IBM WebSphere Business Integration Adapters

Adapter for HTTP User Guide

V 1.1.x

IBM WebSphere Business Integration Adapters

Adapter for HTTP User Guide

V 1.1.x

Note!
Before using this information and the product it supports, read the information in "Notices" on page 119.

25June2004

This edition of this document applies to IBM WebSphere Business Integration Adapter for HTTP (5724-H49), version 1.1.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 HTTP.

Audience

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

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, http protocol. 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, and for using adapters with WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, WebSphere Business Integration Message Broker), see the IBM WebSphere Business Integration Adapters InfoCenter:
  http://www.ibm.com/websphere/integration/wbiadapters/infocenter
- For using adapters with WebSphere InterChange Server, see the IBM WebSphere InterChange Server InfoCenters:
  http://www.ibm.com/websphere/integration/wicserver/infocenter
  http://www.ibm.com/websphere/integration/wbicollaborations/infocenter
- For more information about WebSphere message brokers:
  http://www.ibm.com/software/integration/mqfamily/library/manualsa/

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.
ProductDir	Represents the directory where the product is installed.
->	Indicates a choice from a menu such as: Choose File ->Update -> SGML References

New in this release

New in release 1.1

This release includes the following enhancements:

- 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 53.
- 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 57.
- In the case of synchronous event processing by 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 1.1.x, the adapter is not supported on Solaris 7, so references to that platform version have been deleted from this guide.

Chapter 1. Overview of the Adapter

The connector is a runtime component of the WebSphere Business Integration Adapter for HTTP. The connector allows businesses to aggregate, publish, and consume HTTP(S) messages 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 message that can be conveyed via the HTTP and HTTPS protocols.

This chapter describes the scope, components, design tools, and architecture used to implement the WebSphere Business Integration Adapter for HTTP. It also provides an overview of tasks you must complete to install and configure the HTTP components described in this document. For information about installing and configuring the components, see "Install, configure, and design checklist" on page 8.

Note: The adapter for HTTP 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.

Adapter for HTTP 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 1.1 version of the adapter for HTTP is supported on the following adapter framework and integration brokers:

- Adapter framework: WebSphere Business Integration Adapter Framework, versions:
  - 2.2.0
  - 2.3.0
  - 2.3.1
  - 2.4.0
- WebSphere InterChange Server, versions:

  - 4.2
  - 4.2.1
  - 4.2.2
- WebSphere MQ Integrator 2.1.0
- WebSphere MQ Integrator Broker 2.1.0
- WebSphere Business Integration Message Broker 5.0

See the Release Notes for any exceptions.

Software prerequisites

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

- 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

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

The standards used by the adapter are as follows:

- HTTP 1.0

The APIs used by the adapter are as follows:

- IBM JSSE 1.0.2

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

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 53.

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 preserve 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.

HTTP connector

This section discusses globalization and the connector.

Event notification: The connector uses pluggable protocol listeners for event notification. The protocol listeners extract the message from the transport and invoke the data handler specified in the message meta-data. For further information on listener processing, see "HTTP and HTTPS protocol listener processing" on page 42.

Request processing: The connector uses a pluggable HTTP-HTTPS protocol handler framework for request processing. The protocol handlers invoke the data handler. For further information, see "HTTP-HTTPS protocol handler processing" on page 49.

Data handler

You can configure the HTTP adapter to use any data handler. For an overview of data handler configuration, see "Configuring the data handler" on page 9.

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.
- **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.
- **Protocol Config MO** During request processing, the HTTP-HTTPS protocol handlers use a Protocol Config MO to determine the destination. If during event processing you are exposing collaborations, the connector uses the Protocol Config MO to convey message header information from the HTTP or HTTPS protocol listener to the collaboration.

- **Top-Level Business Object** A top-level business object contains a Request, a Response (optional) and one or more Fault (optional) business objects. A TLO is used by the connector for both event processing and request processing.

Components of connector for HTTP

Figure 1 illustrates the connector for HTTP, including its protocol handler and listener frameworks.

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

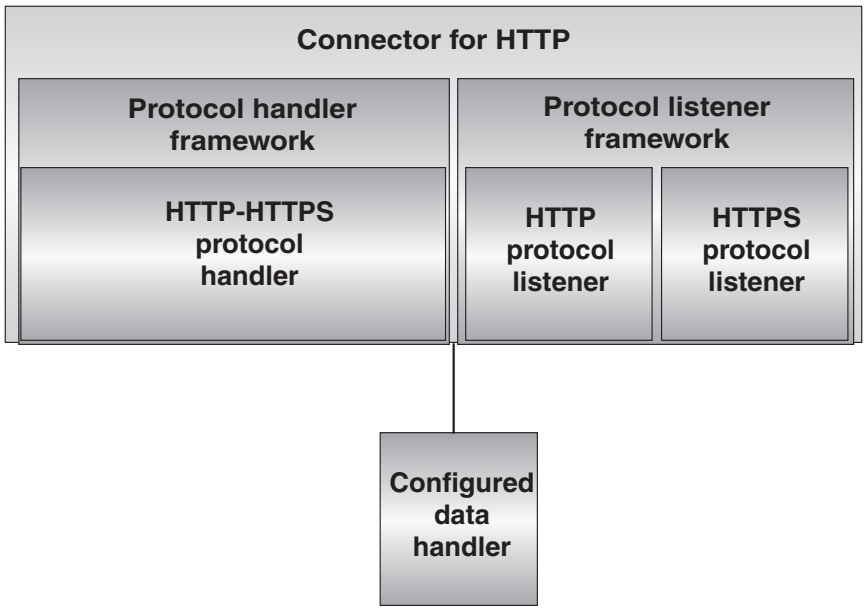


Figure 1. The connector for HTTP

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

- HTTP connector, including the configured data handler and protocol listeners and handlers
- HTTP-enabled collaborations
- Business objects and HTTP(S) messages
- WebSphere Business Integration InterChange Server

Connector for HTTP

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

During event processing, the connector processes request messages from clients by converting them into request business objects and passing them on to collaborations for processing. The connector optionally receives response business objects from the collaboration, which are converted to response messages and then returned to clients.

For further information, see Chapter 4, "HTTP connector," on page 39

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

Protocol listeners and handlers

The connector includes the following protocol listeners and handler:

- HTTP protocol listener
- HTTPS protocol listener
- HTTP-HTTPS protocol handler

Protocol listeners detect events from internal or external clients in HTTP, or HTTPS formats. They notify the connector of events that require processing by a collaboration. 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 transaction. For a detailed account of protocol listener processing, see "Protocol listeners" on page 42.

Protocol handlers invoke HTTP services in HTTP or HTTPS formats on behalf of a collaboration. The HTTP(S) protocol handler reads 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 messages to business objects and vice versa, and to determine the destination (from the Destination attribute of the request business object Protocol Config MO). For synchronous transactions, the protocol handler processes response messages, converting them into response business objects and passing them back to the collaboration.

For further information on protocol handlers, see "Protocol handling" on page 48.

Data handler

You can configure the HTTP adapter to use any data handler. For purposes of illustration, this document often makes references to a `text/xml` mime type and an XML data handler.

The configured data handler converts business objects to messages and vice versa. For further information see the documentation for the data handler you are using with the HTTP adapter.

Object discovery agents

If you are using a data handler for which there is an object discovery agent (ODA), you can use that ODA to generate business objects. For example, if your requirements include XML encoding and if you configure the adapter with the XML data handler, you can use the XML ODA to create and modify business objects.

