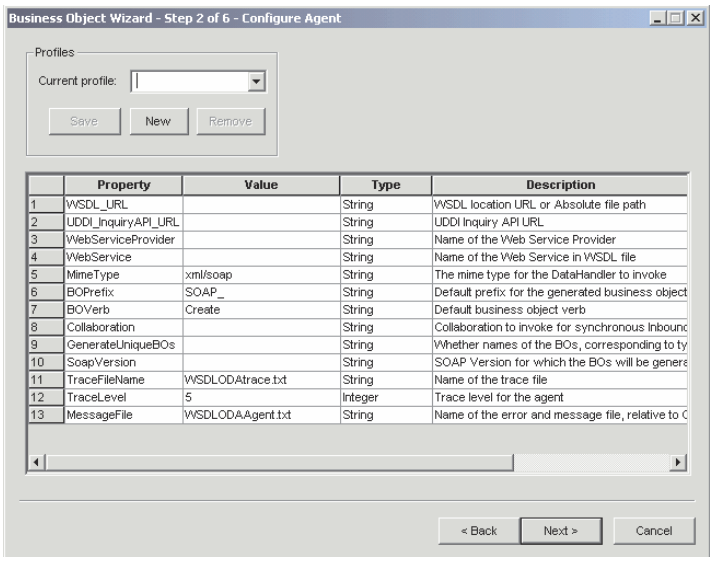


Figure 65. Configure Agent window

Table 47 lists the properties you must configure for the WSDL ODA.

Note: The first time you use the WSDL ODA, you must specify values for each configuration properties. After doing so, you can save the property values in a profile by clicking the Save button. The next time you use the WSDL ODA, you can select the saved profile from the "Select profile" box.

Table 47. WSDL ODA configuration properties

Property	Type	Required	Default	Description
WSDL_URL	`String`	Yes, when not specifying a UDDI	None	The URL of the WSDL document. This value can also be set to the absolute path to a local WSDL file. You can specify the URL in a native language.
UDDI_InquiryAPI_URL	`String`	Yes for UDDI	None	The URL of the UDDI inquiry API.
WebServiceProvider	`String`	Yes for UDDI	None	The name of the target web service provider. This is normally the Business name as published on the UDDI registry. This entry is case sensitive and requires English characters only.
WebService	`String`	Yes for UDDI		The name of the web service. This entry is case sensitive and requires English characters only.

Table 47. WSDL ODA configuration properties (continued)

Property	Type	Required	Default	Description
MimeType	String	No	xml/soap	The mime type of the data handler that the connector invokes. This is set in the business object TLO as the default value and must be in English characters only.
BOPrefix	String	No	SOAP_	This is appended to the front of every business object created. User configurable (English characters only) up to eight characters.
BOVerb	String	Yes	Create	The verb set in the SOAP Config MO of the Request, and, optionally, Response, and Fault business objects.
Collaboration	String	No	None	This value dictates the ws_collab ASI in the TLO and is mandatory when generating objects for event processing.
GenerateUniqueBOs	Sring	No	true	If this property is true, the business object names will be unique among all web services. If this property is false, you can reuse the business objects among operations with the same part types.
SOAPVersion	String	No	1.1	Determines the SOAP standard used to generate BOs. Possible values are 1.1 and 1.2.

The next section describes how to specify the WSDL document in the Configure Agent window.

Specifying the WSDL document

Web service business objects are generated from WSDL documents. This section shows you how to select and specify the source of a WSDL document in the Configure Agent window of the ODA.

The WSDL document may reside on the local file system or at a URL location on the web or in a UDDI registry—you specify where the WSDL document resides and the WSDL ODA retrieves it. (A complete WSDL service description may consist of more than one document.)

Getting a WSDL document from a URL location

As shown in Figure 65 above:

1. Specify the URL for the WSDL document in the configuration property WSDL_URL

   The ODA then retrieves the list of web services from the WSDL document, resolving the URLs of imported documents. The WSDL_URL property also allows you to specify the location of the WSDL file on the local file system using URL syntax (for example: `file://C:/test/wsdl`) or an absolute path (for example: `C:\test\wsdl`). You must ensure that the ODA has access to this document and its dependencies (all the imported documents).
2. Click Next.

   The ODA queries the URL for the web service provider and retrieves the list of services defined in the WSDL at this URL location and then displays the list as shown in Figure 66.

   Note: The WSDL ODA displays the ports that have SOAP/JMS or SOAP/HTTP bindings only and excludes other types of bindings.

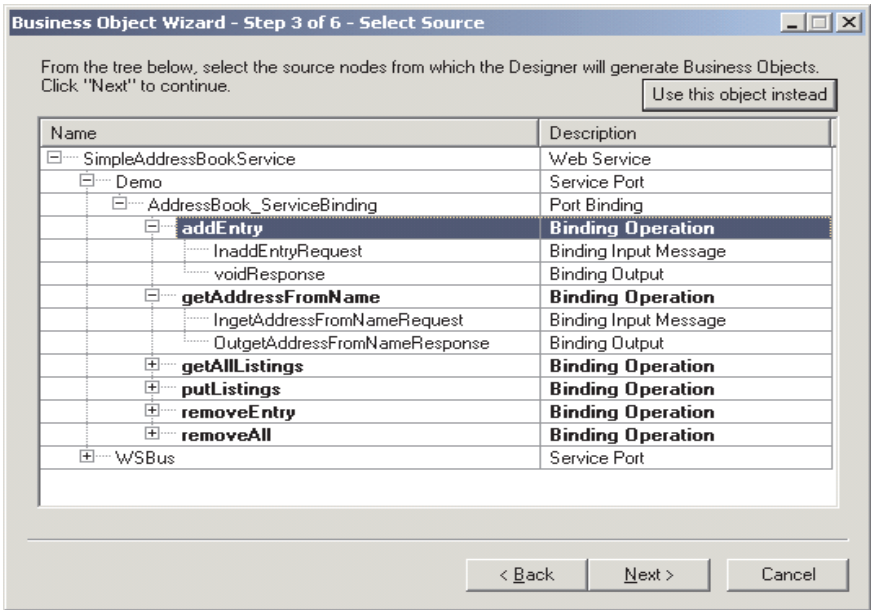


Figure 66. Select Source window

3. Select one and only one of the operations from the list for the port (the selectable operations are highlighted). You cannot select the service or port nodes, which are for display purposes only. Note that WSDL operations may be of several types: ONE_WAY, REQUEST_RESPONSE, SOLICIT_RESPONSE, and NOTIFICATION. The WSDL ODA supports and displays only REQUEST_RESPONSE and ONE_WAY operations.
4. Click Next and go to “Confirming selections” on page 160.

Getting a WSDL document from a UDDI registry

The ODA can also retrieve a WSDL document from a UDDI registry instead of a URL location. For this to occur:

1. Specify the following properties in the Configure Agent window for your “search key”:
   - UDDI_InquiryAPI_URL (for example: https://uddi.ibm.com/ubr/inquiryapi)
   - WebServiceProvider (for example: IBM Corporation)

- WebService (for example: StockQuoteService)
- The WSDL ODA uses exact name match (findQualifier) for inquiry within the UDDI registry. Ensure that you are entering the right values for the parameters. You can use a regular UDDI browser to find services provided by the service provider.

The WSDL ODA uses these properties, which are described in Table 47, to connect to the UDDI registry.

2. Click Next.

   The ODA queries the UDDI registry for the web service provider and retrieves the list of services matching the web service parameter you specified. The WSDL ODA displays the list of services offered by the web service provider in a window like that shown in Figure 66 When the UDDI query returns more than one match, the WSDL ODA displays them appended with an underscore (_) and a sequence number. For example: StockQuoteService_1, StockQuoteService_2, and so on.

   Note: The WSDL ODA displays the ports that have SOAP/JMS or SOAP/HTTP bindings only.

3. Select one and only one of the operations from the list for the port. You cannot select the service or port nodes, which are for display purposes only. Note that WSDL operations may be of several types: ONE_WAY, REQUEST_RESPONSE, SOLICIT_RESPONSE, and NOTIFICATION. The WSDL ODA supports and displays only REQUEST_RESPONSE and ONE_WAY operations.
4. Click Next and go to “Confirming selections”

Note: The connector supports the UDDI Version 2 API only. Accordingly, you cannot retrieve WSDL from UDDI registries that do not support UDDI Version 2.

Confirming selections

After selecting a web service operation source, the WSDL ODA Business Object Wizard displays a confirmation screen like that shown in Figure 67:

1. Confirm your selections.
2. Click Next and go to “Generating the objects” on page 161.

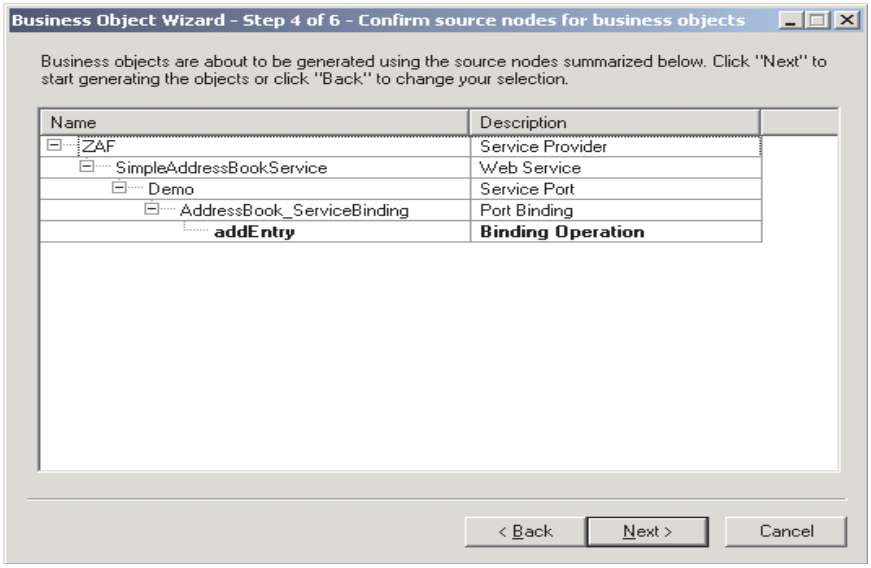


Figure 67. Confirm window

Generating the objects

After you confirm your WSDL document sources, the WSDL ODA generates the business objects and meta-objects for the web service you wish to invoke or for the collaboration you want to expose as a web service.

Note: The WSDL ODA cannot automatically select a key attribute for the top-level business object. For business objects at all other levels, the WSDL ODA sets the first attribute as the key. Accordingly, when you save WSDL ODA-generated objects in Business Object Designer, an error message informs you that the top-level object is missing a key attribute. Assign a key attribute that reflects your business data and business object requirements, then re-save the objects. Use caution when selecting the key attribute; it is used in event sequencing and may lead to performance issues if not selected carefully.

1. Check Save business objects to a file, or check Open the business objects in separate windows. The latter choice launches the Business Object Designer and opens the business objects in that application.
2. Check Shutdown ODA and click Finish.

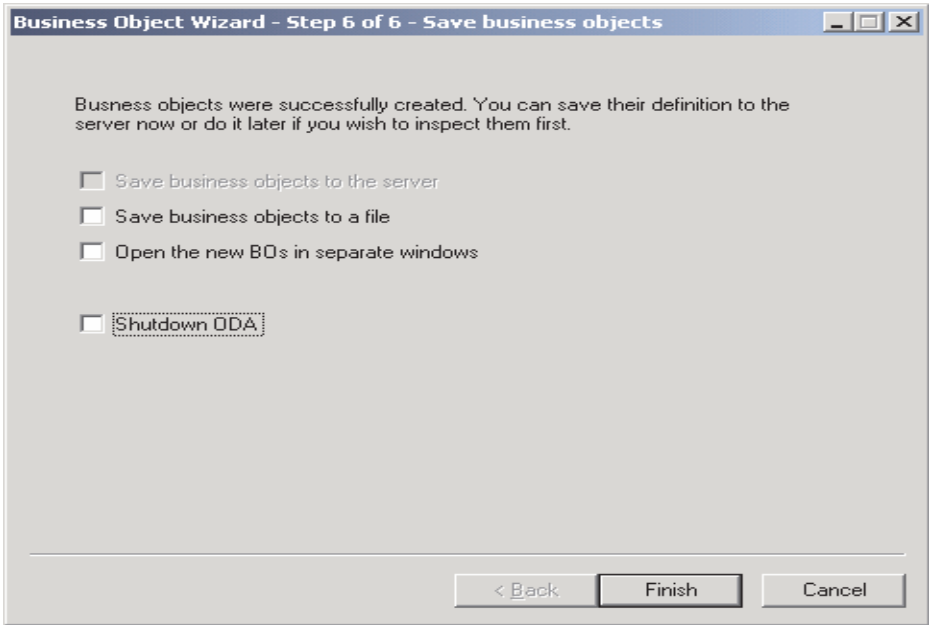


Figure 68. Save window

For request processing, the call to the web service must have a request and, if synchronous, a response and fault messages. For event processing, the collaboration exposed must have a request and, if synchronous, a response and fault messages. The WSDL ODA generates business objects for each of these including the application-specific information (ASI) at every level as well as SOAP data handler, and protocol Config MOs. The SOAP bindings in WSDL document determine the structure of SOAP message. For more on business object structure, see Chapter 3, “Business object requirements,” on page 21.

Limitations

Table 48 describes WSDL ODA support for various combinations of attributes style, use, and part definitions using type and element.

Table 48. WSDL ODA limitations

Style/Use/Parts defined using	Description
rpc/encoded/type	Supported
rpc/encoded/element	Supported
rpc/literal/type	Supported
rpc/literal/element	Supported
doc/encoded/type	Not supported
doc/encoded/element	Not supported
doc/literal/type	Supported
doc/literal/element	Supported

The WSDL ODA can retrieve WSDL files that are completely self-contained (in one file) or are separated into an implementation file containing the service element, an interface file containing all the other WSDL elements including types, messages, portTypes, and bindings, and one or more files for the schemas. The WSDL ODA is not able to successfully retrieve WSDL files that have more than one interface file, for example, with messages and portTypes in one file and bindings in another file.

Schema in the WSDL document must be self-contained in terms of namespace prefixes. You cannot use a namespace prefix that is defined in the <definitions>/<types> element of the WSDL document in the <schema> element that is a child of the <types> element. You need to re-define the namespace prefix on the <schema> element if it is to be used in the sub-elements of the <schema> element. The following is an example of a schema that is not self-contained:

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:NS="NS">
    <types>
         <schema xmlns="http://www.w3.org/1999/XMLSchema">
        <element name="NSElem" type="NS:NSType"/>
         </schema>
   </types>
</definitions>
```

Namespace prefix NS is defined on the <definitions> element and is used without re-definition on the <schema> element. Hence the WSDL ODA will throw an error. To work around this limitation, re-define the namespace prefix NS on the <schema> element as shown below:

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:NS="NS">
    <types>
         <schema xmlns="http://www.w3.org/1999/XMLSchema" xmlns:NS="NS">
        <element name="NSElem" type="NS:NSType"/>
         </schema>
   </types>
</definitions>
```

Chapter 9. Troubleshooting

The chapter describes problems that you may encounter when starting up or running the connector.