Deploying the connector

There are two ways to deploy the HTTP connector:

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

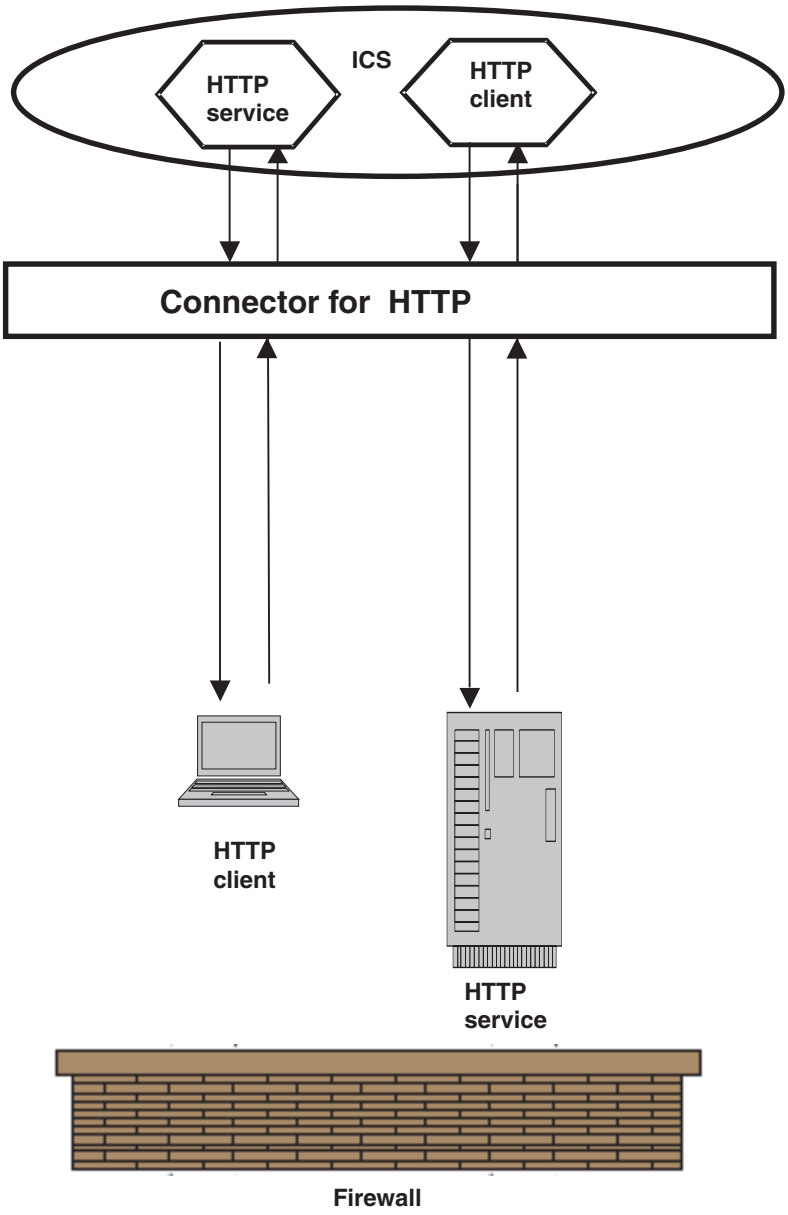


Figure 2. HTTP adapter as an intranet solution

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

Note: The HTTP connector does not include a gateway or front-end for managing incoming or outgoing messages from or to external clients. 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 HTTP

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.

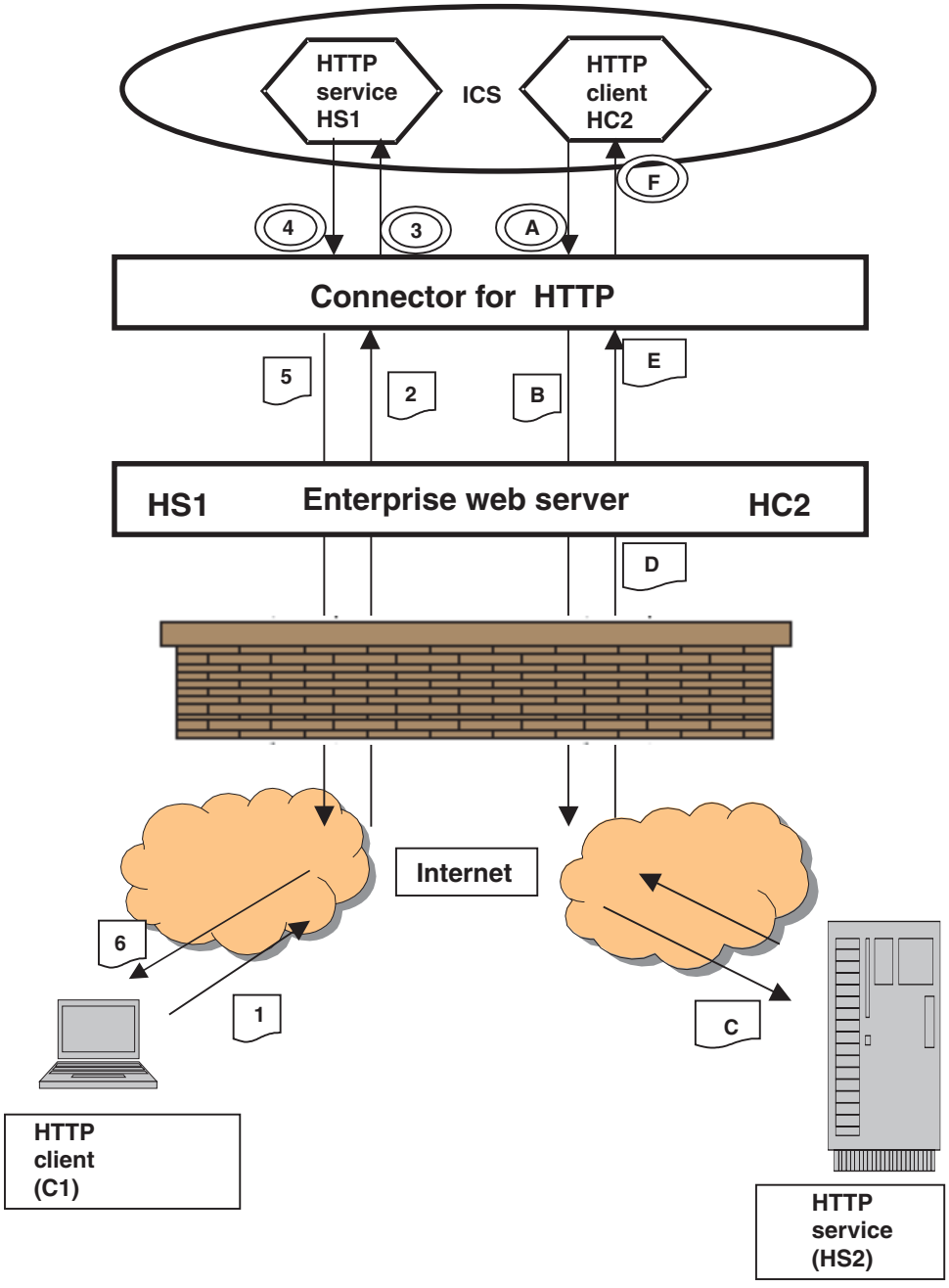


Figure 3. Flow of an HTTP message

Request processing illustrates the sequence of events that occurs when a collaboration makes a service call request to the connector. 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 a data handler to convert the business object to a request message.

B The connector invokes the URL of the enterprise web server by sending the request message.

C The enterprise web server invokes the URL of the HTTP server (HS2).

D The HTTP server HS2 processes the request and returns the response. The response is returned as part of the same connection.

E The enterprise web server returns the response message to the adapter.

F The connector receives the response (or fault) message, calls the data handler to convert the message to a business object, and returns it to the collabortion.

Event processing illustrates the sequence of events that occurs when a collaboration is invoked by an HTTP client. In this scenario, the collaboration plays the role of the server, accepting a request from a client, external or internal, and responding as required.

1 The HTTP client (C1) sends a request message to the destination—the collaboration.

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

3 The connector sends the message to the data handler for conversion to a business object. The connector invokes the collaboration.

4 The collaboration returns a response (or fault) business object.

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

6 If the client is external, the gateway routes the response message to the HTTP client (C1).

Install, configure, and design checklist

This section summarizes the tasks you must perform to install, configure, and design your HTTP 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 11 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, “HTTP connector,” on page 39.

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, “HTTP connector,” on page 39.

Creating or modifying business objects

Depending on the data handler(s) you are using with the HTTP connector, there may be an ODA available. ODA’s automate the process of creating and modifying business objects. Otherwise, you can manually create or modify business objects using Business Object Designer. For further information, see the documentation for the data handler you are using and the *Business Object Development Guide*.

Configuring the data handler

You configure the data handler meta-object(s) after you install the product files, but before startup. You start by specifying a connector-specific configuration property, the DataHandlerMetaObjectName. You specify the name of the top-level meta-object (MO_DataHandler_Default) that the data handler uses to retrieve configuration properties. Then follow any additional configuration steps required by the data handler you are using. You can optionally specify a data handler using the MimeType TLO attribute. For further information, see Table 5 on page 20.

For further information on configuring the data handler, see “Connector-specific configuration properties” on page 55

Chapter 2. Installation and startup

This chapter describes how to install components for implementing the connector for HTTP. 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 HTTP environment" on page 1.

To install the connector for HTTP, 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
connectors\HTTP\BIA_HTTP.jar	WebSphere Business Integration Adapter jar file
connectors\HTTP\start_HTTP.bat	The startup file for the connector
bin\Data\App\HTTPConnectorTemplate	HTTP connector template
connectors\HTTP\dependencies\ibmjsse.jar	JSSE (Java Secure Socket Extension) API from IBM
connectors\HTTP\dependencies\IBMReadme.txt	License
connectors\HTTP\samples\WebSphereICS\HTTPSample.jar	Repository file for samples
connectors\HTTP\samples\WebSphereICS\CLIENT_SYNCH_TLO_OrderStatus.bo	Sample (synchronous) business object for test connector
connectors\HTTP\samples\WebSphereICS\CLIENT_ASYNCH_TLO_Order.bo	Sample (asynchronous) business object for test connector
connectors\HTTP\samples\WebSphereICS\SERVICE_SYNCH_TLO_OrderStatus.bo	Sample (synchronous) business object
connectors\messages\HTTPConnector.txt	Connector message file

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
connectors/HTTP/BIA_HTTP.jar	WebSphere Business Integration Adapter jar file
connectors/HTTP/start_HTTP.sh	The startup file for the connector
bin/Data/App/HTTPConnectorTemplate	HTTP connector template
connectors/HTTP/dependencies/ibmjsse.jar	JSSE (Java Secure Socket Extension) API from IBM
connectors/HTTP/dependencies/IBMReadme.txt	License
connectors/HTTP/samples/WebSphereICS/HTTPSample.jar	Repository file for samples
connectors/HTTP/samples/WebSphereICS/CLIENT_SYNCH_TLO_OrderStatus.bo	Sample (synchronous) business object for test connector
connectors/HTTP/samples/WebSphereICS/CLIENT_ASYNCH_TLO_Order.bo	Sample (asynchronous) business object for test connector
connectors/HTTP/samples/WebSphereICS/SERVICE_SYNCH_TLO_OrderStatus.bo	Sample (synchronous) business object
connectors/messages/HTTPConnector.txt	Connector message file

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 55.

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:
  - Outgoing request messages
  - Each possible response, including faults

  For further information, review Chapter 3, "Business object requirements," on page 17.
- **Event Processing** You use TLO business objects. For further information, review Chapter 3, "Business object requirements," on page 17.

You can create business objects manually using Business Object Designer or, depending on your data handler, an ODA, which automates the process of making business objects. For further information, see your data handler documentation.

Configure the data handler

You configure the data handler by specifying a connector-specific configuration property, the DataHandlerMetaObjectName. You specify the name of the top-level meta-object (MO_DataHandler_Default) that the data handler uses to retrieve configuration properties. Then follow any additional configuration steps required by the data handler you are using.

You can optionally specify a data handler using the MimeType TLO attribute. For further information, see Table 5 on page 20.

For further information on configuring the data handler, see "Connector-specific configuration properties" on page 55

Configure collaborations

To enable collaborations for request or event processing, see the following documentation:

- *IBM Websphere InterChange Server Collaboration Development Guide*
- *IBM Websphere InterChange Server Map Development Guide*

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 13.
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 the connector

Important: As noted earlier in this chapter, the connector, business objects, the 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 12. 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*.

Stopping the connector

The way to stop a connector depends on the way that the connector was started, as follows:

- If you started the connector from the command line, with its connector startup script:
  - On Windows systems, invoking the startup script creates a separate "console" window for the connector. In this window, type "Q" and press Enter to stop the connector.
  - On UNIX-based systems, connectors run in the background so they have no separate window. Instead, run the following command to stop the connector:

    `connector_manager_`*`connName`* `-stop`

    where *connName* is the name of the connector.
- 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 stops when the Windows system shuts down.

Chapter 3. Business object requirements

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

Business object meta-data

The connector for HTTP 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 http , your modifications must conform to the rules the connector is designed to follow, or the connector cannot process new or modified business objects correctly.

Connector business object structure

The connector can process the following business object:

- **TLOs** A top-level business object (TLO) contains a request business object and, optionally, response and fault business objects. These child objects contain content data and, optionally, Protocol Config MOs. They are also data-handler specific objects; for example, if you are using the XML data handler, the request child would be a business object that is comprehensible to the XML data handler. 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*.

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 shows the business object hierarchy for synchronous event processing. Request and response objects are required, fault objects are optional.

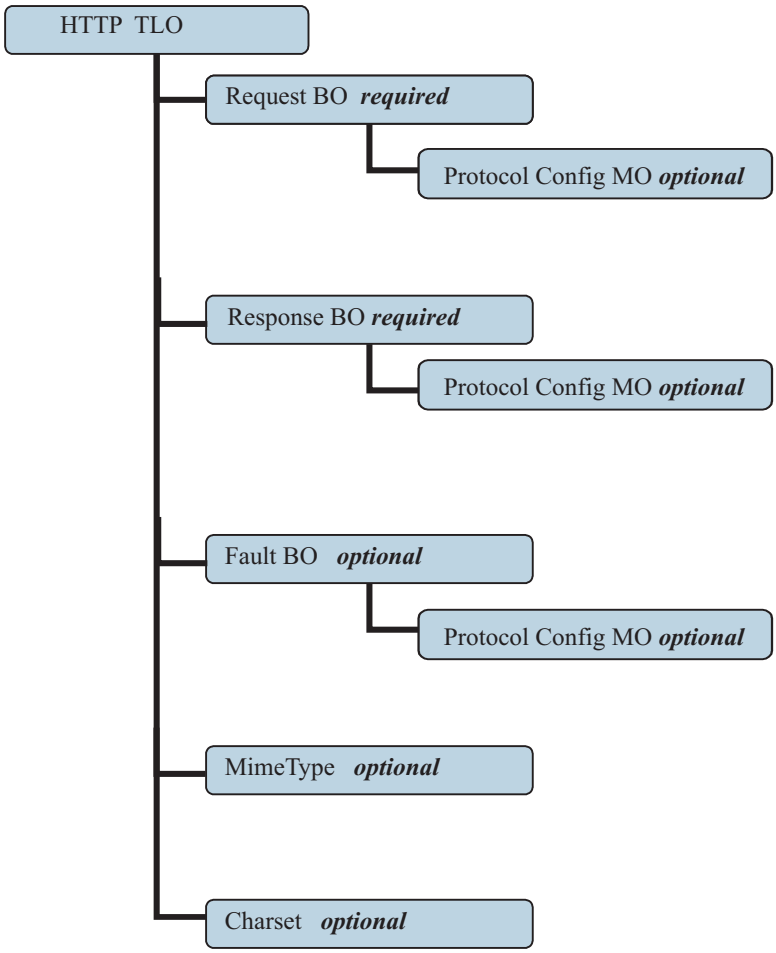


Figure 4. Business object hierarchy for synchronous event processing

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.