Start-up problems

Problem	Potential solution / explanation
Algorithm Not Supported/Algorithm 'SSL' not available	This error occurs when the SSL version specified in the connector configurator is not supported by your JSSE provider. Solution: check JSSE provider's documentation for the supported SSL versions. For IBM JSSE make sure your java.security file in the *ProductDir*/lib/security directory has the following entry `security.provider.<number>=com.ibm.jsse.IBMJSSEProvider` where `<number>` is the preference order for loading the security provider.
Error loading keystore:Keystore file path:"*<path>*" incorrectly specified:KeyStore not found	This error occurs if you specify an incorrect path for the keystore and/or truststore files. Solution: check the keystore file path specified in the SSL->KeyStore property in the Connector configurator. Also, if you are using truststore, check the truststore file path specified in SSL->TrustStore property in the Connector configurator.
KeyManagementError: KeyStore is tampered with, KeyManagement error	This error occurs if your keystore and/or truststore have been tampered with or otherwise corrupted. This error may also occur if you have specified an incorrect value for the password. Solution: ensure that the keystore has not been tampered. Try recreating the keystore. Also make sure you have entered a correct password in the SSL->KeyStorePassword and SSL->TrustStorePassword connector properties.
Error loading certificates from keystore	This error occurs if your certificates and/or keystore, truststore have been tampered with. This error may also occur if you have specified an incorrect value for the password. Solution: check to see if the certificate, keystore or truststore have been tampered with. Also, ensure that you have specified a correct password in the SSL->KeyStorePassword and SSL->TruststorePassword connector properties.
Error creating the server socket, terminating: error	This error occurs if the SOAP/HTTP or SOAP/HTTPS protocol listener cannot bind to the port specified in connector properties. Solution: check the ports specified for all of the SOAP/HTTP and SOAP/HTTPS protocol listeners. If the same port is specified for more than one listener, only one of the listeners can start up. Additionally, check if you have any other service running on that port. If so, then you may want to choose a different port for the protocol listeners.
KeyManagementError:UnrecoverableKeyException, Keys could not be recovered	This error occurs if the keystore or truststore cannot be used. Solution: create a new keystore.

Problem	Potential solution / explanation
SSL Handshake Exception: Unknown CA	This occurs if you do not have a CA certificate in your truststore. Solution: check whether the CA's certificate, as well as its self-signed certificates, reside in the truststore. Also, ensure that the DN of the certificate has the host name (preferably the IP address).
You notice excessive JSSE logging in your log file.	If you do not want to see all of the underlying JSSE details on your console, set the value of SSL->SSLDebug property in the connector configurator to `false`.
You have specified a protocol listener but the listener is not getting initialized; you see the following warning message in the connector: `Skipping Protocol Listener Property Set "SOME_LISTENER_NAME" with protocol property "": unable to determine the protocol listener class.]`	The connector was unable to extract a valid value for the Protocol property of the protocol listener. Valid values are `soap/http`, `soap/https`, or `soap/jms`. Solution: this is not an error condition. However, if you want the connector to use this listener, specify a valid Protocol property value.
You have specified a protocol handler, but it is not getting initialized; you see following warning message in the connector. `Unable to determine the type of the handler; skipping initializing of current handler. Handler property details: Name: <Handler Name>; Value: Name: Protocol; Value: Name: ResponseWaitTimeout; Value: Name: ReplyToQueue; Value: .]`	The connector was unable to extract a valid value for the Protocol property of the handler. Valid values are `soap/http`, `soap/https`, or `soap/jms`. Solution: This is not an error condition. However, if you want connector to use this handler, specify a valid Protocol property value.
`java.lang.NoClassDefFoundError: Javax/jms/JMSException...`	The connector cannot find `jms.jar` Solution: make sure that `jms.jar` is in the connector classpath.
`Fail to lookup, queue: "InProgressQueue" for specified queue name: "<queue name>" queue using JNDI "<queue name>"" javax.naming.NameNotFoundException: <queue name>`	If you are using SOAP/JMS web services with the connector, then this problem occurs when you do not create queues. This error may also occur, if you have set JNDI->LookupQueuesUsingJNDI to `true` and the connector is not able to look up the queues using JNDI. Solution: create the queues required by the connector. If JNDI->LookupQueuesUsingJNDI is set to `true`, make sure queues required by the connector can be looked up using JNDI.
Error in initializing, JNDI Context is not initialized, user can not use JMS protocol	If you have configured the connector to use a SOAP/JMS protocol listener or SOAP/JMS protocol handler, you must specify JNDI properties. Solution: make sure that you have specified required JNDI connector-specific properties. Refer to your JNDI provider documentation to determine the libraries and jar files required to connect to your JNDI provider. Make sure all of the required jar files are in the classpath of the connector. Also, make sure all of the required libraries are in the path of the connector.
Error in getting initial context	If you have configured the connector to use a SOAP/JMS protocol listener or a SOAP/JMS protocol handler, you must specify JNDI properties. This error may also occur if you have not specified JNDI properties correctly. Solution: check the JNDI properties. Make sure your JNDI is configured properly. Refer to your JNDI provider documentation to determine the libraries and jar files required to connect to your JNDI provider. Make sure all of the required jar files are in the classpath of the connector. Also, make sure all of the required libraries are in the path of the connector.

Run-time errors

Problem	Potential solution / explanation
`Error parsing HTTP response:Reached end of stream while reading HTTP response header`	This error occurs when the connector invokes a SOAP/HTTP web service. It occurs because your target web service sent an incorrect HTTP response. Solution: make sure your target SOAP/HTTP web service end point address is correct.
`Error in the url mentioned , unable to extract host and port details ,destination is wrong <destination URL>`	This error occurs when the connector invokes an SOAP/HTTP Web Service. It occurs because you have specified an incorrect end point address for the SOAP/HTTP web service. Solution: make sure you have specified the correct end point address for the web service.
Failure in sending event business object <*BO Name*> with verb <*Verb*> to the broker. Received execution status ″-1″ and error message: `MapException: Unable to find the map to map business objects <BO Name> for the connector controller WebServicesConnector` .	This error occurs when the integration broker fails to process the event because the collaboration to which the connector is sending the event synchronously either does not exist or does not accept the business object verb. Solution: if you are using a web services TLO for event notification, examine the `ws_collab` object-level ASI of the TLO. (The name of the TLO is given in the error message.) Check the value of the `ws_collab` ASI. Make sure this collaboration exists and is running. If `ws_mode` BO level ASI is set to `synch`, `ws_collab` ASI is required. Check the value of `ws_verb` object-level ASI. Make sure the collaboration specified by the `ws_collab` ASI can be triggered by the verb specified in the `ws_verb` ASI. If you are using a non-TLO for event notification, examine the WSCollaborations connector property. Find the collaboration that will be invoked synchronously by this business object. Make sure this collaboration exists and is running.
Failed to transform a soap request into a request business object. Soap Fault: `Failure in generating request object - no verb could be set on the request bo`	This error occurs during event notification when the connector is unable to determine the verb of the business object that the connector is attempting to send to the integration broker. Solution: if you are using a web services TLO for event notification, make sure you have specified `ws_verb` object-level ASI for this TLO. Specify the verb as the value of this ASI. If you are using a non-TLO for event notification, the SOAP message sent by your web service client must contain the verb element. The SOAP data handler sets the verb of the business object using the value of the verb element in the SOAP message.If the web service client does not send the verb in the SOAP message, the SOAP data handler cannot set the verb on the business object. In this case, the connector cannot deliver the business object to the integration broker. If you suspect that your web service clients may not include a verb element in the SOAP message, you may provide a `DefaultVerb` verb-level ASI for this business object. If you do so, the connector sets this verb on the business object before sending it to the integration broker.

Appendix A. Standard configuration properties for connectors

This appendix describes the standard configuration properties for the connector component of WebSphere Business Integration adapters. The information covers connectors running on the following integration brokers:

- WebSphere InterChange Server (ICS)
- WebSphere MQ Integrator, WebSphere MQ Integrator Broker, and WebSphere Business Integration Message Broker, collectively referred to as the WebSphere Message Brokers (WMQI).
- WebSphere Application Server (WAS)

Not every connector makes use of all these standard properties. When you select an integration broker from Connector Configurator, you will see a list of the standard properties that you need to configure for your adapter running with that broker.

For information about properties specific to the connector, see the relevant adapter user guide.

Note: In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes and follow the conventions for each operating system.

New and deleted properties

These standard properties have been added in this release.

New properties

- XMLNameSpaceFormat

Deleted properties

- RestartCount

Configuring standard connector properties

Adapter connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

This section describes the standard configuration properties. For information on configuration properties specific to a connector, see its adapter user guide.

Using Connector Configurator

You configure connector properties from Connector Configurator, which you access from System Manager. For more information on using Connector Configurator, refer to the sections on Connector Configurator in this guide.

Note: Connector Configurator and System Manager run only on the Windows system. If you are running the connector on a UNIX system, you must have a Windows machine with these tools installed. To set connector properties

for a connector that runs on UNIX, you must start up System Manager on the Windows machine, connect to the UNIX integration broker, and bring up Connector Configurator for the connector.

Setting and updating property values

The default length of a property field is 255 characters.

The connector uses the following order to determine a property's value (where the highest number overrides other values):

1. Default
2. Repository (only if WebSphere InterChange Server is the integration broker)
3. Local configuration file
4. Command line

A connector obtains its configuration values at startup. If you change the value of one or more connector properties during a run-time session, the property's **Update Method** determines how the change takes effect. There are four different update methods for standard connector properties:

- **Dynamic**
  The change takes effect immediately after it is saved in System Manager. If the connector is working in stand-alone mode (independently of System Manager), for example with one of the WebSphere message brokers, you can only change properties through the configuration file. In this case, a dynamic update is not possible.
- **Agent restart (ICS only)**
  The change takes effect only after you stop and restart the application-specific component.
- **Component restart**
  The change takes effect only after the connector is stopped and then restarted in System Manager. You do not need to stop and restart the application-specific component or the integration broker.
- **Server restart**
  The change takes effect only after you stop and restart the application-specific component and the integration broker.

To determine how a specific property is updated, refer to the **Update Method** column in the Connector Configurator window, or see the Update Method column in Table 49 on page 171 below.

Summary of standard properties

Table 49 on page 171 provides a quick reference to the standard connector configuration properties. Not all the connectors make use of all these properties, and property settings may differ from integration broker to integration broker, as standard property dependencies are based on `RepositoryDirectory`.

You must set the values of some of these properties before running the connector. See the following section for an explanation of each property.

Note: In the "Notes"column in Table 49 on page 171, the phrase "Repository directory is REMOTE" indicates that the broker is the InterChange Server. When the broker is WMQI or WAS, the repository directory is set to LOCAL

Table 49. Summary of standard configuration properties

Property name	Possible values	Default value	Update method	Notes
AdminInQueue	Valid JMS queue name	*CONNECTORNAME* /ADMININQUEUE	Component restart	Delivery Transport is JMS
AdminOutQueue	Valid JMS queue name	*CONNECTORNAME*/ADMINOUTQUEUE	Component restart	Delivery Transport is JMS
AgentConnections	1-4	1	Component restart	Delivery Transport is MQ or IDL: Repository directory is <REMOTE> (broker is ICS)
AgentTraceLevel	0-5	0	Dynamic	
ApplicationName	Application name	Value specified for the connector application name	Component restart	
BrokerType	ICS, WMQI, WAS		Component restart	
CharacterEncoding	ascii7, ascii8, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437 **Note:** This is a subset of supported values.	ascii7	Component restart	
ConcurrentEventTriggeredFlows	1 to 32,767	1	Component restart	Repository directory is <REMOTE> (broker is ICS)
ContainerManagedEvents	No value or JMS	No value	Component restart	Delivery Transport is JMS
ControllerStoreAndForwardMode	true or false	true	Dynamic	Repository directory is <REMOTE> (broker is ICS)
ControllerTraceLevel	0-5	0	Dynamic	Repository directory is <REMOTE> (broker is ICS)
DeliveryQueue		*CONNECTORNAME*/DELIVERYQUEUE	Component restart	JMS transport only
DeliveryTransport	MQ, IDL, or JMS	JMS	Component restart	If Repository directory is local, then value is JMS only

Table 49. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
DuplicateEventElimination	`true or false`	`false`	Component restart	JMS transport only: Container Managed Events must be <NONE>
FaultQueue		*CONNECTORNAME*/FAULTQUEUE	Component restart	JMS transport only
jms.FactoryClassName	`CxCommon.Messaging.jms .IBMMQSeriesFactory` or `CxCommon.Messaging .jms.SonicMQFactory` or any Java class name	`CxCommon.Messaging. jms.IBMMQSeriesFactory`	Component restart	JMS transport only
jms.MessageBrokerName	If FactoryClassName is `IBM`, use `crossworlds.queue. manager`. If FactoryClassName is `Sonic`, use `localhost:2506`.	`crossworlds.queue.manager`	Component restart	JMS transport only
jms.NumConcurrentRequests	Positive integer	`10`	Component restart	JMS transport only
jms.Password	Any valid password		Component restart	JMS transport only
jms.UserName	Any valid name		Component restart	JMS transport only
JvmMaxHeapSize	Heap size in megabytes	`128m`	Component restart	Repository directory is <REMOTE> (broker is ICS)
JvmMaxNativeStackSize	Size of stack in kilobytes	`128k`	Component restart	Repository directory is <REMOTE> (broker is ICS)
JvmMinHeapSize	Heap size in megabytes	`1m`	Component restart	Repository directory is <REMOTE> (broker is ICS)
ListenerConcurrency	`1- 100`	`1`	Component restart	Delivery Transport must be `MQ`
Locale	`en_US`, `ja_JP`, `ko_KR`, `zh_CN`, `zh_TW`, `fr_FR`, `de_DE`, `it_IT`, `es_ES`, `pt_BR` **Note:** This is a subset of the supported locales.	`en_US`	Component restart	

Table 49. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
LogAtInterchangeEnd	`true` or `false`	`false`	Component restart	Repository Directory must be <REMOTE> (broker is ICS)
MaxEventCapacity	`1-2147483647`	`2147483647`	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
MessageFileName	Path or filename	`CONNECTORNAMEConnector.txt`	Component restart	
MonitorQueue	Any valid queue name	*`CONNECTORNAME`*`/MONITORQUEUE`	Component restart	JMS transport only: DuplicateEvent Elimination must be true
OADAutoRestartAgent	`true` or `false`	`false`	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
OADMaxNumRetry	A positive number	`1000`	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
OADRetryTimeInterval	A positive number in minutes	`10`	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
PollEndTime	`HH:MM`	`HH:MM`	Component restart	
PollFrequency	A positive integer in milliseconds `no` (to disable polling) `key` (to poll only when the letter `p` is entered in the connector's Command Prompt window)	`10000`	Dynamic	
PollQuantity	`1-500`	`1`	Agent restart	JMS transport only: Container Managed Events is specified
PollStartTime	`HH:MM`(HH is 0-23, MM is 0-59)	`HH:MM`	Component restart	

Table 49. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
RepositoryDirectory	Location of metadata repository		Agent restart	For ICS: set to <REMOTE> For WebSphere MQ message brokers and WAS: set to C:\crossworlds\ repository
RequestQueue	Valid JMS queue name	*CONNECTORNAME*/REQUESTQUEUE	Component restart	Delivery Transport is JMS
ResponseQueue	Valid JMS queue name	*CONNECTORNAME*/RESPONSEQUEUE	Component restart	Delivery Transport is JMS: required only if Repository directory is <REMOTE>
RestartRetryCount	0-99	3	Dynamic	
RestartRetryInterval	A sensible positive value in minutes: 1 - 2147483547	1	Dynamic	
RHF2MessageDomain	mrm, xml	mrm	Component restart	Only if Delivery Transport is JMS and WireFormat is CwXML.
SourceQueue	Valid WebSphere MQ name	*CONNECTORNAME*/SOURCEQUEUE	Agent restart	Only if Delivery Transport is JMS and Container Managed Events is specified
SynchronousRequestQueue		*CONNECTORNAME*/ SYNCHRONOUSREQUESTQUEUE	Component restart	Delivery Transport is JMS
SynchronousRequestTimeout	0 - any number (millisecs)	0	Component restart	Delivery Transport is JMS
SynchronousResponseQueue		*CONNECTORNAME*/ SYNCHRONOUSRESPONSEQUEUE	Component restart	Delivery Transport is JMS
WireFormat	CwXML, CwBO	CwXML	Agent restart	CwXML if Repository Directory is not <REMOTE>: CwBO if Repository Directory is <REMOTE>
WsifSynchronousRequestTimeout	0 - any number (millisecs)	0	Component restart	WAS only
XMLNameSpaceFormat	short, long	short	Agent restart	WebSphere MQ message brokers and WAS only

Standard configuration properties

This section lists and defines each of the standard connector configuration properties.