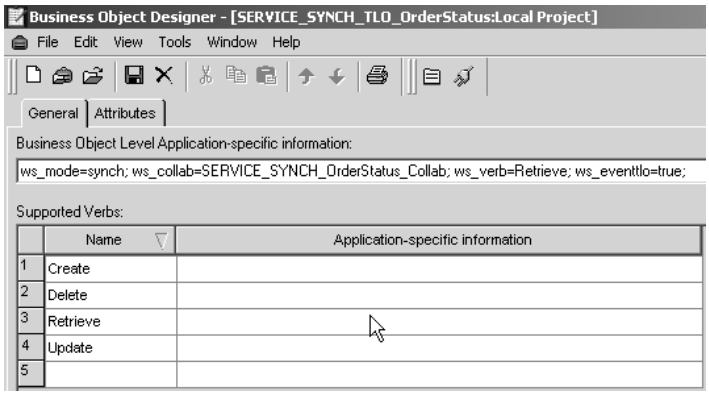


Figure 5. Top-level business object for synchronous event processing

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 enabled for event processing.
`ws_collab=`*`collabname`*	This ASI tells the connector which collaboration to invoke. Its value is the name of the collaboration. 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.

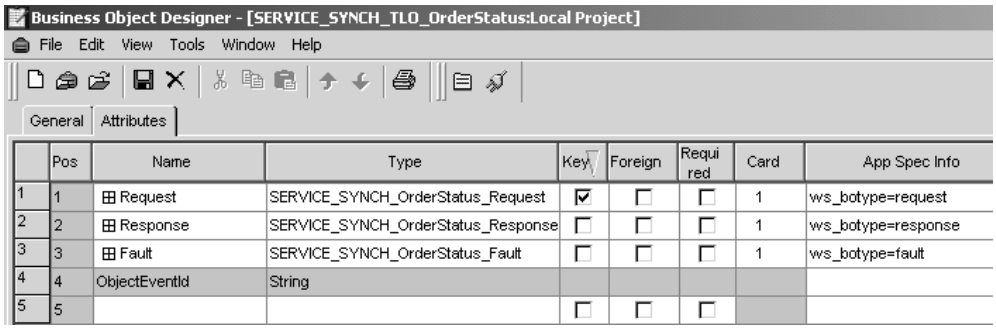


Figure 6. TLO attributes for synchronous event processing

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.
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 an HTTP service request. The connector uses its ASI to determine whether this TLO attribute is of type 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 an HTTP service. The connector uses this ASI to determine whether this TLO attribute is of type 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 Fault BO.If

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. The object-level ASI for a request business object for synchronous event processing is described in Table 6.. 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_http=*HTTPCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. The ASI designates the HTTP or HTTPS protocol listener. Both the ASI and the Protocol Config MO are optional. For further information, see "Protocol Config MO" on page 23. Note: The data handler that you configure for business object transformations should be capable of reading any ASI that begins with cw_mo as metadata and not as part of the busisenss data to be converted. The XML data handler has the capability to detect cw_mo metadata, ignoring the attributes that such values point to.
ws_tloname=*tloname*	This ASI specifies the name of the 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_http=*HTTPCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. The ASI designates the HTTP or HTTPS protocol listener. Both the ASI and the Protocol Config MO are optional. For further information, see "Protocol Config MO" on page 23. Note: The data handler that you configure for business object transformations should be capable of reading any ASI that begins with cw_mo as metadata and not as part of the busisenss data to be converted. The XML data handler has the capability to detect cw_mo metadata, ignoring the attributes that such values point to.

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_http=*HTTPCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. The ASI designates the HTTP or HTTPS protocol listener. Both the ASI and the Protocol Config MO are optional. For further information, see "Protocol Config MO." Note: The data handler that you configure for business object transformations should be capable of reading any ASI that begins with cw_mo as metadata and not as part of the busisenss data to be converted. The XML data handler has the capability to detect cw_mo metadata, ignoring the attributes that such values point to.

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

Protocol Config MO

The Protocol Config 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_http=*HTTPProtocolListenerConfigMOAttribute*

During event processing, the connector uses protocol listeners (HTTP or HTTPS) to retrieve events from the transport. These events are messages from internal or external clients requesting service from collaborations. 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 message. The connector sets the value of these attributes in the business object using inbound message content.

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

Table 9. 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/xml; 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 25

For further information on protocol listeners, see "Protocol listeners" on page 42.(For information describing the Protocol Config MO for request processing, see "Synchronous request processing TLOs" on page 28).

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 10summarizes the application-specific information for these attributes.

Table 10. 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.

If the given custom property ASI (the ws_prop_name) is invalid and there is no logical way to process this header, 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 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 an HTTP or HTTPS protocol listener processes an HTTP 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.

Asynchronous event processing TLOs

Figure 7 on page 26 shows the business object hierarchy for asynchronous event processing. A request object only is required.

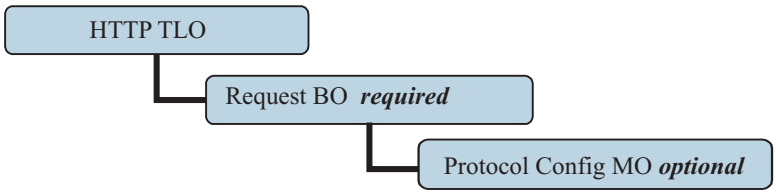


Figure 7. Business object hierarchy for asynchronous event processing

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

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 8 shows the object-level ASI for SERVICE_ASYNCH_TLO_Order, a sample TLO for asynchronous event processing.

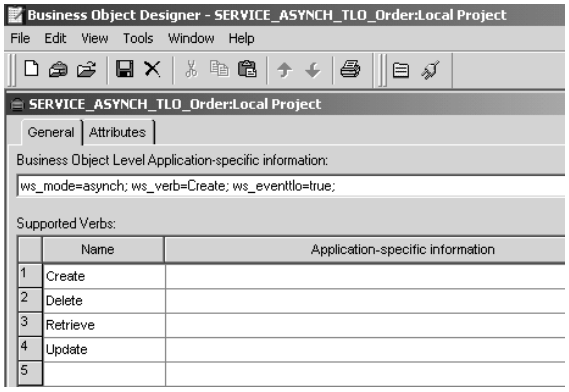


Figure 8. Top-level business object for asynchronous event processing

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

Table 11. 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.
`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 8, the verb is `Create`.

Table 11. Asynchronous event processing TLO object ASI (continued)

Object-level ASI	Description
`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 collaborations subscribing to such BO-verb combinations.

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 9 shows the request attribute of SERVICE_ASYNCH_TLO_Order, a sample TLO, and the attribute's ASI.

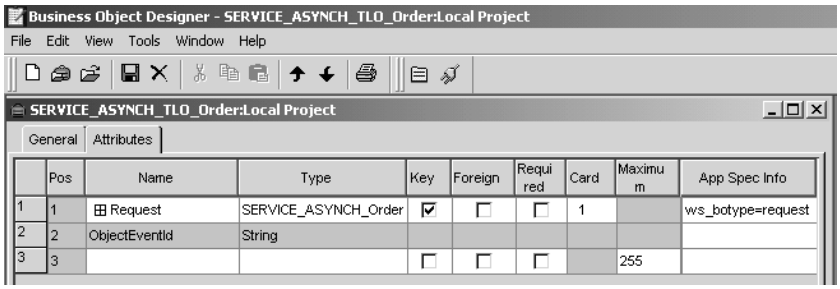


Figure 9. TLO attribute for asynchronous event processing

Table 12 summarizes the attribute-level ASI for the request attribute of an asynchronous event processing TLO.

Table 12. Asynchronous event processing TLO attribute ASI

TLO attribute	Attribute-level ASI	Description
Request	`ws_botype=request`	This attribute corresponds to a request. The connector uses its ASI to determine whether this TLO attribute is of type 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 13.

Table 13. Asynchronous event processing: object-level ASI for Request business objects

Object-level ASI	Description
`cw_mo_http=`*HTTPCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. The ASI designates the HTTP or HTTPS protocol listener. Both the ASI and the Protocol Config MO are optional. For further information, see “Protocol Config MO” on page 23. Note: The data handler that you configure for business object transformations should be capable of reading any ASI that begins with `cw_mo` as metadata and not as part of the busisenss data to be converted. The XML data handler has the capability to detect `cw_mo` metadata, ignoring the attributes that such values point to.
`ws_tloname=`*tloname*	This ASI specifies the name of the 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.

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 10 shows the TLO business object hierarchy for synchronous request processing. Request, response, and handler 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.