AdminInQueue

The queue that is used by the integration broker to send administrative messages to the connector.

The default value is `CONNECTORNAME/ADMININQUEUE`.

AdminOutQueue

The queue that is used by the connector to send administrative messages to the integration broker.

The default value is `CONNECTORNAME/ADMINOUTQUEUE`.

AgentConnections

Applicable only if `RepositoryDirectory` is <REMOTE>.

The AgentConnections property controls the number of ORB (Object Request Broker) connections opened by `orb.init[]`.

The default value of this property is set to 1. You can change it as required.

AgentTraceLevel

Level of trace messages for the application-specific component. The default is `0`. The connector delivers all trace messages applicable at the tracing level set or lower.

ApplicationName

Name that uniquely identifies the connector's application. This name is used by the system administrator to monitor the WebSphere business integration system environment. This property must have a value before you can run the connector.

BrokerType

Identifies the integration broker type that you are using. The options are ICS, WebSphere message brokers (WMQI, WMQIB or WBIMB) or WAS.

CharacterEncoding

Specifies the character code set used to map from a character (such as a letter of the alphabet, a numeric representation, or a punctuation mark) to a numeric value.

Note: Java-based connectors do not use this property. A C++ connector currently uses the value `ascii7` for this property.

By default, a subset of supported character encodings only is displayed in the drop-down list. To add other supported values to the drop-down list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory. For more information, see the sections on Connector Configurator in this guide.

ConcurrentEventTriggeredFlows

Applicable only if RepositoryDirectory is <REMOTE>.

Determines how many business objects can be concurrently processed by the connector for event delivery. Set the value of this attribute to the number of business objects you want concurrently mapped and delivered. For example, set the value of this property to 5 to cause five business objects to be concurrently processed. The default value is 1.

Setting this property to a value greater than 1 allows a connector for a source application to map multiple event business objects at the same time and deliver them to multiple collaboration instances simultaneously. This speeds delivery of business objects to the integration broker, particularly if the business objects use complex maps. Increasing the arrival rate of business objects to collaborations can improve overall performance in the system.

To implement concurrent processing for an entire flow (from a source application to a destination application), you must:

- Configure the collaboration to use multiple threads by setting its Maximum number of concurrent events property high enough to use multiple threads.
- Ensure that the destination application's application-specific component can process requests concurrently. That is, it must be multi-threaded, or be able to use connector agent parallelism and be configured for multiple processes. Set the Parallel Process Degree configuration property to a value greater than 1.

The ConcurrentEventTriggeredFlows property has no effect on connector polling, which is single-threaded and performed serially.

ContainerManagedEvents

This property allows a JMS-enabled connector with a JMS event store to provide guaranteed event delivery, in which an event is removed from the source queue and placed on the destination queue as a single JMS transaction.

There is no default value.

When ContainerManagedEvents is set to JMS, you must configure the following properties to enable guaranteed event delivery:

- PollQuantity = 1 to 500
- SourceQueue = /SOURCEQUEUE

You must also configure a data handler with the MimeType, DHClass (data handler class), and DataHandlerConfigMOName (the meta-object name, which is optional) properties. To set those values, use the **Data Handler** tab in Connector Configurator.

Thes properties are adapter-specific, but **example** values are:

- MimeType = text\xml
- DHClass = com.crossworlds.DataHandlers.text.xml
- DataHandlerConfigMOName = MO_DataHandler_Default

The fields for these values in the Data Handler tab will be displayed only if you have set ContainerManagedEvents to JMS.

Note: When ContainerManagedEvents is set to JMS, the connector does *not* call its pollForEvents() method, thereby disabling that method's functionality.

This property only appears if the DeliveryTransport property is set to the value JMS.

ControllerStoreAndForwardMode

Applicable only if RepositoryDirectory is <REMOTE>.

Sets the behavior of the connector controller after it detects that the destination application-specific component is unavailable.

If this property is set to true and the destination application-specific component is unavailable when an event reaches ICS, the connector controller blocks the request to the application-specific component. When the application-specific component becomes operational, the controller forwards the request to it.

However, if the destination application's application-specific component becomes unavailable **after** the connector controller forwards a service call request to it, the connector controller fails the request.

If this property is set to false, the connector controller begins failing all service call requests as soon as it detects that the destination application-specific component is unavailable.

The default is true.

ControllerTraceLevel

Applicable only if RepositoryDirectory is <REMOTE>.

Level of trace messages for the connector controller. The default is 0.

DeliveryQueue

Applicable only if DeliveryTransport is JMS.

The queue that is used by the connector to send business objects to the integration broker.

The default value is CONNECTORNAME/DELIVERYQUEUE.

DeliveryTransport

Specifies the transport mechanism for the delivery of events. Possible values are MQ for WebSphere MQ, IDL for CORBA IIOP, or JMS for Java Messaging Service.

- If the RepositoryDirectory is remote, the value of the DeliveryTransport property can be MQ, IDL, or JMS, and the default is IDL.
- If the RepositoryDirectory is a local directory, the value may only be JMS.

The connector sends service call requests and administrative messages over CORBA IIOP if the value configured for the DeliveryTransport property is MQ or IDL.

WebSphere MQ and IDL

Use WebSphere MQ rather than IDL for event delivery transport, unless you must have only one product. WebSphere MQ offers the following advantages over IDL:

- Asynchronous communication:
  WebSphere MQ allows the application-specific component to poll and persistently store events even when the server is not available.
- Server side performance:
  WebSphere MQ provides faster performance on the server side. In optimized mode, WebSphere MQ stores only the pointer to an event in the repository database, while the actual event remains in the WebSphere MQ queue. This saves having to write potentially large events to the repository database.
- Agent side performance:
  WebSphere MQ provides faster performance on the application-specific component side. Using WebSphere MQ, the connector's polling thread picks up an event, places it in the connector's queue, then picks up the next event. This is faster than IDL, which requires the connector's polling thread to pick up an event, go over the network into the server process, store the event persistently in the repository database, then pick up the next event.

JMS

Enables communication between the connector and client connector framework using Java Messaging Service (JMS).

If you select JMS as the delivery transport, additional JMS properties such as `jms.MessageBrokerName`, `jms.FactoryClassName`, `jms.Password`, and `jms.UserName`, appear in Connector Configurator. The first two of these properties are required for this transport.

Important: There may be a memory limitation if you use the JMS transport mechanism for a connector in the following environment:

- AIX 5.0
- WebSphere MQ 5.3.0.1
- When ICS is the integration broker

In this environment, you may experience difficulty starting both the connector controller (on the server side) and the connector (on the client side) due to memory use within the WebSphere MQ client. If your installation uses less than 768M of process heap size, IBM recommends that you set:

- The `LDR_CNTRL` environment variable in the `CWSharedEnv.sh` script.

  This script resides in the `\bin` directory below the product directory. With a text editor, add the following line as the first line in the `CWSharedEnv.sh` script:

  ```
  export LDR_CNTRL=MAXDATA=0x30000000
  ```

  This line restricts heap memory usage to a maximum of 768 MB (3 segments * 256 MB). If the process memory grows more than this limit, page swapping can occur, which can adversely affect the performance of your system.
- The `IPCCBaseAddress` property to a value of `11` or `12`. For more information on this property, see the *System Installation Guide for UNIX*.

DuplicateEventElimination

When you set this property to `true`, a JMS-enabled connector can ensure that duplicate events are not delivered to the delivery queue. To use this feature, the connector must have a unique event identifier set as the business object's **ObjectEventId** attribute in the application-specific code. This is done during connector development.

This property can also be set to `false`.

Note: When DuplicateEventElimination is set to true, you must also configure the MonitorQueue property to enable guaranteed event delivery.

FaultQueue

If the connector experiences an error while processing a message then the connector moves the message to the queue specified in this property, along with a status indicator and a description of the problem.

The default value is CONNECTORNAME/FAULTQUEUE.

JvmMaxHeapSize

The maximum heap size for the agent (in megabytes). This property is applicable only if the RepositoryDirectory value is <REMOTE>.

The default value is 128m.

JvmMaxNativeStackSize

The maximum native stack size for the agent (in kilobytes). This property is applicable only if the RepositoryDirectory value is <REMOTE>.

The default value is 128k.

JvmMinHeapSize

The minimum heap size for the agent (in megabytes). This property is applicable only if the RepositoryDirectory value is <REMOTE>.

The default value is 1m.

jms.FactoryClassName

Specifies the class name to instantiate for a JMS provider. You *must* set this connector property when you choose JMS as your delivery transport mechanism (DeliveryTransport).

The default is CxCommon.Messaging.jms.IBMMQSeriesFactory.

jms.MessageBrokerName

Specifies the broker name to use for the JMS provider. You *must* set this connector property when you choose JMS as your delivery transport mechanism (DeliveryTransport).

The default is crossworlds.queue.manager. Use the default when connecting to a local message broker.

When you connect to a remote message broker, this property takes the following (mandatory) values:
QueueMgrName:<Channel>:<HostName>:<PortNumber>,
where the variables are:
QueueMgrName: The name of the queue manager.
Channel: The channel used by the client.
HostName: The name of the machine where the queue manager is to reside.
PortNumber: The port number to be used by the queue manager for listening.

For example:
`jms.MessageBrokerName = WBIMB.Queue.Manager:CHANNEL1:RemoteMachine:1456`

jms.NumConcurrentRequests

Specifies the maximum number of concurrent service call requests that can be sent to a connector at the same time. Once that maximum is reached, new service calls block and wait for another request to complete before proceeding.

The default value is 10.

jms.Password

Specifies the password for the JMS provider. A value for this property is optional.

There is no default.

jms.UserName

Specifies the user name for the JMS provider. A value for this property is optional.

There is no default.

ListenerConcurrency

This property supports multi-threading in MQ Listener when ICS is the integration broker. It enables batch writing of multiple events to the database, thus improving system performance. The default value is 1.

This property applies only to connectors using MQ transport. The `DeliveryTransport` property must be set to `MQ`.

Locale

Specifies the language code, country or territory, and, optionally, the associated character code set. The value of this property determines such cultural conventions as collation and sort order of data, date and time formats, and the symbols used in monetary specifications.

A locale name has the following format:
ll_TT.codeset

where:

ll	a two-character language code (usually in lower case)
TT	a two-letter country or territory code (usually in upper case)
codeset	the name of the associated character code set; this portion of the name is often optional.

By default, only a subset of supported locales appears in the drop-down list. To add other supported values to the drop-down list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory. For more information, refer to the sections on Connector Configurator in this guide.

The default value is en_US. If the connector has not been globalized, the only valid value for this property is en_US. To determine whether a specific connector has been globalized, see the connector version list on these websites:

http://www.ibm.com/software/websphere/wbiadapters/infocenter, or
http://www.ibm.com/websphere/integration/wicserver/infocenter

LogAtInterchangeEnd

Applicable only if RespositoryDirectory is <REMOTE>.

Specifies whether to log errors to the integration broker's log destination. Logging to the broker's log destination also turns on e-mail notification, which generates e-mail messages for the MESSAGE_RECIPIENT specified in the InterchangeSystem.cfg file when errors or fatal errors occur.

For example, when a connector loses its connection to its application, if LogAtInterChangeEnd is set to true, an e-mail message is sent to the specified message recipient. The default is false.

MaxEventCapacity

The maximum number of events in the controller buffer. This property is used by flow control and is applicable only if the value of the RepositoryDirectory property is <REMOTE>.

The value can be a positive integer between 1 and 2147483647. The default value is 2147483647.

MessageFileName

The name of the connector message file. The standard location for the message file is \connectors\messages in the product directory. Specify the message filename in an absolute path if the message file is not located in the standard location.

If a connector message file does not exist, the connector uses InterchangeSystem.txt as the message file. This file is located in the product directory.

Note: To determine whether a specific connector has its own message file, see the individual adapter user guide.

MonitorQueue

The logical queue that the connector uses to monitor duplicate events. It is used only if the DeliveryTransport property value is JMS and DuplicateEventElimination is set to TRUE.

The default value is CONNECTORNAME/MONITORQUEUE

OADAutoRestartAgent

Valid only when the RepositoryDirectory is <REMOTE>.

Specifies whether the connector uses the automatic and remote restart feature. This feature uses the MQ-triggered Object Activation Daemon (OAD) to restart the connector after an abnormal shutdown, or to start a remote connector from System Monitor.

This property must be set to `true` to enable the automatic and remote restart feature. For information on how to configure the MQ-triggered OAD feature. see the *Installation Guide for Windows* or *for UNIX*.

The default value is `false`.

OADMaxNumRetry

Valid only when the `RepositoryDirectory` is <REMOTE>.

Specifies the maximum number of times that the MQ-triggered OAD automatically attempts to restart the connector after an abnormal shutdown. The `OADAutoRestartAgent` property must be set to `true` for this property to take effect.

The default value is `1000`.

OADRetryTimeInterval

Valid only when the `RepositoryDirectory` is <REMOTE>.

Specifies the number of minutes in the retry-time interval for the MQ-triggered OAD. If the connector agent does not restart within this retry-time interval, the connector controller asks the OAD to restart the connector agent again. The OAD repeats this retry process as many times as specified by the `OADMaxNumRetry` property. The `OADAutoRestartAgent` property must be set to `true` for this property to take effect.

The default is `10`.

PollEndTime

Time to stop polling the event queue. The format is `HH:MM`, where *HH* represents 0-23 hours, and *MM* represents 0-59 seconds.

You must provide a valid value for this property. The default value is `HH:MM`, but must be changed.

PollFrequency

This is the interval between the end of the last poll and the start of the next poll. `PollFrequency` specifies the amount of time (in milliseconds) between the end of one polling action, and the start of the next polling action. This is not the interval between polling actions. Rather, the logic is as follows:

- Poll to obtain the number of objects specified by the value of `PollQuantity`.
- Process these objects. For some adapters, this may be partly done on separate threads, which execute asynchronously to the next polling action.
- Delay for the interval specified by `PollFrequency`.
- Repeat the cycle.

Set `PollFrequency` to one of the following values:

- The number of milliseconds between polling actions (an integer).
- The word `key`, which causes the connector to poll only when you type the letter `p` in the connector's Command Prompt window. Enter the word in lowercase.
- The word `no`, which causes the connector not to poll. Enter the word in lowercase.

The default is 10000.

Important: Some connectors have restrictions on the use of this property. Where they exist, these restrictions are documented in the chapter on installing and configuring the adapter.

PollQuantity

Designates the number of items from the application that the connector should poll for. If the adapter has a connector-specific property for setting the poll quantity, the value set in the connector-specific property will override the standard property value.

FIX

An email message is also considerd an event. The connector behaves as follows when it is polled for email.

Polled once - connector goes to pick 1. the body of the message as it is also considered an attachment also. Since no DH was specified for this mime type, it it will ignore the body. 2. conector process first PO attachment. DH is avaiable for this mime type so it sends the business object to the Visual Test Connector. If the 3. accept in VTC again no BO should come thru Polled second time 1. conector process second PO attachment. DH is avaiable for this mime type so it sends teh BO to VTC2. accept in VTC again now the third PO attachment should come through. This is the correct behaviour.

PollStartTime

The time to start polling the event queue. The format is *HH:MM*, where *HH* represents 0-23 hours, and *MM* represents 0-59 seconds.

You must provide a valid value for this property. The default value is HH:MM, but must be changed.

RequestQueue

The queue that is used by the integration broker to send business objects to the connector.

The default value is CONNECTOR/REQUESTQUEUE.

RepositoryDirectory

The location of the repository from which the connector reads the XML schema documents that store the meta-data for business object definitions.

When the integration broker is ICS, this value must be set to <REMOTE> because the connector obtains this information from the InterChange Server repository.

When the integration broker is a WebSphere message broker or WAS, this value must be set to <*local directory*>.

ResponseQueue

Applicable only if DeliveryTransport is JMS and required only if RepositoryDirectory is <REMOTE>.

Designates the JMS response queue, which delivers a response message from the connector framework to the integration broker. When the integration broker is ICS, the server sends the request and waits for a response message in the JMS response queue.

RestartRetryCount

Specifies the number of times the connector attempts to restart itself. When used for a parallel connector, specifies the number of times the master connector application-specific component attempts to restart the slave connector application-specific component.

The default is 3.

RestartRetryInterval

Specifies the interval in minutes at which the connector attempts to restart itself. When used for a parallel connector, specifies the interval at which the master connector application-specific component attempts to restart the slave connector application-specific component. Possible values ranges from 1 to 2147483647.

The default is 1.

RHF2MessageDomain

WebSphere message brokers and WAS only.

This property allows you to configure the value of the field domain name in the JMS header. When data is sent to WMQI over JMS transport, the adapter framework writes JMS header information, with a domain name and a fixed value of `mrm`. A connfigurable domain name enables users to track how the WMQI broker processes the message data.

A sample header would look like this:

```
<mcd><Msd>mrm</Msd><Set>3</Set><Type>
Retek_POPhyDesc</Type><Fmt>CwXML</Fmt></mcd>
```

The default value is `mrm`, but it may also be set to `xml`. This property only appears when DeliveryTransport is set to `JMS`and WireFormat is set to `CwXML`.

SourceQueue

Applicable only if `DeliveryTransport` is JMS and ContainerManagedEvents is specified.

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see “ContainerManagedEvents” on page 176.

The default value is `CONNECTOR/SOURCEQUEUE`.

SynchronousRequestQueue

Applicable only if `DeliveryTransport` is JMS.

Delivers request messages that require a synchronous response from the connector framework to the broker. This queue is necessary only if the connector uses synchronous execution. With synchronous execution, the connector framework

sends a message to the SynchronousRequestQueue and waits for a response back from the broker on the SynchronousResponseQueue. The response message sent to the connector bears a correlation ID that matches the ID of the original message.

The default is CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE

SynchronousResponseQueue

Applicable only if DeliveryTransport is JMS.

Delivers response messages sent in reply to a synchronous request from the broker to the connector framework. This queue is necessary only if the connector uses synchronous execution.

The default is CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE

SynchronousRequestTimeout

Applicable only if DeliveryTransport is JMS.

Specifies the time in minutes that the connector waits for a response to a synchronous request. If the response is not received within the specified time, then the connector moves the original synchronous request message into the fault queue along with an error message.

The default value is 0.

WireFormat

Message format on the transport.

- If the RepositoryDirectory is a local directory, the setting is CwXML.
- If the value of RepositoryDirectory is <REMOTE>, the setting isCwBO.

WsifSynchronousRequestTimeout

WAS integration broker only.

Specifies the time in minutes that the connector waits for a response to a synchronous request. If the response is not received within the specified, time then the connector moves the original synchronous request message into the fault queue along with an error message.

The default value is 0.

XMLNameSpaceFormat

WebSphere message brokers and WAS integration broker only.

A strong property that allows the user to specify short and long name spaces in the XML format of business object definitions.

The default value is short.

Appendix B. Connector Configurator

This appendix describes how to use Connector Configurator to set configuration property values for your adapter.

You use Connector Configurator to:

- Create a connector-specific property template for configuring your connector
- Create a configuration file
- Set properties in a configuration file

Note:

In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes and follow the conventions for each operating system.

The topics covered in this appendix are:

Overview of Connector Configurator

Connector Configurator allows you to configure the connector component of your adapter for use with these integration brokers:

- WebSphere InterChange Server (ICS)
- WebSphere MQ Integrator, WebSphere MQ Integrator Broker, and WebSphere Business Integration Message Broker, collectively referred to as the WebSphere Message Brokers (WMQI)
- WebSphere Application Server (WAS)

You use Connector Configurator to:

- Create a **connector-specific property template** for configuring your connector.
- Create a **connector configuration file**; you must create one configuration file for each connector you install.
- Set properties in a configuration file.
  You may need to modify the default values that are set for properties in the connector templates. You must also designate supported business object definitions and, with ICS, maps for use with collaborations as well as specify messaging, logging and tracing, and data handler parameters, as required.

The mode in which you run Connector Configurator, and the configuration file type you use, may differ according to which integration broker you are running. For example, if WMQI is your broker, you run Connector Configurator directly, and not from within System Manager (see "Running Configurator in stand-alone mode" on page 188).

Connector configuration properties include both standard configuration properties (the properties that all connectors have) and connector-specific properties (properties that are needed by the connector for a specific application or technology).

Because **standard properties** are used by all connectors, you do not need to define those properties from scratch; Connector Configurator incorporates them into your configuration file as soon as you create the file. However, you do need to set the value of each standard property in Connector Configurator.

The range of standard properties may not be the same for all brokers and all configurations. Some properties are available only if other properties are given a specific value. The Standard Properties window in Connector Configurator will show the properties available for your particular configuration.

For **connector-specific properties**, however, you need first to define the properties and then set their values. You do this by creating a connector-specific property template for your particular adapter. There may already be a template set up in your system, in which case, you simply use that. If not, follow the steps in "Creating a new template" on page 189 to set up a new one.

Note: Connector Configurator runs only in a Windows environment. If you are running the connector in a UNIX environment, use Connector Configurator in Windows to modify the configuration file and then copy the file to your UNIX environment.

Starting Connector Configurator

You can start and run Connector Configurator in either of two modes:

- Independently, in stand-alone mode
- From System Manager

Running Configurator in stand-alone mode

You can run Connector Configurator independently and work with connector configuration files, irrespective of your broker.

To do so:

- From **Start>Programs**, click **IBM WebSphere InterChange Server>IBM WebSphere Business Integration Tools>Connector Configurator**.
- Select **File>New>Connector Configuration**.
- When you click the pull-down menu next to **System Connectivity Integration Broker**, you can select ICS, WebSphere Message Brokers or WAS, depending on your broker.

You may choose to run Connector Configurator independently to generate the file, and then connect to System Manager to save it in a System Manager project (see "Completing a configuration file" on page 193.)

Running Configurator from System Manager

You can run Connector Configurator from System Manager.

To run Connector Configurator:

1. Open the System Manager.

2. In the System Manager window, expand the **Integration Component Libraries** icon and highlight **Connectors**.
3. From the System Manager menu bar, click **Tools>Connector Configurator**. The Connector Configurator window opens and displays a **New Connector** dialog box.
4. When you click the pull-down menu next to **System Connectivity Integration Broker**, you can select ICS, WebSphere Message Brokers or WAS, depending on your broker.

To edit an existing configuration file:

- In the System Manager window, select any of the configuration files listed in the Connector folder and right-click on it. Connector Configurator opens and displays the configuration file with the integration broker type and file name at the top.
- From Connector Configurator, select **File>Open**. Select the name of the connector configuration file from a project or from the directory in which it is stored.
- Click the Standard Properties tab to see which properties are included in this configuration file.

Creating a connector-specific property template

To create a configuration file for your connector, you need a connector-specific property template as well as the system-supplied standard properties.

You can create a brand-new template for the connector-specific properties of your connector, or you can use an existing connector definition as the template.

- To create a new template, see "Creating a new template" on page 189.
- To use an existing file, simply modify an existing template and save it under the new name. You can find existing templates in your `\WebSphereAdapters\bin\Data\App` directory.

Creating a new template

This section describes how you create properties in the template, define general characteristics and values for those properties, and specify any dependencies between the properties. Then you save the template and use it as the base for creating a new connector configuration file.

To create a template in Connector Configurator:

1. Click **File>New>Connector-Specific Property Template**.
2. The **Connector-Specific Property Template** dialog box appears.
   - Enter a name for the new template in the **Name** field below **Input a New Template Name.** You will see this name again when you open the dialog box for creating a new configuration file from a template.
   - To see the connector-specific property definitions in any template, select that template's name in the **Template Name** display. A list of the property definitions contained in that template appears in the **Template Preview** display.
3. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template. If you do not see any template that displays the connector-specific properties used by your connector, you will need to create one.

- If you are planning to modify an existing template, select the name of the template from the list in the **Template Name** table below **Select the Existing Template to Modify: Find Template.**
- This table displays the names of all currently available templates. You can also search for a template.

Specifying general characteristics

When you click **Next** to select a template, the **Properties - Connector-Specific Property Template** dialog box appears. The dialog box has tabs for General characteristics of the defined properties and for Value restrictions. The General display has the following fields:

- **General:**
  Property Type
  Updated Method
  Description
- **Flags**
  Standard flags
- **Custom Flag**
  Flag

After you have made selections for the general characteristics of the property, click the **Value** tab.

Specifying values

The **Value** tab enables you to set the maximum length, the maximum multiple values, a default value, or a value range for the property. It also allows editable values. To do so:

1. Click the **Value** tab. The display panel for Value replaces the display panel for General.
2. Select the name of the property in the **Edit properties** display.
3. In the fields for **Max Length** and **Max Multiple Values**, enter your values.

To create a new property value:

1. Select the property in the **Edit properties** list and right-click on it.
2. From the dialog box, select **Add**.
3. Enter the name of the new property value and click OK. The value appears in the **Value** panel on the right.

The **Value** panel displays a table with three columns:

The **Value** column shows the value that you entered in the **Property Value** dialog box, and any previous values that you created.

The **Default Value** column allows you to designate any of the values as the default.

The **Value Range** shows the range that you entered in the **Property Value** dialog box.

After a value has been created and appears in the grid, it can be edited from within the table display.

To make a change in an existing value in the table, select an entire row by clicking on the row number. Then right-click in the **Value** field and click **Edit Value**.

Setting dependencies

When you have made your changes to the **General** and **Value** tabs, click **Next**. The **Dependencies - Connector-Specific Property Template** dialog box appears.

A dependent property is a property that is included in the template and used in the configuration file *only if* the value of another property meets a specific condition. For example, `PollQuantity` appears in the template only if JMS is the transport mechanism and `DuplicateEventElimination` is set to `True`.
To designate a property as dependent and to set the condition upon which it depends, do this:

1. In the **Available Properties** display, select the property that will be made dependent.
2. In the **Select Property** field, use the drop-down menu to select the property that will hold the conditional value.
3. In the **Condition Operator** field, select one of the following:

   == (equal to)

   != (not equal to)

   > (greater than)

   < (less than)

   >= (greater than or equal to)

   <=(less than or equal to)
4. In the **Conditional Value** field, enter the value that is required in order for the dependent property to be included in the template.
5. With the dependent property highlighted in the **Available Properties** display, click an arrow to move it to the **Dependent Property** display.
6. Click **Finish**. Connector Configurator stores the information you have entered as an XML document, under `\data\app` in the`\bin` directory where you have installed Connector Configurator.

Creating a new configuration file

When you create a new configuration file, you must name it and select an integration broker.

- In the System Manager window, right-click on the **Connectors** folder and select **Create New Connector**. Connector Configurator opens and displays the **New Connector** dialog box.
- In stand-alone mode: from Connector Configurator, select **File>New>Connector Configuration**. In the New Connector window, enter the name of the new connector.

You also need to select an integration broker. The broker you select determines the properties that will appear in the configuration file. To select a broker:

- In the **Integration Broker** field, select ICS, WebSphere Message Brokers or WAS connectivity.
- drop-downte the remaining fields in the **New Connector** window, as described later in this chapter.

Creating a configuration file from a connector-specific template

Once a connector-specific template has been created, you can use it to create a configuration file:

1. Click **File>New>Connector Configuration**.
2. The **New Connector** dialog box appears, with the following fields:
   - **Name**

     Enter the name of the connector. Names are case-sensitive. The name you enter must be unique, and must be consistent with the file name for a connector that is installed on the system.

     Important: Connector Configurator does not check the spelling of the name that you enter. You must ensure that the name is correct.
   - **System Connectivity**

     Click ICS or WebSphere Message Brokers or WAS.
   - **Select Connector-Specific Property Template**

     Type the name of the template that has been designed for your connector. The available templates are shown in the **Template Name** display. When you select a name in the Template Name display, the **Property Template Preview** display shows the connector-specific properties that have been defined in that template.

     Select the template you want to use and click **OK**.
3. A configuration screen appears for the connector that you are configuring. The title bar shows the integration broker and connector name. You can fill in all the field values to drop-downte the definition now, or you can save the file and complete the fields later.
4. To save the file, click **File>Save>To File** or **File>Save>To Project**. To save to a project, System Manager must be running.
   If you save as a file, the **Save File Connector** dialog box appears. Choose `*.cfg` as the file type, verify in the File Name field that the name is spelled correctly and has the correct case, navigate to the directory where you want to locate the file, and click **Save**. The status display in the message panel of Connector Configurator indicates that the configuration file was successfully created.

   Important: The directory path and name that you establish here must match the connector configuration file path and name that you supply in the startup file for the connector.
5. To complete the connector definition, enter values in the fields for each of the tabs of the Connector Configurator window, as described later in this chapter.

Using an existing file

You may have an existing file available in one or more of the following formats:

- A connector definition file.
  This is a text file that lists properties and applicable default values for a specific connector. Some connectors include such a file in a `\repository` directory in their delivery package (the file typically has the extension `.txt`; for example, CN_XML.txt for the XML connector).
- An ICS repository file.
  Definitions used in a previous ICS implementation of the connector may be available to you in a repository file that was used in the configuration of that connector. Such a file typically has the extension `.in` or `.out`.
- A previous configuration file for the connector.
  Such a file typically has the extension *.cfg.

Although any of these file sources may contain most or all of the connector-specific properties for your connector, the connector configuration file will not be complete until you have opened the file and set properties, as described later in this chapter.

To use an existing file to configure a connector, you must open the file in Connector Configurator, revise the configuration, and then resave the file.