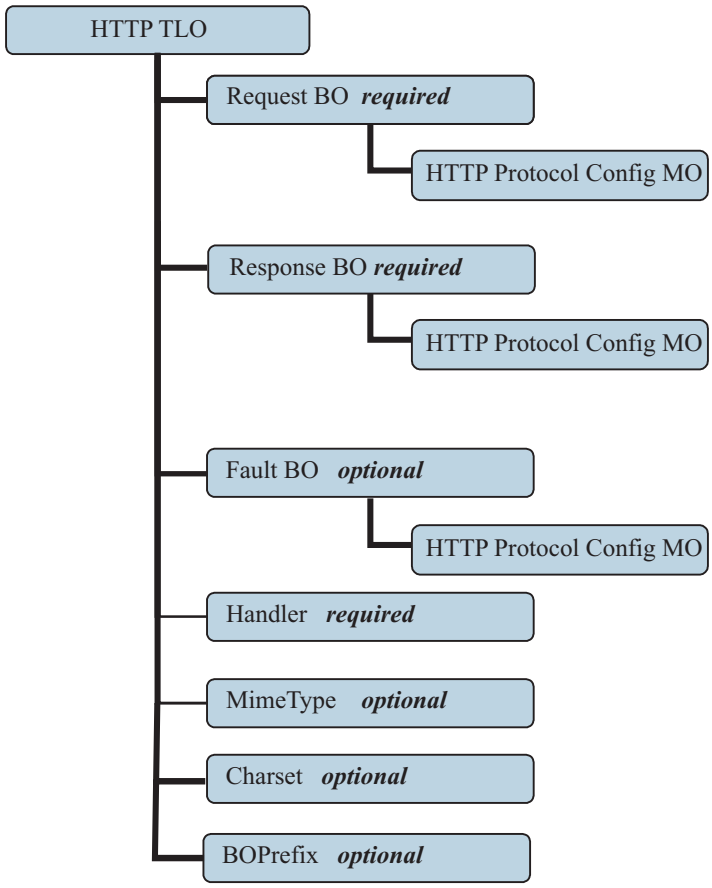


Figure 10. Business object hierarchy for synchronous request processing

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 11 shows CLIENT_SYNCH_TLO_OrderStatus, a sample TLO for synchronous request processing.
Table 14 describes the object-level ASI for a synchronous request processing TLO.

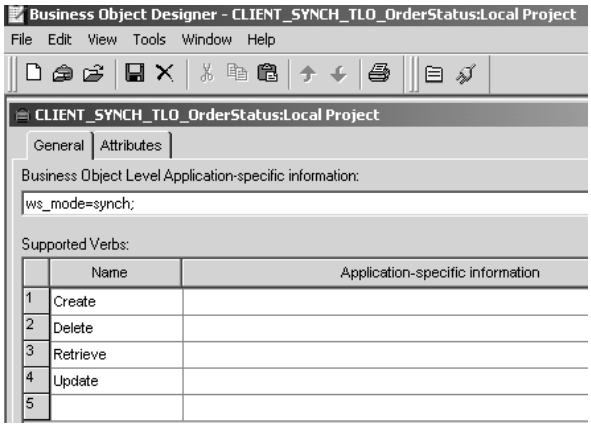


Figure 11. Top-level business object for synchronous request processing

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 14. 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 HTTP 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

Table 15 describes the attributes and ASI for synchronous request processing TLOs.

Table 15. 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 request and is for request processing only. It takes the value`http`, which designates the HTTP- HTTPS protocol handler. The default is `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 request business object. The connector uses this attribute ASI to determine whether this TLO attribute is of type 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.

Table 15. Request processing TLO attributes (continued)

TLO attribute	Attribute-level ASI	Description
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 response BO. This ASI, not the attribute name, determines the attribute type.

Fault	`ws_botype=fault`
	or
	`ws_botype=defaultfault`
		This attribute, optional for synchronous request processing, corresponds to a fault message returned by an HTTP service when it cannot successfully populate a response.
		
		The connector uses this ASI to determine if the attribute of TLO is of type 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.

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.

Table 16 describes the object-level ASI for a request business object for synchronous request processing.

Table 16. Synchronous request processing: object-level ASI for request business objects

Object-level ASI	Description
`cw_mo_http=`*`HTTPCfgMO`*	The value of this optional ASI must match the name of the attribute that corresponds to the Protocol Config MO. This Protocol Config MO specifies the destination for the HTTP-HTTPS protocol handler. This ASI is used by the HTTP-HTTPS Protocol Handler. Note that the TLO request attribute must have an HTTP Protocol Config MO for request processing. For further information, see "HTTP Protocol Config MO for request processing" on page 33. Note: The data handler that you configure for business object transformations should be capable of reading any ASI that begins with `cw_mo` as metadata and not as part of the busisenss data to be converted.The XML data handler has the capability to detect `cw_mo` metadata, ignoring the attributes that such values point to.

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 17.

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

Object-level ASI	Description
cw_mo_http=*HTTPCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. This Protocol Config MO, optional for a response business object, specifies the headers in the response message for the HTTP(s) protocol handler. For further information, see "HTTP Protocol Config MO for request processing" on page 33. Note: The data handler that you configure for business object transformations should be capable of reading any ASI that begins with cw_mo as metadata and not as part of the busisenss data to be converted. The XML data handler has the capability to detect cw_mo metadata, ignoring the attributes that such values point to.

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

Object-level ASI	Description
cw_mo_http=*HTTPCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. This Protocol Config MO, optional for a fault business object, specifies the headers in the response message for the HTTP-HTTPS protocol handler. For further information, see "Protocol Config MO" on page 23. Note: The data handler that you configure for business object transformations should be capable of reading any ASI that begins with cw_mo as metadata and not as part of the busisenss data to be converted. The XML data handler has the capability to detect cw_mo metadata, ignoring the attributes that such values point to.

HTTP Protocol Config MO for request processing

During request processing, the HTTP-HTTPS protocol handlers use the HTTP Protocol Config MO to determine the destination of the target HTTP service. This Protocol Config MO is required for a request business object. The HTTP-HTTPS protocol handlers support HTTP 1.0 POST request only. As shown in Table 19 the sole required attribute (Destination) is the full URL of the target HTTP service. The optional authorization attributes are described in the sections below.

Table 19. HTTP Protocol Config MO Attributes for Request Processing

Attribute	Required	Type	Description
Destination	Yes	`String`	The destination URL of the target HTTP service. The HTTP-HTTPS protocol handler uses this attribute to determine the destination of the HTTP 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/xml; charset=ISO-8859-4`.
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 35
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 35
One or more HTTP headers	No	String	This attribute allows the handler to pass or retrieve the value for the specified HTTP header.
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 34.

The HTTP Protocol Config MO attributes are described in:

- “User-defined properties for request processing”
- “Message transformation maps” on page 34
- “HTTP credential propagation for request processing” on page 35

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`. The application-specific information of the attribute can contain the name-value pair defining the name of the protocol message property to which the attribute maps.

Table 20 summarizes the application-specific information for these attributes.

Table 20. 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.

If the given custom property ASI (the ws_prop_name) is invalid and there is no logical way to process this header, 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 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, 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, the connector uses the rules in that object to transform a message.

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 HTTP-HTTPS 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 "HTTP-HTTPS protocol handler processing" on page 49.

Each instance of a TransformationRule business object must have attributes specified as shown in Table 21.

Table 21. 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 "HTTP-HTTPS protocol handler processing" on page 49. Note that if the 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.

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 HTTP service. The 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 12 shows the business object hierarchy for asynchronous request processing. A request and handler object are required. The request object contains a Protocol Config MO for the HTTP-HTTPS protocol handler. These are described in the sections below.

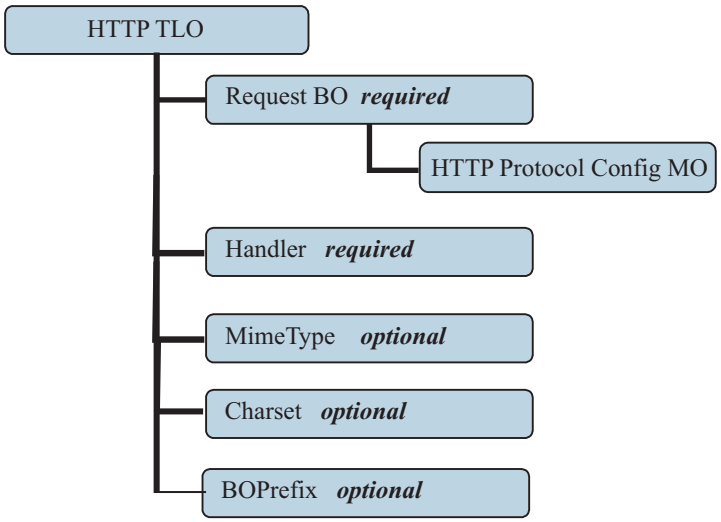


Figure 12. Business object hierarchy for asynchronous request processing

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

Object-level ASI for asynchronous event processing TLOs

Figure 13 shows CLIENT_ASYNCH_Order_TLO, a sample TLO for asynchronous request processing.

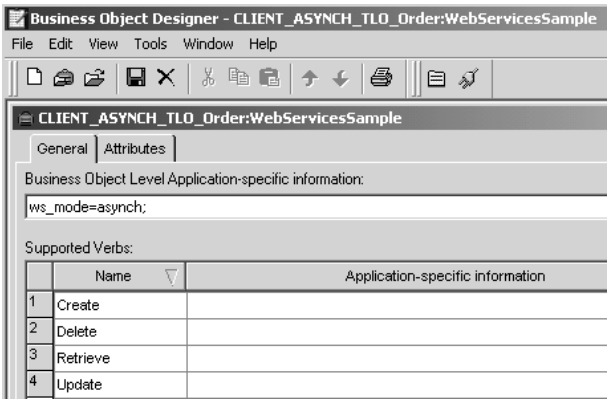


Figure 13. Top-level business object for asynchronous request processing

Table 22 below describes the object-level ASI for an asynchronous request processing TLO.

Table 22. 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

Table 23 summarizes the attribute-level ASI for the request attribute of an asynchronous request processing TLO.

Table 23. 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 only.
BOPrefix	None	The value of this attribute is passed to the data handler.
Handler	None	This attribute specifies the protocol handler to use to process the request and is for request processing only. It takes the value `http`, which designates the HTTP-HTTPS protocol handler to process the request. The default is `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 an HTTP service request business object. The connector uses this attribute ASI to determine whether this TLO attribute is of type 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 24.

Table 24. Asynchronous request processing: object-level ASI for Request business objects

Object-level ASI	Description
cw_mo_http=*HTTPCfgMO*	The value of this ASI must match the name of the attribute that corresponds to the Protocol Config MO. This Protocol Config MO specifies the destination for the HTTP-HTTPS protocol handler. This ASI is used by the HTTP-HTTPS Protocol Handler. Note that the TLO request attribute must have an HTTP Protocol Config MO for request processing. For further information, see "HTTP Protocol Config MO for request processing" on page 33. Note: The data handler that you configure for business object transformations should be capable of reading any ASI that begins with cw_mo as metadata and not as part of the busisenss data to be converted. The XML data handler has the capability to detect cw_mo metadata, ignoring the attributes that such values point to.

Protocol Config MO for asynchronous request processing

During request processing, the HTTP-HTTPS protocol handler uses the HTTP Protocol Config MO to determine the destination of the target HTTP 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 33.

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 Appendix B, "Connector Configurator," on page 89.

Chapter 4. HTTP connector

This chapter describes the HTTP connector and how to configure it.

All WebSphere business integration connectors operate with an integration broker. The HTTP 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 Guide*.

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:

- Process calls from HTTP clients (event processing)
- Process a request by a collaboration that invokes an HTTP service (request processing)

Event processing overview

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

- HTTP protocol listener

- HTTPS protocol listener

The connector uses these components to listen on the transport for calls from clients to collaborations.

When requests from clients arrive, the listener converts the 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 response message. The listener then transports the response message to the client. Note that event sequencing is not a requirement for this connector; the connector may deliver the events in any order.

The HTTP connector utilizes the configured data handler to convert incoming request messages into business objects. To aid the data handler in determining which business object to resolve for the incoming 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 is comprised of supported TLOs only. Supported TLO business objects are those that have object-level ASI `ws_eventtlo=true`.

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`)

The connector inspects the request business object returned by the data handler. It uses `ws_tloname` ASI of this business object to extract the name of the parent TLO. This TLO will be instantiated and the request business object will be set in the TLO. Finally, this constructed TLO will be used to invoke the collaboration.

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

Request processing overview

On behalf of a collaboration, the connector can invoke HTTP services over HTTP(S). This request processing functionality is supported by a protocol handler framework. The protocol handler framework is a configurable run-time module that consists of the HTTP-HTTPS protocol handler, which is discussed in detail later in this chapter.

Upon receipt of a collaboration request business object, which is always set in a TLO, the protocol handler framework loads the protocol handler. The protocol handler manages transport-level details required for invoking the HTTP service and (optionally) securing a response, performing three main tasks: converting a collaboration request business object into a request message, invoking the HTTP service with the request message, and, if in request/response (synchronous) mode, converting the response message into a business object and returning that object to the collaboration.

The HTTP connector is always called from a collaboration using TLOs. The connector determines the request business object from the TLO, and invokes the

data handler with this business object. The data handler returns a request message which is sent on by the connector to the HTTP service.

For synchronous execution, the connector utilizes the data handler to convert response and fault messages into response and fault business objects. To aid the data handler in determining which business object to resolve for these 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. 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.

HTTP(S) services

HTTP 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 an HTTP service. The HTTP server dispatches the message containing the invocation and closes the connection.

The connector's HTTP and HTTPS protocol listeners make use of the HTTP client-server and the Request/Response models when handling client requests to a collaboration. However, the HTTP listener is not intended to function as an HTTP server— proxy, intermediary, or otherwise. Rather the 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 Adapter," on page 1.

Synchronous HTTP(S) service

From the perspective of connector processing, a synchronous HTTP service is one that follows a Request/Response path. If the HTTP or HTTPS protocol listener successfully processes an HTTP request message, the body will contain the 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 HTTP(S) service

From the perspective of connector processing, an asynchronous HTTP service is one that follows a request-only path. If the HTTP or HTTPS protocol listener successfully receives and processes a request-only 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 32. If a fault occurs, an HTTP status code of `500` is generated. There is no response, although a fault body may be returned.

Event processing

During event processing, the connector uses protocol listeners and the configured data handler(s) to convert request messages from HTTP service clients to business objects that can be manipulated by collaborations. Protocol listeners play a crucial role in event processing.

Protocol listeners

HTTP requests may come over HTTP or HTTPS transports. The listener monitors the arrival of such requests on its transport channel. There are two protocol listeners and corresponding channels:

- HTTP protocol listener
- HTTPS protocol listener

Each of these consists of a thread that listens on its transport. When it receives a 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 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 64 and“Connector-specific configuration properties” on page 55.

HTTP and HTTPS protocol listener processing

The HTTP(S) protocol listener consists of a thread that continuously listens for HTTP(S) requests from 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 14 illustrates HTTP protocol listener processing for a synchronous operation.

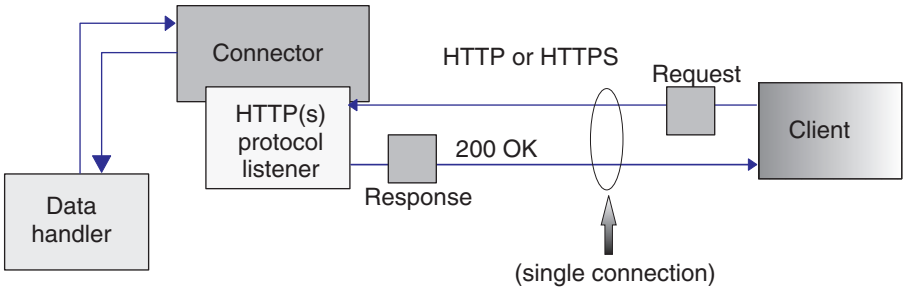


Figure 14. HTTP protocol listener: synchronous event processing

Figure 15 shows HTTP protocol listener processing for an asynchronous operation.