Follow these steps to open a *.txt, *.cfg, or *.in file from a directory:

1. In Connector Configurator, click **File>Open>From File**.
2. In the **Open File Connector** dialog box, select one of the following file types to see the available files:
   - Configuration (`*.cfg`)
   - ICS Repository (`*.in`, `*.out`)

     Choose this option if a repository file was used to configure the connector in an ICS environment. A repository file may include multiple connector definitions, all of which will appear when you open the file.
   - All files (*.*)

     Choose this option if a `*.txt` file was delivered in the adapter package for the connector, or if a definition file is available under another extension.
3. In the directory display, navigate to the appropriate connector definition file, select it, and click **Open**.

Follow these steps to open a connector configuration from a System Manager project:

1. Start System Manager. A configuration can be opened from or saved to System Manager only if System Manager has been started.
2. Start Connector Configurator.
3. Click **File>Open>From Project**.

Completing a configuration file

When you open a configuration file or a connector from a project, the Connector Configurator window displays the configuration screen, with the current attributes and values.

The title of the configuration screen displays the integration broker and connector name as specified in the file. Make sure you have the correct broker. If not, change the broker value before you configure the connector. To do so:

1. Under the **Standard Properties** tab, select the value field for the `BrokerType` property. In the drop-down menu, select the value `ICS`, `WMQI`, or `WAS`.
2. The Standard Properties tab will display the properties associated with the selected broker. You can save the file now or complete the remaining configuration fields, as described in “Specifying supported business object definitions” on page 196..
3. When you have finished your configuration, click **File>Save>To Project** or **File>Save>To File**.

   If you are saving to file, select `*.cfg` as the extension, select the correct location for the file and click **Save**.

   If multiple connector configurations are open, click **Save All to File** to save all of the configurations to file, or click **Save All to Project** to save all connector configurations to a System Manager project.

Before it saves the file, Connector Configurator checks that values have been set for all required standard properties. If a required standard property is missing a value, Connector Configurator displays a message that the validation failed. You must supply a value for the property in order to save the configuration file.

Setting the configuration file properties

When you create and name a new connector configuration file, or when you open an existing connector configuration file, Connector Configurator displays a configuration screen with tabs for the categories of required configuration values.

Connector Configurator requires values for properties in these categories for connectors running on all brokers:

- Standard Properties
- Connector-specific Properties
- Supported Business Objects
- Trace/Log File values
- Data Handler (applicable for connectors that use JMS messaging with guaranteed event delivery)

Note: For connectors that use JMS messaging, an additional category may display, for configuration of data handlers that convert the data to business objects.

For connectors running on **ICS**, values for these properties are also required:

- Associated Maps
- Resources
- Messaging (where applicable)

Important: Connector Configurator accepts property values in either English or non-English character sets. However, the names of both standard and connector-specific properties, and the names of supported business objects, must use the English character set only.

Standard properties differ from connector-specific properties as follows:

- Standard properties of a connector are shared by both the application-specific component of a connector and its broker component. All connectors have the same set of standard properties. These properties are described in Appendix A of each adapter guide. You can change some but not all of these values.
- Application-specific properties apply only to the application-specific component of a connector, that is, the component that interacts directly with the application. Each connector has application-specific properties that are unique to its application. Some of these properties provide default values and some do not; you can modify some of the default values. The installation and configuration chapters of each adapter guide describe the application-specific properties and the recommended values.

The fields for **Standard Properties** and **Connector-Specific Properties** are color-coded to show which are configurable:

- A field with a grey background indicates a standard property. You can change the value but cannot change the name or remove the property.

- A field with a white background indicates an application-specific property. These properties vary according to the specific needs of the application or connector. You can change the value and delete these properties.
- Value fields are configurable.
- The **Update Method** field is displayed for each property. It indicates whether a component or agent restart is necessary to activate changed values. You cannot configure this setting.

Setting standard connector properties

To change the value of a standard property:

1. Click in the field whose value you want to set.
2. Either enter a value, or select one from the drop-down menu if it appears.
3. After entering all the values for the standard properties, you can do one of the following:
   - To discard the changes, preserve the original values, and exit Connector Configurator, click **File>Exit** (or close the window), and click **No** when prompted to save changes.
   - To enter values for other categories in Connector Configurator, select the tab for the category. The values you enter for **Standard Properties** (or any other category) are retained when you move to the next category. When you close the window, you are prompted to either save or discard the values that you entered in all the categories as a whole.
   - To save the revised values, click **File>Exit** (or close the window) and click **Yes** when prompted to save changes. Alternatively, click **Save>To File** from either the File menu or the toolbar.

Setting application-specific configuration properties

For application-specific configuration properties, you can add or change property names, configure values, delete a property, and encrypt a property. The default property length is 255 characters.

1. Right-click in the top left portion of the grid. A pop-up menu bar will appear. Click **Add** to add a property. To add a child property, right-click on the parent row number and click **Add child**.
2. Enter a value for the property or child property.
3. To encrypt a property, select the **Encrypt** box.
4. Choose to save or discard changes, as described for "Setting standard connector properties."

The Update Method displayed for each property indicates whether a component or agent restart is necessary to activate changed values.

Important: Changing a preset application-specific connector property name may cause a connector to fail. Certain property names may be needed by the connector to connect to an application or to run properly.

Encryption for connector properties

Application-specific properties can be encrypted by selecting the **Encrypt** check box in the Connector-specific Properties window. To decrypt a value, click to clear the **Encrypt** check box, enter the correct value in the **Verification** dialog box, and click **OK**. If the entered value is correct, the value is decrypted and displays.

The adapter user guide for each connector contains a list and description of each property and its default value.

If a property has multiple values, the **Encrypt** check box will appear for the first value of the property. When you select **Encrypt**, all values of the property will be encrypted. To decrypt multiple values of a property, click to clear the **Encrypt** check box for the first value of the property, and then enter the new value in the **Verification** dialog box. If the input value is a match, all multiple values will decrypt.

Update method

Refer to the descriptions of update methods found in the *Standard configuration properties for connectors* appendix, under "Setting and updating property values" on page 170.

Specifying supported business object definitions

Use the **Supported Business Objects** tab in Connector Configurator to specify the business objects that the connector will use. You must specify both generic business objects and application-specific business objects, and you must specify associations for the maps between the business objects.

Note: Some connectors require that certain business objects be specified as supported in order to perform event notification or additional configuration (using meta-objects) with their applications. For more information, see the *Connector Development Guide for C++* or the *Connector Development Guide for Java*.

If ICS is your broker

To specify that a business object definition is supported by the connector, or to change the support settings for an existing business object definition, click the **Supported Business Objects** tab and use the following fields.

Business object name: To designate that a business object definition is supported by the connector, with System Manager running:

1. Click an empty field in the **Business Object Name** list. A drop-down list displays, showing all the business object definitions that exist in the System Manager project.
2. Click on a business object to add it.
3. Set the **Agent Support** (described below) for the business object.
4. In the File menu of the Connector Configurator window, click **Save to Project**. The revised connector definition, including designated support for the added business object definition, is saved to an ICL (Integration Component Library) project in System Manager.

To delete a business object from the supported list:

1. To select a business object field, click the number to the left of the business object.
2. From the **Edit** menu of the Connector Configurator window, click **Delete Row**. The business object is removed from the list display.
3. From the **File** menu, click **Save to Project**.

Deleting a business object from the supported list changes the connector definition and makes the deleted business object unavailable for use in this implementation

of this connector. It does not affect the connector code, nor does it remove the business object definition itself from System Manager.

Agent support: If a business object has Agent Support, the system will attempt to use that business object for delivering data to an application via the connector agent.

Typically, application-specific business objects for a connector are supported by that connector's agent, but generic business objects are not.

To indicate that the business object is supported by the connector agent, check the **Agent Support** box. The Connector Configurator window does not validate your Agent Support selections.

Maximum transaction level: The maximum transaction level for a connector is the highest transaction level that the connector supports.

For most connectors, `Best Effort` is the only possible choice.

You must restart the server for changes in transaction level to take effect.

If a WebSphere Message Broker is your broker

If you are working in stand-alone mode (not connected to System Manager), you must enter the business object name manually.

If you have System Manager running, you can select the empty box under the **Business Object Name** column in the **Supported Business Objects** tab. A combo box appears with a list of the business object available from the Integration Component Library project to which the connector belongs. Select the business object you want from the list.

The **Message Set ID** is an optional field for WebSphere Business Integration Message Broker 5.0, and need not be unique if supplied. However, for WebSphere MQ Integrator and Integrator Broker 2.1, you must supply a unique **ID**.

If WAS is your broker

When WebSphere Application Server is selected as your broker type, Connector Configurator does not require message set IDs. The **Supported Business Objects** tab shows a **Business Object Name** column only for supported business objects.

If you are working in stand-alone mode (not connected to System Manager), you must enter the business object name manually.

If you have System Manager running, you can select the empty box under the Business Object Name column in the Supported Business Objects tab. A combo box appears with a list of the business objects available from the Integration Component Library project to which the connector belongs. Select the business object you want from this list.

Associated maps (ICS only)

Each connector supports a list of business object definitions and their associated maps that are currently active in WebSphere InterChange Server. This list appears when you select the **Associated Maps** tab.

The list of business objects contains the application-specific business object which the agent supports and the corresponding generic object that the controller sends

to the subscribing collaboration. The association of a map determines which map will be used to transform the application-specific business object to the generic business object or the generic business object to the application-specific business object.

If you are using maps that are uniquely defined for specific source and destination business objects, the maps will already be associated with their appropriate business objects when you open the display, and you will not need (or be able) to change them.

If more than one map is available for use by a supported business object, you will need to explicitly bind the business object with the map that it should use.

The **Associated Maps** tab displays the following fields:

- **Business Object Name**

  These are the business objects supported by this connector, as designated in the **Supported Business Objects** tab. If you designate additional business objects under the Supported Business Objects tab, they will be reflected in this list after you save the changes by choosing **Save to Project** from the **File** menu of the Connector Configurator window.
- **Associated Maps**

  The display shows all the maps that have been installed to the system for use with the supported business objects of the connector. The source business object for each map is shown to the left of the map name, in the **Business Object Name** display.
- **Explicit**

  In some cases, you may need to explicitly bind an associated map.

  Explicit binding is required only when more than one map exists for a particular supported business object. When ICS boots, it tries to automatically bind a map to each supported business object for each connector. If more than one map takes as its input the same business object, the server attempts to locate and bind one map that is the superset of the others.

  If there is no map that is the superset of the others, the server will not be able to bind the business object to a single map, and you will need to set the binding explicitly.

  To explicitly bind a map:

  1. In the **Explicit** column, place a check in the check box for the map you want to bind.
  2. Select the map that you intend to associate with the business object.
  3. In the **File** menu of the Connector Configurator window, click **Save to Project**.
  4. Deploy the project to ICS.
  5. Reboot the server for the changes to take effect.

Resources (ICS)

The **Resource** tab allows you to set a value that determines whether and to what extent the connector agent will handle multiple processes concurrently, using connector agent parallelism.

Not all connectors support this feature. If you are running a connector agent that was designed in Java to be multi-threaded, you are advised not to use this feature, since it is usually more efficient to use multiple threads than multiple processes.

Messaging (ICS)

The messaging properties are available only if you have set MQ as the value of the `DeliveryTransport` standard property and ICS as the broker type. These properties affect how your connector will use queues.

Setting trace/log file values

When you open a connector configuration file or a connector definition file, Connector Configurator uses the logging and tracing values of that file as default values. You can change those values in Connector Configurator.

To change the logging and tracing values:

1. Click the **Trace/Log Files** tab.
2. For either logging or tracing, you can choose to write messages to one or both of the following:
   - To console (STDOUT):
     Writes logging or tracing messages to the STDOUT display.

     Note: You can only use the STDOUT option from the **Trace/Log Files** tab for connectors running on the Windows platform.
   - To File:
     Writes logging or tracing messages to a file that you specify. To specify the file, click the directory button (ellipsis), navigate to the preferred location, provide a file name, and click **Save**. Logging or tracing message are written to the file and location that you specify.

     Note: Both logging and tracing files are simple text files. You can use the file extension that you prefer when you set their file names. For tracing files, however, it is advisable to use the extension `.trace` rather than `.trc`, to avoid confusion with other files that might reside on the system. For logging files, `.log` and `.txt` are typical file extensions.

Data handlers

The data handlers section is available for configuration only if you have designated a value of JMS for DeliveryTransport and a value of JMS for ContainerManagedEvents. Not all adapters make use of data handlers.

See the descriptions under `ContainerManagedEvents` in Appendix A, Standard Properties, for values to use for these properties. For additional details, see the *Connector Development Guide for C++* or the *Connector Development Guide for Java.*

Saving your configuration file

When you have finished configuring your connector, save the connector configuration file. Connector Configurator saves the file in the broker mode that you selected during configuration. The title bar of Connector Configurator always displays the broker mode (ICS, WMQI or WAS) that it is currently using.

The file is saved as an XML document. You can save the XML document in three ways:

- From System Manager, as a file with a `*.con` extension in an Integration Component Library, or
- In a directory that you specify.

- In stand-alone mode, as a file with a `*.cfg` extension in a directory folder. By default, the file is saved to `\WebSphereAdapters\bin\Data\App`.
- You can also save it to a WebSphere Application Server project if you have set one up.

For details about using projects in System Manager, and for further information about deployment, see the following implementation guides:

- For ICS: *Implementation Guide for WebSphere InterChange Server*
- For WebSphere Message Brokers: *Implementing Adapters with WebSphere Message Brokers*
- For WAS: *Implementing Adapters with WebSphere Application Server*

Changing a configuration file

You can change the integration broker setting for an existing configuration file. This enables you to use the file as a template for creating a new configuration file, which can be used with a different broker.

Note: You will need to change other configuration properties as well as the broker mode property if you switch integration brokers.

To change your broker selection within an existing configuration file (optional):

- Open the existing configuration file in Connector Configurator.
- Select the **Standard Properties** tab.
- In the **BrokerType** field of the Standard Properties tab, select the value that is appropriate for your broker.
  When you change the current value, the available tabs and field selections on the properties screen will immediately change, to show only those tabs and fields that pertain to the new broker you have selected.

Completing the configuration

After you have created a configuration file for a connector and modified it, make sure that the connector can locate the configuration file when the connector starts up.

To do so, open the startup file used for the connector, and verify that the location and file name used for the connector configuration file match exactly the name you have given the file and the directory or path where you have placed it.

Using Connector Configurator in a globalized environment

Connector Configurator is globalized and can handle character conversion between the configuration file and the integration broker. Connector Configurator uses native encoding. When it writes to the configuration file, it uses UTF-8 encoding.

Connector Configurator supports non-English characters in:

- All value fields
- Log file and trace file path (specified in the **Trace/Log files** tab)

The drop list for the `CharacterEncoding` and `Locale` standard configuration properties displays only a subset of supported values. To add other values to the drop list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory.

For example, to add the locale en_GB to the list of values for the Locale property, open the stdConnProps.xml file and add the line in boldface type below:

```
<Property name="Locale"
isRequired="true"
updateMethod="component restart">
                <ValidType>String</ValidType>
            <ValidValues>
                                <Value>ja_JP</Value>
                                <Value>ko_KR</Value>
                                <Value>zh_CN</Value>
                                <Value>zh_TW</Value>
                                <Value>fr_FR</Value>
                                <Value>de_DE</Value>
                                <Value>it_IT</Value>
                                <Value>es_ES</Value>
                                <Value>pt_BR</Value>
                                <Value>en_US</Value>
                                <Value>en_GB</Value>
                    <DefaultValue>en_US</DefaultValue>
            </ValidValues>
    </Property>
```

Appendix C. Adapter for Web Services tutorial

This appendix contains step-by-step procedures that:

- demonstrate asynchronous and synchronous event transmission for both request and event processing
- illustrate how to configure the web services connector for a SOAP/HTTPS sample
- illustrate how to configure the web services connector for a SOAP/HTTP sample
- illustrate how to configure the web services connector for a SOAP/JMS sample

About the tutorial

This tutorial is intended to demonstrate asynchronous and synchronous event transmission for both the request and event processing facets of the Adapter for Web Services with each of the supported protocols: SOAP/HTTP, SOAP/HTTPS and SOAP/JMS. In each scenario, the adapters act as:

- a web service client for collaborations that invoke a web service
- a proxy that exposes a WebSphere ICS collaboration as a web service

The tutorial is designed to show the basic functionality of the adapter in sample scenarios:

- **An asynchronous scenario** that illustrates an asynchronous (request-only) web service and its client with the connector. There are two samples in this scenario—for configuration simplicity, the same Web Services connector is used to expose a collaboration as a Web Service and invoke a Web Service as a client.
  - **A collaboration that is exposed as a web service** In this sample, the web service is simply a collaboration `SERVICE_ASYNCH_Order_Collab` within WebSphere ICS that is being exposed as a web service by the connector. The web service is referred to as Asynch Order Service. If the connector is properly configured, this Web Service can be invoked using any (one) of the Web Services protocols: SOAP/HTTP, SOAP/HTTPS or SOAP/JMS. `SERVICE_ASYNCH_Order_Collab` is a simple pass-through collaboration that takes `SERVICE_ASYNCH_TLO_Order`. The triggering port (From) of this collaboration is bound to the Web Services connector. The service port (To) is bound to SampleSiebelConnector.
  - **A collaboration that is invoked by a web services client** In this sample, the web service client is another collaboration `CLIENT_ASYNCH_Order_Collab` within WebSphere ICS that will invoke the Web Service Asynch Order Service using the Web Services connector. If the connector is configured properly, this web service client can invoke the Web Service over any (one) of the Web Services protocols: SOAP/HTTP, SOAP/HTTPS or SOAP/JMS. `CLIENT_ASYNCH_Order_Collab` is a simple pass-through collaboration which takes `CLIENT_ASYNCH_TLO_Order`. The triggering port (From) of this collaboration is bound to SampleSAPConnector. The service port (To) is bound to the Web Services connector.

Both samples in the asynchronous scenario involve two applications:

- SampleSiebel: Creates an order for its clients.
- SampleSAP: Creates an order

- **A synchronous scenario** that illustrates a synchronous (request-response) web service and its client with the connector. There are two samples in this scenario—for configuration simplicity, the same Web Services connector is used to expose a collaboration as a Web Service and invoke a Web Service as a client.
  - A collaboration that is exposed as a web service In this sample, the Web Service is simply a collaboration `SERVICE_SYNCH_OrderStatus_Collab` within WebSphere ICS that is being exposed as a web service by the connector. In this sample, this web service is referred to as Synch OrderStatus Service. If the connector is properly configured, the web service can be invoked using any of the web services protocols: SOAP/HTTP, SOAP/HTTPS or SOAP/JMS. `SERVICE_SYNCH_OrderStatus_Collab` is a simple pass-through collaboration which takes `SERVICE_SYNCH_TLO_OrderStatus`. The triggering port (From) of this collaboration is bound to the Web Services connector. The service port (To) is bound to SampleSiebelConnector.
  - **A collaboration that is invoked by a web services client** In this sample, the web service client is another collaboration `CLIENT_SYNCH_OrderStatus_Collab` within WebSphere ICS that will invoke the web service Synch OrderStatus Service using the Web Services connector. If the connector is properly configured, this web service client can invoke the web service over any of the web services protocols: SOAP/HTTP, SOAP/HTTPS or SOAP/JMS. `CLIENT_SYNCH_OrderStatus_Collab` is a simple pass-through collaboration which takes `CLIENT_SYNCH_TLO_OrderStatus`. The triggering port (From) of this collaboration is bound to SampleSAPConnector. The service port (To) is bound to the Web Services connector.

  Both samples in the synchronous scenario involve two applications:

  - SampleSiebel: Retrieves the status of orders for its clients.
  - SampleSAP: Requests the status of the order

Both scenarios involve simulating the SampleSiebelConnector and SampleSAPConnector using two Test Connectors.

Before you start

Before you start the tutorial, be sure that:

- You have installed, and are experienced with, WebSphere ICS 4.2.x or later.
- You have installed the WebSphere Business Integration Adapter For Web Services in the WebSphere ICS home directory.
- You are experienced with Web Services technology.
- You are experienced with SOAP technology.

Installing and configuring

In the sections that follow, *WBI_folder* refers to the folder containing your current WebSphere ICS installation. All environment variables and file separators are specified in the Windows NT/2000 format. Please make the appropriate changes if running on AIX or Solaris. (for example, WBI_folder\`connectors` would be WBI_folder/`connectors`).

Start server and tool

1. Start WebSphere InterChange Server (ICS) from the shortcut.
2. Start the WebSphere Business Integration System Manager and open the Component Navigator Perspective.
3. Register and connect your server as a Server Instance in the Interchange Server view.

Load the sample content

From the Component Navigator Perspective:

1. Create a new Integration Component Library.
2. Import the repos file named `WebServicesSample.jar` located in: WBI_folder\connectors\WebServices\samples\WebSphereICS\

Compile the collaboration templates

Using WebSphere Business Integration System Manager:

- **Compile All** of the Collaboration Templates that were imported from the `WebServicesSample.jar` repos file.

Configure the connector

1. If you have not done so already, configure the connector as described in this guide and according to your system.
2. Using WebSphere Business Integration System Manager, open WebServicesConnector in Connector Configurator.
3. You must also configure WebServicesConnector for the protocol you want to use with the sample:
   - If you want to use SOAP/HTTP, see "Configuring for the SOAP/HTTP protocol scenario" to configure the connector for SOAP/HTTP.
   - If you want to use SOAP/HTTPS, see "Configuring for the SOAP/HTTPS protocol scenario" on page 206 to configure the connector for SOAP/HTTPS.
   - If you want to use SOAP/JMS, see "Configuring for the SOAP/JMS protocol scenario" on page 208 to configure the connector for SOAP/JMS.

Configuring for the SOAP/HTTP protocol scenario

This section shows you how to configure the connector for the SOAP/HTTP sample scenario. As described in the body of this document, the connector includes a SOAP/HTTP protocol listener and SOAP/HTTP-HTTPS protocol handler. The sample scenario exposes `SERVICE_ASYNCH_Order_Collab` and `SERVICE_SYNCH_OrderStatus_Collab` collaborations as SOAP/HTTP web services. To expose a collaboration as a SOAP/HTTP web service, the connector uses the SOAP/HTTP protocol listener. The sample scenario comes with the `CLIENT_ASYNCH_Order_Collab` and `CLIENT_SYNCH_OrderStatus_Collab` collaborations, which are SOAP/HTTP clients of SOAP/HTTP web services. To invoke a SOAP/HTTP web service, the connector uses SOAP/HTTPHTTPS Protocol Handler.

In the steps and descriptions that follow, hierarchical connector configuration properties are represented with the " symbol. For example, A" B implies A is a hierarchical property, and B is child property of A.

To configure the SOAP/HTTP protocol listener for this sample:

1. In Connector Configurator, click on **Connector-Specific Properties** for the WebServicesConnector.
2. Expand the **ProtocolListenerFramework** property to display the ProtocolListeners child property.
3. Expand the **ProtocolListeners** child property to display the **SOAPHTTPListener1** child property.
4. Check the value of **SOAPHTTPListener1"Host** and **SOAPHTTPListener1"Port** property. Make sure there is no other process running on your host and listening on this TCP/IP port. Optionally, you may want to set the value of **SOAHTTPListener1"Host** to the machine name on which you will run the connector.

You need not configure the SOAP/HTTP-HTTPS protocol handler for the sample.

Configuring for the SOAP/HTTPS protocol scenario

This section shows you how to configure the connector for the SOAP/HTTPS sample scenario. The connector includes a SOAP/HTTPS protocol listener and SOAP/HTTP-HTTPS protocol handler. The sample scenario exposes the `SERVICE_ASYNCH_Order_Collab` and `SERVICE_SYNCH_OrderStatus_Collab` collaborations as SOAP/HTTPS web services. To expose a collaboration as a SOAP/HTTPS web service, the connector uses the SOAP/HTTPS protocol listener. The sample scenario comes with the `CLIENT_ASYNCH_Order_Collab` and `CLIENT_SYNCH_OrderStatus_Collab` collaborations, which are SOAP/HTTPS clients of SOAP/HTTPS web services. To invoke a SOAP/HTTPS web service, the connector uses the SOAP/HTTPHTTPS protocol handler.

In the steps and descriptions that follow, hierarchical connector configuration properties are represented with the " symbol. For example, A" B implies A is a hierarchical property, and B is child property of A.

Note: In addition to the pre-install items listed above in"Before you start" on page 204, you should also have created and tested your keystore and truststore using your Key and Certificate management software.

Configure SSL connector-specific properties: For SOAP/HTTPS, the connector requires that you configure the SSL connector-specific hierarchical property.

1. In Connector Configurator, click on the **Connector-Specific Properties** tab for the WebServicesConnector.
2. Expand the **SSL** hierarchical property to view all of its children properties. Additionally, check or change the following child properties of the hierarchical SSL connector-specific property.
   - **SSL" KeyStore** Set to the complete path to your keystore file, which you must create using your Key and Certificate management software.
   - **SSL"KeyStorePassword** Set to the password required to access your KeyStore.
   - **SSL"KeyStoreAlias** Set to the alias of the private key in your KeyStore.
   - **SSL"TrustStore** Set to the complete path of your truststore file which you have created using your Key and Certificate management software.
   - **SSL"TrustStorePassword** Set to the password required to access your TrustStore.

Note: Do not forget to save the changes in Connector Configurator.

Configure the SOAP/HTTPS protocol listener:

1. In Connector Configurator, click on **Connector-Specific Properties** for the WebServicesConnector.
2. Expand the **ProtocolListenerFramework** property to display the **ProtocolListeners** child property.
3. Expand the **ProtocolListeners** child property to display the **SOAPHTTPSListener1** child property. Check the value of the **SOAPHTTPSListener1"Host** and **SOAPHTTPSListener1"Port** properties. Make sure no other processes are running on your host and listening on this TCP/IP port. Optionally, you may want to set the value of **SOAHTTPSListener1"Host** to the machine name on which you are running the connector.

You need not configure the SOAP/HTTP-HTTPS protocol handler for the sample.

Setting up KeyStore and TrustStore: You can quickly set up KeyStore and TrustStore to use with the sample scenario. For production systems, you must use third-party software for to set up and manage keystores as well as certificate and key generation. No tool is provided as part of the Adapter for Web Services to set up and manage these resources.

This section assumes that Java Virtual Machine is installed on your system and that you are familiar with the keytool shipped with your JVM (Java Virtual Machine). For more information or for troubleshooting problems with the keytool, please see the documentation that accompanies your JVM.

To set up KeyStore:

1. You create KeyStore using keytool. You must create a key pair in the KeyStore. To do so, enter the following at the command line:

   ```
   keytool -genkey -alias wsadapter -keystore c:\security\keystore
   ```

2. keytool immediately prompts for a password. Specify the password that you entered for the value of SSL"KeyStorePassword connector property.
   Note that in the above example if you specified `-keystore c:\security\keystore` in the command line, you would enter `c:\security\keystore` as the value of the SSL"KeyStore property. Also, if you specified `-alias wsadapter` in the command line, you would enter `wsadapter` as the value of the SSL"KeyStoreAlias connector property. keytool would then prompt you for the details of the certificate. The following illustrates what you may enter at each of the prompts, but is an example only: always refer, and defer, to keytool documentation.

   ```
   What is your first and last name?
        [Unknown]:  HostName
   What is the name of your organizational unit?
        [Unknown]:  myunit
   What is the name of your organization?
        [Unknown]:  myorganization
   What is the name of your City or Locality?
        [Unknown]:  mycity
   What is the name of your State or Province?
        [Unknown]:  mystate
   What is the two-letter country code for this unit?
        [Unknown]:  mycountryIs <CN=HostName, OU=myunit, O=myorganization,
        L=mycity, ST=mystate, C=mycountry> correct?
        [no]:  yes
   ```

3. Note that for `What is your first and last name?`, you should enter the name of the machine on which you are running the connector. keytool then prompts you:

   ```
   Enter key password for <wsadapter> (RETURN if same as keystore password):
   ```

4. Press **Return** to use the same password. If you want to use a self-signed certificate, you may want to export the certificate created above. To do so, enter following on the command line:

   ```
   C:\security>keytool -export -alias wsadapter -keystore c:\security\keystore
   -file c:\security\wsadapter.cer
   ```

5. keytool now prompts for the keystore password. Enter the password that you entered above

To set up TrustStore:

1. To import the trusted certificates into the TrustStore, enter the following command:

   ```
   keytool -import -alias trusted1 -keystore c:\security\truststore
   -file c:\security\wsadapter.cer
   ```

2. keytool now prompts for the keystore password. If you entered `-keystore c:\security\truststore`, make sure that SSL"TrustStore property is set to `c:\security\truststore`. Also, set the value of the SSL"TrustStorePassword property to the password you entered above.

Configuring for the SOAP/JMS protocol scenario

This section shows you how to configure the connector for the SOAP/JMS sample scenario. The sample scenario exposes the `SERVICE_ASYNCH_Order_Collab` and `SERVICE_SYNCH_OrderStatus_Collab` collaborations as SOAP/JMS web services. To expose a collaboration as a SOAP/JMS web service, the connector uses the SOAP/JMS protocol listener. The sample scenario comes with the `CLIENT_ASYNCH_Order_Collab` and `CLIENT_SYNCH_OrderStatus_Collab` collaborations, which are SOAP/JMS clients of SOAP/JMS web services. To invoke a SOAP/JMS web service, the connector uses the SOAP/JMS protocol handler.

In the steps and descriptions that follow, hierarchical connector configuration properties are represented with the " symbol. For example, A" B implies A is a hierarchical property, and B is child property of A.

Note: In addition to the pre-install items listed above in"Before you start" on page 204, you should also have installed a JMS service provider and installed and configured your JNDI.

Configuring JNDI properties: For SOAP/JMS, you must configure JNDI connector configuration properties:

1. In Connector Configurator, click **Connector-Specific Properties** for the WebServicesConnector.
2. Expand the JNDI hierarchical property to display its child properties. Then check or change the child properties to match the values listed below.
   - **JNDI"JNDIProviderURL** Set this property to the URL of the JNDI Service provider. Refer to your JNDI provider documentation.
   - **JNDI"InitialContextFactory** Set this property to fully qualified class name of the factory class that will create the JNDI initial context. Refer to your JNDI provider documentation.
   - **JNDI"JNDIConnectionFactoryName** Set this property to the JNDI name of the connection factory to lookup using JNDI context. Make sure that this name can be looked up using the JNDI.
   - Refer to your JNDI documentation to see if any of the following properties are required by your JNDI provider:
     - **JNDI"CTX_ObjectFactories**

- **JNDI”CTX_ObjectFactories**
- **JNDI”CTX_StateFactories**
- **JNDI”CTX_URLPackagePrefixes**
- **JNDI”CTX_DNS_URL**
- **JNDI”CTX_Authoritative**
- **JND”CTX_Batchsize**
- **JNDI”CTX_Referral**
- **JNDI”CTX_SecurityProtocol**
- **JND”CTX_SecurityAuthentication**
- **JNDI”CTX_SecurityPrincipal**
- **JNDI”CTX_SecurityCredentials**
- **JNDI”CTX_Language**

3. Save the changes in Connector Configurator.

Configure the JMS queues and SOAP/JMS protocol listener: The scenario requires that six queues be defined with your JMS service provider. Before doing so, check your JMS provider documentation; defining queues varies between providers.