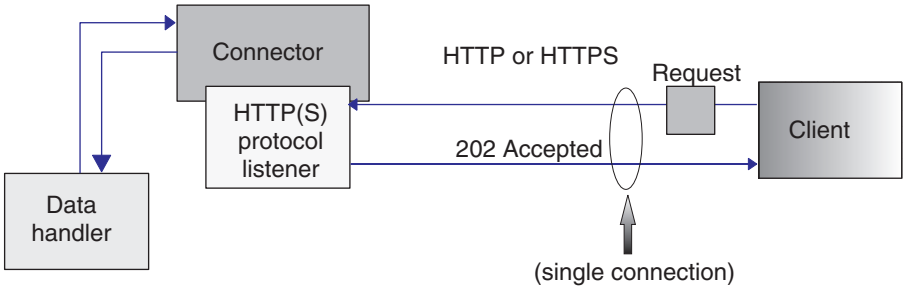


Figure 15. HTTP protocol listener: asynchronous event processing

When a client initiates a HTTP or HTTPS request, it posts a request message to the HTTP or 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 the 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 25 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 25. 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			

Table 25. HTTP(s) protocol listener processing rules for inbound message (continued)

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 Table 25 on page 43:

- 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 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 all previous steps fail to determine the MimeType, the value of ContentType will be used as the MimeType to invoke the data handler and convert the request message into a 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 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 response business object.

Table 26 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 26. 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
3	The request message Charset, but only if the request and response ContentType match.	Use ContentType value as the MimeType		
4	If the ContentType is `text/*`, default to ISO-8859-1. Otherwise, charset will not be used.			

As shown in Table 26,

- 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, the listener checks if Charset is specified in the TLO.
  3. If there is no Charset specified in the 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. 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 24.

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 data handler to convert the response business object returned by the collaboration into a response message.

The listener delivers the response message to the client and includes a `200 OK` HTTP status code. If the collaboration returns a fault business object, it is converted to a fault message. This fault message is delivered to the 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 HTTP protocol listener processing features

The 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.

HTTPS listener processing using secure sockets

HTTPS protocol listener processing is the same as that described in the HTTP protocol listener processing section except that HTTPS uses secure sockets. For further information, see "SSL" on page 53.

Event persistence and delivery

Event persistence is protocol contingent:

- **HTTP protocol listener** no persistence and therefore no guaranteed delivery
- **HTTPS protocol listener** no persistence and therefore no guaranteed delivery

Event sequencing

The connector may deliver events in any sequence.

Event triggering

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

- **HTTP protocol listener** Listening occurs over a ServerSocket for HTTP connection requests
- **HTTPS protocol listener** Listening occurs over a secure ServerSocket layer for HTTPS connection requests

Note: The 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 55.

Event status

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

- **HTTP protocol listener** HTTP is inherently non-persistent and synchronous in nature. Accordingly, event status is not maintained.
- **HTTPS protocol listener** HTTP is inherently non-persistent and synchronous in nature. Accordingly, event status is not maintained.

Event retrieval

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

- **HTTP protocol listener** Events are retrieved by extracting HTTP requests from the socket.
- **HTTPS protocol listener** Events are retrieved by extracting HTTP requests from the socket.

Event archiving

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

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

Event recovery

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

- **HTTP protocol listener** Because of the non-persistent nature of the transport, event recovery is not performed.
- **HTTPS protocol listener** Because of the non-persistent nature of the transport, event recovery is not performed.

Request processing

You use the request processing capability of the connector to enable a collaboration to invoke an HTTP service. You must 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— request, and optionally response and fault business objects— are contained by the TLO issued by a collaboration that is configured to use HTTP 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. The protocol handler uses this information to invoke an instance of the data handler, convert the request business object to a request message, and invoke the target HTTP service. If the mode is synchronous, the protocol handler again invokes the data handler to convert the response message into a response business object and returns this to the collaboration.

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

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

Protocol handlers play a key role in request processing.

Protocol handling

A collaboration can invoke an HTTP service over HTTP or HTTPS transports. The connector has one protocol handler and corresponding channel: an HTTP-HTTPS protocol handler for invoking HTTP and HTTPS services

The protocol handler framework manages the protocol handler, loading it 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 “HTTP-HTTPS protocol handler processing” on page 49), the protocol handler invokes a data handler to convert the request business object into a request message. The protocol handler packages the request message into the transport—HTTP(S)— message.

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 HTTP 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.

HTTP-HTTPS protocol handler processing

The HTTP-HTTPS protocol handler performs as described in "Protocol handling" on page 48 with exceptions noted in this section. Figure 16 shows the HTTP-HTTPS protocol handler for a synchronous operation.

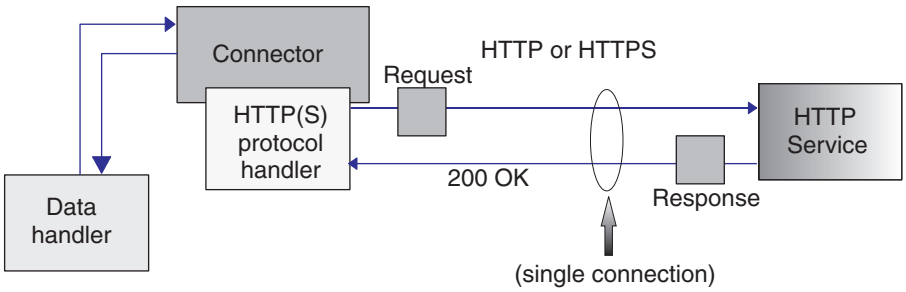


Figure 16. HTTP-HTTPS protocol handler: synchronous request processing

Figure 17 shows the HTTP-HTTPS protocol handler for an asynchronous request process

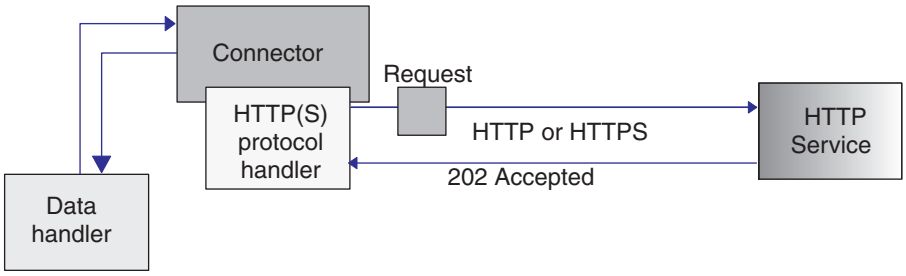


Figure 17. HTTP-HTTPS protocol handler: asynchronous request processing

Note: This section describes HTTP protocol handling only.

The HTTP-HTTPS protocol handler uses the object-level ASI (`cw_mo_http`) of the request business object to determine the Protocol Config MO. The HTTP-HTTPS protocol handler determines the URL of the target HTTP 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 33.

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

Table 27 summarizes the order of precedence of rules used by the HTTP-HTTPS protocol handler to determine the Charset, MimeType, ContentType, and Content-Type header for outgoing request messages.

Table 27. HTTP-HTTPS protocol handler processing rules for outbound messages

Order of Precedence	Charset	MimeType	ContentType	Content-Type 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		
3	If the ContentType is `text/*`, default to ISO-8859-1. Otherwise, charset will not be used.			

As shown in Table 27:

- The 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 28. 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 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 HTTP-HTTPS protocol handler determines the ContentType for the request message according to these rules:
  - 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.
- The HTTP-HTTPS protocol handler determines the Content-Type header for the request message according to these rules:
  - If the Content-Type header is specified in the request business object Protocol Config MO, its value is set on the outgoing message.

Table 29 summarizes the order of precedence for rules used by the handler when determining the Charset, MimeType, ContentType, and Content-Type header for response messages.

Table 29. HTTP(s) protocol handler processing rules for inbound synchronous response message

Order of Precedence	Charset	MimeType	ContentType	Content-Type 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 29:

- 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 MessageTransformationMap for the given request, 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.
  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 33.

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.