1. Define (or make available via JNDI lookup) the following queues:
   - ORDER_INPUT
   - ORDER_INPROGRESS
   - ORDER_ERROR
   - ORDER_ARCHIVE
   - ORDER_UNSUBSCRIBED
   - ORDER_REPLYTO
2. From CSM open **WebServicesConnector** in Connector Configurator. If you have not done so already, configure the connector as described in the installation guide for your system.
3. Click **Application Config Properties** in Connector Configurator.
4. Expand the **ProtocolListenerFramework** property to display the ProtocolListeners child property.
5. Expand **ProtocolListeners** property to display the SOAPJMSListener1 child property.
6. Check or change the values of the **SOAPJMSListner1** child properties to match those listed below:
   - **SOAPJMSListener”Protocol** Set to soap/jms
   - **SOAPJMSListener1”Protocol** Set to soap/jms
   - **SOAPJMSListener1”InputQueue** Set to ORDER_INPUT
   - **SOAPJMSListener1”InProgressQueue** Set to ORDER_INPROGRESS
   - **SOAPJMSListener1”ArchiveQueue** Set to ORDER_ARCHIVE
   - **SOAPJMSListener1”UnsubscribedQueue** Set to ORDER_UNSUBSCRIBED
   - **SOAPJMSListener1”ErrorQueue** Set to ORDER_ERROR
   - **SOAPJMSListener1”ReplyToQueue** Set to ORDER_REPLYTO
7. Save the changes in Connector Configurator.

Configure the SOAP/JMS protocol handler:

1. From System Manager open **WebServicesConnector** in Connector Configurator. If you have not done so already, configure the connector as described in the installation guide for your system.
2. Click **Connector-Config Properties** in Connector Configurator.
3. Expand the **ProtocolHandlerFramework** property to display the ProtocolHandlers child property.
4. Expand the **ProtocolHandlers** child property to display the SOAPJMSHandler child property. Check or change the values of SOAPJMSHandler child properties to match the those below:
   - **SOAPJMSHandler"Protocol** Set to `soap/jms`
   - **SOAPJMSHandler"ReplyToQueue** Set to value `ORDER_REPLYTO`
5. Save the changes in Connector Configurator.

Create user project

- Using WebSphere Business Integration System Manager, create a new **User Project**. Select all of the components from the Integration Component Library that was created in "Load the sample content" on page 205.

Add and deploy the project

1. From the Server Instance view, add the **User Project** created in "Create user project"to WebSphere ICS
2. Deploy all of the components from this User Project to the ICS.

Reboot ICS

1. Reboot ICS to ensure that all changes take effect.
2. Use the System Monitor tool to ensure that all of the collaboration objects, connector controllers, and maps are in a green state.

Running the asynchronous scenario

This scenario invokes the Asynch Order Service web service. Before running the scenario, review this step-by-step synopsis of its data flow.

1. A `CLIENT_ASYNCH_TLO_Order.Create` event originates in the application `SampleSAP` running in one instance of the Test Connector.
2. The event is sent from `SampleSAP` to the collaboration `CLIENT_ASYNCH_Order_Collab`.
3. The event is then sent from the collaboration to the Web Services connector.
4. The Web Services connector finds the `CLIENT_ASYNCH_Order` object that is a child of the `CLIENT_ASYNCH_TLO_Order` object.
5. The Request business object is converted into a SOAP message using the SOAP data handler.
6. The Web Services connector sends the SOAP Message to the end-point (Destination) of the web service Asynch Order Service. The end-point is provided by the Destination attribute of the Protocol Config Meta-Object (MO). The Protocol Config MO used by the connector depends on the value of the Handler attribute of `CLIENT_ASYNCH_TLO_Order`. If it is set to `soap/http` or `soap/https`, the Destination attribute of `CLIENT_ASYNCH_Order_SOAP_HTTP_CfgMO` will give the end-point as the URL of the web service. Otherwise if the Handler attribute is set to `soap/jms`, the Destination attribute of `CLIENT_ASYNCH_Order_SOAP_JMS_CfgMO` gives the end-point as a destination queue name.

7. The Asynch Order Service web service receives the SOAP request. As mentioned earlier, the Web Services connector is the end-point for this web service. The connector's protocol listener, listening on the end-point (to which the request was sent), receives the SOAP message.
8. The connector converts the SOAP message into SERVICE_ASYNCH_Order and then creates a SERVICE_TLO_Order object. The SERVICE_ASYNCH_Order object is set as a child of the SERVICE_TLO_Order object.
9. The Web Services connector now asynchronously posts the SERVICE_TLO_Order object to ICS. This completes the asynchronous web service invocation.

Because this is an asynchronous web service (request-only), no response is sent back to the web service client. When SERVICE_ASYNCH_Order_Collab receives this object, the collaboration then sends the business object to the application namedSampleSiebel, which is running as the second instance of Test Connector. The object is displayed in the Test Connector. When Reply Success is selected from theSampleSiebel application, the event will be sent back to SERVICE_ASYNCH_Order_Collab.

To run the asynchronous scenario:

1. Start your ICS integration broker, if it is not already running.
2. Start the Web Services connector.
3. Start two instances of the Test Connector.
4. Using the Test Connector, define a profile for the SampleSAPConnector and the SampleSiebelConnector.
5. Select **FILE"CONNECT AGENT** from each Test Connector menu to begin simulating agents.
6. While simulating the SampleSAPConnector using the Test Connector, select **EDIT"LOAD BO** from the menu. Load the following file:

   WBI_folder\connectors\WebServices\samples\WebSphereICS\OrderStatus\CLIENT_ASYNCH_TLO_Order.bo

   The Test Connector should show that the CLIENT_ASYNCH_TLO_Order is loaded.
7. Verify the web services end-point address:
   - **For SOAP/HTTP web service** If you want to use SOAP/HTTP:
     a. Make sure you have configured the Web Services connector for SOAP/HTTP. In your Test Connector, make sure that the value of the Handler attribute for the CLIENT_ASYNCH_TLO_Order business object is set to soap/http. No quotes are allowed in this value.
     b. Expand the Request attribute of CLIENT_ASYNCH_TLO_Order. This attribute is of type CLIENT_ASYNCH_Order business object.
     c. Expand the SOAPHTTPCfgMO attribute of CLIENT_ASYNCH_Order. This attribute is of type CLIENT_ASYNCH_Order_SOAP_HTTP_CfgMO.
     d. Make sure the value of the Destination attribute of CLIENT_ASYNCH_Order_SOAP_HTTP_CfgMO is set to http://localhost:8080/wbia/webservices/samples. No quotes are allowed in this value.
   - **For SOAP/HTTPS web service** If you want to use SOAP/HTTPS:
     a. Make sure that you have configured the Web Services connector for SOAP/HTTPS. In your Test Connector, make sure that the value of the Handler attribute for the CLIENT_ASYNCH_TLO_Order business object is set to soap/https. No quotes are allowed in this value.

b. Expand the Request attribute of CLIENT_ASYNCH_TLO_Order. This attribute is of type CLIENT_ASYNCH_Order business object.
c. Expand the SOAPHTTPCfgMO attribute of CLIENT_ASYNCH_Order. This attribute is of type CLIENT_ASYNCH_Order_SOAP_HTTP_CfgMO.
d. Make sure the value of the Destination attribute of CLIENT_ASYNCH_Order_SOAP_HTTP_CfgMO is set to https://localhost:8443/wbia/webservices/samples. No quotes are allowed in this value.

- **For SOAP/JMS web service** If you want to use SOAP/JMS:
  a. Make sure you have configured the Web Services connector for SOAP/JMS. In your Test Connector, make sure that the value of the Handler attribute of the CLIENT_ASYNCH_TLO_Order business object is set to soap/jms. No quotes are allowed in this value.
  b. Expand the Request attribute of CLIENT_ASYNCH_TLO_Order. This attribute is of type CLIENT_ASYNCH_Order business object.
  c. Expand the SOAPJMSCfgMO attribute of CLIENT_ASYNCH_Order. This attribute is of type CLIENT_ASYNCH_Order_SOAP_JMS_CfgMO.
  d. Make sure the value of the Destination attribute of CLIENT_ASYNCH_Order_SOAP_JMS_CfgMO is set to ORDER_INPUT. No quotes are allowed in this value.

8. While simulating the SampleSAPConnector with the Test Connector, click on the loaded **Test BO**. Select **REQUEST"SEND** from the menu. See the step-by-step synopsis earlier in this section for more details regarding the flow of the event.
9. While simulating the SampleSiebelConnector with the Test Connector, select **REQUEST"ACCEPT REQUEST**. An Event Labeled SERVICE_ASYNCH_TLO_Order.Create is displayed in the right panel of the Test Connector.
10. Double-click the business object. The business object opens up in a window.
11. Expand the Request attribute of the business object. The Request attribute is of type SERVICE_ASYNCH_Order. Inspect the OrderId, Customarily and other attributes of SERVICE_ASYNCH_Order to verify the Order received. This completes the execution of asynchronous scenario.
12. Once you have inspected the business object, close the window. Select **REQUEST "REPLY" SUCCESS**.

Running the synchronous scenario

This scenario invokes the Synch OrderStatus Service web service. Before running the scenario, review this step-by-step synopsis of its data flow.

1. A CLIENT_SYNCH_TLO_OrderStatus.Retrieve event originates in the application SampleSAP running in one instance of the Test Connector.
2. The event is sent from SampleSAP to the collaboration named CLIENT_SYNCH_OrderStatus_Collab.
3. The event is then sent from the collaboration to the Web Services connector.
4. The Web Services connector finds the CLIENT_SYNCH_OrderStatus_Request object, which is a child of the CLIENT_SYNCH_TLO_OrderStatus object.
5. The Web Services connector invokes the SOAP data handler to convert the CLIENT_SYNCH_OrderStatus_Request business object into a SOAP message.
6. The Web Services connector sends the SOAP message to the end-point (Destination) of the web service Synch OrderStatus Service. The end-point is

provided by the Destination attribute of the Protocol Config MO. The Protocol Config MO used by the connector depends on the value of the Handler attribute of CLIENT_SYNCH_TLO_OrderStatus. If it is set to soap/http or soap/https, the Destination attribute of CLIENT_SYNCH_OrderStatus_Request_SOAP_HTTP_CfgMO will give the end-point as the URL of a web service. Otherwise, if the Handler attribute is set to soap/jms, the Destination attribute of CLIENT_SYNCH_OrderStatus_Request_SOAP_JMS_CfgMO will give the end-point as the destination queue name of the web service).

7. The Web Service Synch OrderStatus Service receives the SOAP request. As mentioned earlier, the Web Services connector is the target end-point. The connector's protocol listener, listening on the end-point (to which request was sent), receive the SOAP message.
8. The connector invokes the SOAP data handler with the SOAP message. The SOAP message is converted into a SERVICE_SYNCH_OrderStatus_Request object by the SOAP data handler. The Web Services connector then creates a SERVICE_TLO_OrderStatus object. The SERVICE_SYNCH_OrderStatus_Request object is set as the child of the SERVICE_TLO_OrderStatus object.
9. The Web Services connector now synchronously posts the SERVICE_TLO_OrderStatus object to the SERVICE_SYNCH_OrderStatus_Collab collaboration running in WebSphere ICS. Since this is a synchronous execution, the Web Services connector remains blocked until the collaboration executes and returns the response.
10. SERVICE_SYNCH_OrderStatus_Collab receives the SERVICE_TLO_OrderStatus object. The collaboration then sends the business object to the application SampleSiebel, which is running as the second instance of the Test Connector.
11. When you select Reply Success from the SampleSiebel application, the event is sent back to the SERVICE_SYNCH_OrderStatus_Collab collaboration.
12. SERVICE_SYNCH_OrderStatus_Collab receives the SERVICE_TLO_OrderStatus object. The collaboration then sends the business object to Web Services connector.
13. The Web Services connector finds the SERVICE_SYNCH_OrderStatus_Response business object (or SERVICE_SYNCH_OrderStatus_Fault, if it is populated) that is a child of the SERVICE_SYNCH_OrderStatus_TLO. This business object will be converted into a SOAP response message (or SOAP fault message) by the SOAP data handler.
14. The Web Services connector returns the SOAP response message (or SOAP fault message) to the web service client.
15. The web service client, which in this case is the connector, receives the response. The connector invokes the SOAP data handler with the response message.
16. The SOAP data handler converts the response message into either a CLIENT_SYNCH_OrderStatus_Response or CLIENT_SYNCH_OrderStatus_Fault business object, depending on what was returned by the Order Synch Service. The Web Services connector sets this object as the child of CLIENT_SYNCH_OrderStatus_TLO. CLIENT_SYNCH_OrderStatus_TLO is returned to the CLIENT_SYNCH_OrderStatus_Collab collaboration.

17) CLIENT_SYNCH_OrderStatus_Collab then sends CLIENT_SYNCH_OrderStatus_TLO to the SampleSAP application, which is running as the first instance of the Test Connector. The Test Connector displays this object.

To run the synchronous scenario:

1. Start your ICS integration broker, if it is not already running.

2. Start the Web Services connector.
3. Start two instances of the Test Connector.
4. Using the Test Connector, define a profile for the SampleSAPConnector and the SampleSiebelConnector.
5. Select **FILE"CONNECT AGENT** from each Test Connector menu to begin simulating agents.
6. While simulating the SampleSAPConnector using the Test Connector, select **EDIT"LOAD BO** from the menu. Load the following file:

   WBI_folder\connectors\WebServices\samples\WebSphereICS\OrderStatus\CLIENT_SYNCH_TLO_OrderStatus.bo

   The Test Connector should show that the CLIENT_SYNCH_TLO_OrderStatus is loaded.
7. Verify the web services end-point address:
   - **For SOAP/HTTP web service** If you want to use SOAP/HTTP:
     a. Make sure you have configured the Web Services connector for SOAP/HTTP. In your Test Connector, make sure that the value of the Handler attribute for the CLIENT_SYNCH_TLO_OrderStatus business object is set to soap/http. No quotes are allowed in this value.
     b. Expand the Request attribute of CLIENT_SYNCH_TLO_OrderStatus. This attribute is of type CLIENT_SYNCH_OrderStatus business object.
     c. Expand SOAPHTTPCfgMO attribute of CLIENT_SYNCH_OrderStatus. This attribute is of type CLIENT_SYNCH_OrderStatus_SOAP_HTTP_CfgMO.
     d. Make sure the value of the Destination attribute of CLIENT_SYNCH_OrderStatus_SOAP_HTTP_CfgMO is set to http://localhost:8080/wbia/webservices/samples. No quotes are allowed in this value.
   - **For SOAP/HTTPS web service** If you want to use SOAP/HTTPS:
     a. Make sure that you have configured the Web Services connector for SOAP/HTTPS. In your Test Connector, make sure that the value of the Handler attribute for the CLIENT_SYNCH_TLO_OrderStatus business object is set to soap/https. No quotes are allowed in this value.
     b. Expand the Request attribute of CLIENT_SYNCH_TLO_OrderStatus. This attribute is of type CLIENT_SYNCH_OrderStatus business object.
     c. Expand the SOAPHTTPCfgMO attribute of CLIENT_SYNCH_OrderStatus. This attribute is of type CLIENT_SYNCH_OrderStatus_SOAP_HTTP_CfgMO.
     d. Make sure value of Destination attribute of CLIENT_SYNCH_OrderStatus_SOAP_HTTP_CfgMO is set to https://localhost:8443/wbia/webservices/samples. No quotes are allowed in this value.
   - **For SOAP/JMS web service** If you want to use SOAP/JMS:
     a. Make sure you have configured the Web Services connector for SOAP/JMS. In your Test Connector, make sure that the value of the Handler attribute of the CLIENT_SYNCH_TLO_OrderStatus business object is set to soap/jms. No quotes are allowed in this value.
     b. Expand the Request attribute of CLIENT_SYNCH_TLO_OrderStatus. This attribute is of type CLIENT_SYNCH_OrderStatus business object.
     c. Expand the SOAPJMSCfgMO attribute of CLIENT_SYNCH_OrderStatus. This attribute is of type CLIENT_SYNCH_OrderStatus_SOAP_JMS_CfgMO.

d. Make sure the value of the Destination attribute of CLIENT_SYNCH_OrderStatus_SOAP_JMS_CfgMO is set to ORDER_INPUT. No quotes are allowed in this value.

8. While simulating the SampleSAPConnector with the Test Connector, click on the loaded **Test BO**. Select **REQUEST"SEND** from the menu. See the step-by-step synopsis earlier in this section for more details regarding the data flow.
9. An event labeled SERVICE_SYNCH_TLO_OrderStatus.Retrieve is displayed in the right panel of the Test Connector instance that is simulating SampleSiebelConnector. Double-click the business object to display it in a window.
10. Expand the Request attribute of the business object. The Request attribute is of type SERVICE_SYNCH_OrderStatus_Request. Inspect the OrderId, attribute of SERVICE_ASYNCH_Order to verify that this is the order for which status is required.
    - If you know the status of the order:
      a. Click the **Response** attribute of SERVICE_SYNCH_TLO_OrderStatus. The Response attribute is of type CLIENT_SYNCH_OrderStatus_Response.
      b. Right-click the **Response** attribute.
      c. Click the **Add Instance** option. A new instance for the CLIENT_SYNCH_OrderStatus_Response business object is created.
      d. Enter values for **OrderId**, **CustomerId** and all other details you know about this order. Once you have entered all the details for this order, close this window.
    - If you do not know the status of the order:
      a. Click the **Fault** attribute of SERVICE_SYNCH_TLO_OrderStatus. The Fault attribute is of type CLIENT_SYNCH_OrderStatus_Fault.
      b. Right-click the **Fault** attribute.
      c. Click the **Add Instance** option. A new instance of CLIENT_SYNCH_OrderStatus_Fault is created.
      d. Enter values for faultcode, faultstring and all other details you want to send in the SOAP fault message. Once you have entered all the values for this fault, close this window.
11. Select **REQUEST"REPLY"SUCCESS.**An event labeled SERVICE_SYNCH_TLO_OrderStatus.Retrieve is displayed in the right panel of the Test Connector that is simulating SampleSAPConnector.
12. Double-click the **SERVICE_SYNCH_TLO_OrderStatus.Retrieve** business object, which is then displayed in a window.
    - If your SampleSiebelConnector returned an order status, you should see the Response attribute of the business object populated. Expand the **Response** attribute to verify the order status.
    - If your SampleSiebelConnector returned a fault, you should see the Fault attribute of the business object populated. Expand the **Fault** attribute to determine the fault.
13. Once you have inspected the business object, close the window. Select **REQUEST"REPLY"SUCCESS**.

This completes the execution of synchronous scenario.

Appendix D. Migrating to 3.0.x

This appendix describes the backwards compatibility of the 3.0.x release of the Adapter for Web Services as well as how to migrate from Web Services Adapter 1.x and 2.x releases.

Backward compatibility

The Adapter for Web Services, version 3.0.x, is not backward compatible:

- None of the new components (web services connector, SOAP data handler, WSDL ODA) can be used, either jointly or individually, with components released in prior versions of this product.
- None of the components (SOAP connector, SOAP data handler, Web Services Generation Utility) released in prior versions of this product can be used either jointly or individually with version 3.0.x.
- Artifacts created or used with the prior versions of this product solution may not be usable with the version 3.0.x.
- The 3.0.x version of this product cannot be used with a release of WebSphere ICS that is prior to version 4.2

Upgrade tasks

The sections below describe how to upgrade components from versions 1.x and 2.x of this product.

Web Services Generation Utility

The Web Services Generation Utility is no longer available. Instead, you use System Manager tools to expose collaborations as web services. None of the artifacts generated by the Web Services Generation Utility can be used with this release:

- **Proxy class** The web services connector now supports event notification. Therefore proxy classes are no longer required to expose a collaborations as a web service. A proxy class generated with the Web Services Generation Utility cannot invoke a collaboration that has been exposed as a web service using System Manager version 4.2 tools.
- **WSDL documents** The Web Services Generation Utility cannot be used to generate WSDL documents for ICS version 4.2 collaborations. Instead System Manager tools must be used to generate WSDL documents. For more information see "Running the wizard" on page 146.
  When you enable a collaboration for request processing, WSDL documents that you generated using the Web Services Generation Utility may not be usable with the WSDL ODA that is available with the 3.0.x release of the connector.

SOAP data handler

You can use the SOAP data handler with the web services connector only. This data handler cannot be used by any other connector nor by Server Access Interface. The sections below discuss additional support issues.

Meta-objects

The top-level SOAP data handler meta-object used with prior releases is no longer supported. Instead you must create a new top-level meta-object for use with the 3.0.x release SOAP data handler. This meta-object must have Classname and SOAPNameHandler attributes only.

The new meta-object no longer requires child meta objects for SOAP message-to-business-object and business-object -to-SOAP-message transformations. Accordingly, make sure that your top-level meta-object does not have SOAPToBOConfigMO and BOToSOAPConfigMO attributes.

The child meta-objects that previously described SOAP message-to-business-object and business-object -to-SOAP-message transformations must now be added as children of the SOAP Request, SOAP Response and SOAP Fault business objects. For further information, see Chapter 5, “SOAP data handler,” on page 107 and Chapter 3, “Business object requirements,” on page 21.

Application-Specific Information

The new SOAP data handler features new application-specific information (ASI) functionality. You can take advantage of this enhancement by adding specific ASI to SOAP business objects, but doing so is not required. With the exception of adding child SOAP Config MOs to business objects, you can deploy SOAP business objects that you created with prior releases of the connector for use with the 3.0.x version.

Connector compatibility

You can use the new SOAP data handler with the 3.0.x web services connector only. The new SOAP data handler cannot be used with components from prior releases such as the SOAP connector or Server Access Interface.

You cannot use the old SOAP data handler with the 3.0.x web services connector.

SOAP connector

The SOAP connector is not supported with release 3.0.x. Instead, you must use the web services connector to invoke web services.

Web services connector

With release 3.0.x, you use the web services connector for both exposing collaborations as web services and invoking web services. New event notification functionality is used to expose collaborations as web services. New request processing features are now used to invoke web services. The sections below highlight the migration tasks that you must complete to use the web services connector.

Note: The migration tasks discussed below may not be exhaustive. Also, you can complete the tasks in any order.

Event notification

The 3.0.x connector can invoke collaborations synchronously or asynchronously with no requirement for creating and deploying a proxy class on a web server. The

connector now has a listener framework that notifies the connector of events. The listener framework supports SOAP/HTTP, SOAP/HTTPS and SOAP/JMS bindings. If you configure the listeners properly, the connector can detect and respond to web service clients on behalf of collaborations that have been exposed as web services. For further information on the listener framework and how to configure it, see "Protocol listeners" on page 61.

Business objects for event notification: You must create an event notification top-level object (TLO). The TLO is a container for a SOAP Request business object and, optionally (for synchronous request processing), a SOAP Response and SOAP Fault business object. The TLO's structural components anticipate a single web services operation: the SOAP Request business object corresponds to a SOAP request message, the SOAP Response business object corresponds to a SOAP response message, and the SOAP Fault business object corresponds to a SOAP fault message. For further information, see "Synchronous event processing TLOs" on page 22.

Event notification and SOAP business objects: The SOAP business objects used with Server Access Interface in prior releases may be used, with modifications described in "SOAP data handler" on page 218 above, with the 3.0.x release. Note that you must specify SOAP business objects as children in the event notification TLO.

Request Processing

Like the SOAP connector in prior releases, the 3.0.x web services connector can invoke web services. In addition, the new connector supports invocation of web services with SOAP/JMS bindings. The sections below discuss further the changes in connector request processing.

Top-level objects request processing: You must create a request processing TLO. The TLO is a container for a SOAP Request business object and, optionally (for synchronous request processing), a SOAP Response and SOAP Fault business object. The TLO's structural components anticipate a single web services operation: the SOAP Request business object corresponds to a SOAP request message, the SOAP Response business object corresponds to a SOAP response message, and the SOAP Fault business object corresponds to a SOAP fault message. In this sense the 3.0.x request processing TLO corresponds to the TLO used with the SOAP connector from prior releases. For further information on request processing TLOs, see "Synchronous request processing TLOs" on page 40.

SOAP business objects: The SOAP business objects used with the SOAP connector of prior releases may be used with modifications as described in"SOAP data handler" on page 218. You must specify these business objects as children of a request processing TLO. Note that in previous releases these business objects were children of a TLO used with SOAP connector.

The 3.0.x web services connector has an additional requirement for SOAP Request business objects. Each SOAP Request business object must have an attribute of type Protocol Config MO (meta-object). The connector uses the Protocol Config MO to determine the destination of the request message. Each Protocol Config MO has a Destination attribute that gives the address of the target web service. If you are using SOAP/HTTP or SOAP/HTTPS to invoke the target web service, then the Destination attribute corresponds to the URL attribute of the TLO used with the SOAP connector from prior releases.For further information, see "Protocol Config MO" on page 29.

Appendix E. Configuring HTTPS/SSL

If you are planning to use SSL, you must use third-party software to manage your keystores, certificates, and key generation. The web services connector does not come with tooling for these tasks. However, you may choose to use keytool, which ships with IBM JRE, to create self-signed certificates and to manage your keystores.

A key and certificate management utility, keytool enables you to administer your own public/private key pairs and associated certificates. These are intended for use in self-authentication (where you authenticate yourself to other users or services) or data integrity and authentication services that use digital signatures. The keytool utility also allows you to store the public keys (in the form of certificates) of peers with whom you communicate.

This appendix describes how to set up keystores using keytool. Note that this appendix is intended for illustration purposes only; it is not intended as a substitute for documentation for keytool or related products. Always refer to source documentation for the tools you use to set up keystores. For further information on keytool, see:

- http://java.sun.com/j2se/1.3/docs/tooldocs/tools.html#security

Keystore setup

To create KeyStore using keytool, you first must create a key pair in the KeyStore. For example, if you enter the following command line:

```
keytool -genkey -alias wsadapter -keystore c:\security\keystore
```

keytool immediately prompts you for a password. You may enter the password of your choice (within keytool parameters), but you should specify the password entered in keytool as the value of the SSL " KeyStorePassword connector property. For further information, see "KeyStorePassword" on page 100.

The sample command creates the keystore named `keystore` in the `c:\security\keystore` directory. Accordingly, you would enter `c:\security\keystore` as the value of the SSL " KeyStore connector hierarchical property. Also from the command line example above, you would enter `-alias wsadapter` as the value of the SSL " KeyStoreAlias connector hierarchical property. The keytool utility then prompts you for the details of the certificate. The following illustrates what you may enter for each of the prompts. (Refer to keytool documentation.)

```
What is your first and last name?
     [Unknown]:  HostName
What is the name of your organizational unit?
     [Unknown]:  wbi
What is the name of your organization?
     [Unknown]:  IBM
What is the name of your City or Locality?
     [Unknown]:  Burlingame
What is the name of your State or Province?
```

```
     [Unknown]:  CA
What is the two-letter country code for this unit?
    [Unknown]:  US
Is <CN=HostName, OU=wbi, O=IBM, L=Burlingame,
ST=CA, C=US> correct?
    [no]:  yes
```

keytool then prompts you for a password:

```
Enter key password for <wsadapter> (RETURN if same as keystore password):
```

Press Return to use the same password. If you want to use a self-signed certificate, you may want to export the certificate created above. In that case, enter following on the command line:

```
keytool -export -alias wsadapter -keystore c:\security\keystore -file wsadapter.cer
```

keytool now prompts you for the keystore password. Enter the password that you entered above.

TrustStore setup

You may want to set up TrustStore for the following: If you want the SOAP/HTTPS protocol listener to authenticate the web service client, set the SSL " UseClientAuth connector configuration property to `true` . In this case, the SOAP/HTTPS protocol listener expect s TrustStore to contain certificates for all trusted web service clients. Note that the connector uses the JSSE default mechanism to trust clients. If you are invoking SOAP/HTTPS web services, the SOAP/HTTP-HTTPS protocol handler requires that TrustStore trust the web service. This means that TrustStore must contain the certificates of all trusted web services. Note that the connector uses the JSSE default mechanism to trust clients. To import the trusted certificates into the TrustStore, enter a command such as the following:

```
keytool -import -alias trusted1 -keystore c:\security\truststore -file
 c:\security\trusted1.cer
```

keytool now prompts for the keystore password. If you enter `-keystore c:\security\truststore`, make sure that the SSL " TrustStore hierarchical property is set to `c:\security\truststore`. Also you must set the value of the SSL " TrustStorePassword hierarchical property to the password you entered previously.

Generating a certificate signing request (CSR) for public key certificates

If the SSL data exchange is among already trusted partners who trust your identity, self-signed certificates may be adequate. However, a certificate is more likely to be trusted by others when it is signed by a certifying authority (CA).

To get a certificate signed by the CA using the keytool utility, you first must generate a Certificate Signing Request (CSR), then give the CSR to a CA. The CA then signs the certificate and returns it to you.

You generate a CSR by entering the following command:

```
keytool -certreq -alias wsadapter -file wsadapter.csr
 -keystore c:\security\keystore
```

In the command, `alias` is the keystore alias that you created for the private key. The keytool utility generates the CSR file, which you provide to your CA. Your CA

then provides you with the signed certificate. You will have to import this certificate into your keystore. To do so, you would enter the following command:

```
keytool -import -alias wsadapter -keystore c:\security\keystore -trustcacerts
-file casignedcertificate.cer
```

Once you import, the self-signed certificate in keystore is replaced by the CA-signed certificate.

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Burlingame Laboratory Director
IBM Burlingame Laboratory
577 Airport Blvd., Suite 800

Burlingame, CA 94010
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not necessarily tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

COPYRIGHT LICENSE

This information may contain sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both:

IBM
the IBM logo
AIX
CrossWorlds
DB2
DB2 Universal Database
Domino
Lotus
Lotus Notes
MQIntegrator
MQSeries
Tivoli
WebSphere

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

MMX, Pentium, and ProShare are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

IBM WebSphere Business Integration Adapter Framework,V2.4.0

Printed in USA