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 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 HTTPS protocol listeners plugged into the connector and by the HTTP-HTTPS protocol handler for each connector instance. For further information on HTTPS/SSL setup, see Appendix D, “Configuring HTTPS/SSL,” on page 115.

SSL and the HTTPS protocol listener

To use the 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 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 clients so that they can authenticate the connector.
- **TrustStore** If you want the HTTPS protocol listener to authenticate 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 clients. Digital certificates used by your 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 D, "Configuring HTTPS/SSL," on page 115.

SSL and the HTTP-HTTPS protocol handler

If you are using SSL with the 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 HTTP provider:

- **SSLVersion** Make sure that the SSLVersion you want to use is supported by your provider and by JSSE.
- **TrustStore** Because the 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 provider. Digital certificates used by your 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 HTTP service provider requires client authentication, you must set up a keystore. The 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 HTTP service provider for authentication.

For further information on HTTPS/SSL setup, see Appendix D, "Configuring HTTPS/SSL," on page 115.

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 71 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 30 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.

Table 30. Connector-specific configuration properties

Name	Possible values	Default value	Required
DataHandlerMetaObjectName	*Data handler meta-object name*	`MO_DataHandler_ Default`	Yes

Table 30. Connector-specific configuration properties (continued)

Name	Possible values	Default value	Required
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
++Handler1	*This is a hierarchical property. For information on its sub-properties, see "Handler1" on page 57.*		Yes
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
+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	`http` or `https`		Yes
+++ListenerSpecific	*Properties unique to or required by the listener* See "ListenerSpecific" on page 58.		
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

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.

Handler1: The name of an 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 31 below shows the sub-properties for the HTTP-HTTPS protocol handler. The + character indicates the entry's position in the property hierarchy.

Table 31. HTTP-HTTPS protocol handler configuration properties

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

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 42.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): HTTPListener1, HTTPSListener1.

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

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

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 32 summarizes the HTTP-HTTPS listener specific properties. The + character indicates the entry's position in the property hierarchy.

Table 32. HTTP and HTTPS protocol listener-specific configuration properties

Name	Possible values	Default value	Required
+++HTTPListener1	*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	http if HTTP protocol listener https if HTTPS protocol listener **Note:** If you do not specify a value for this property, the connector will not initialize this protocol listener.		Yes
++++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 HTTP and* 443 *for 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 HTTP listener 443 for HTTPS listener	No
++++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

Table 32. HTTP and HTTPS protocol listener-specific configuration properties (continued)

Name	Possible values	Default value	Required
++++RequestWaitTimeout	*The time interval in milli-seconds that the listener thread will block on the host and port while waiting for requests to arrive. If it receives a 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 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->HTTPListener1 *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 "HTTP and HTTPS protocol listener processing" on page 42.	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
++++++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

Table 32. HTTP and HTTPS protocol listener-specific configuration properties (continued)

Name	Possible values	Default value	Required
++++++++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 ignores 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 18 shows the properties as displayed in Connector Configurator.

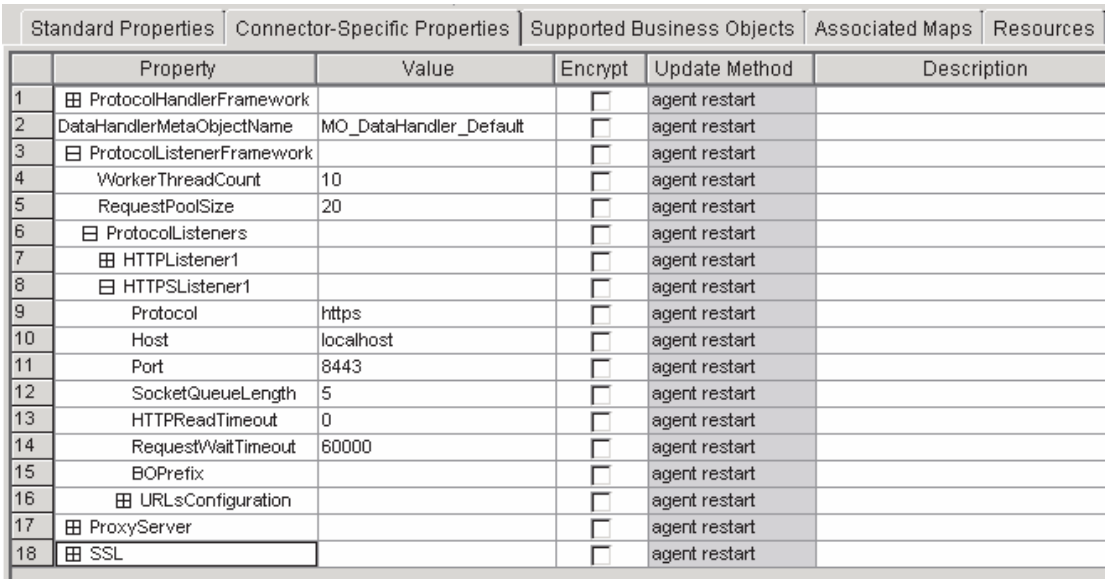
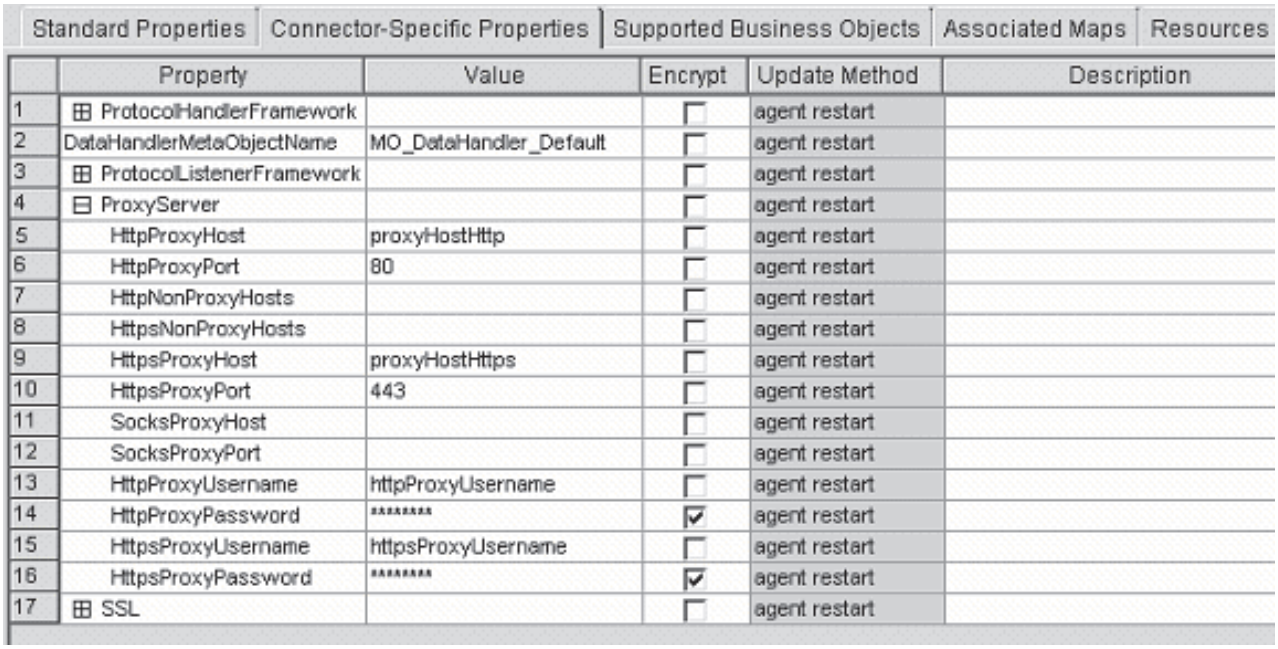


Figure 18. HTTP(S) protocol listener properties

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 HTTP-HTTPS protocol handlers.

Figure 19 shows the ProxyServer properties as displayed in Connector Configurator.



Standard Properties | Connector-Specific Properties | Supported Business Objects | Associated Maps | Resources

	Property	Value	Encrypt	Update Method	Description
1	⊞ ProtocolHandlerFramework		☐	agent restart	
2	DataHandlerMetaObjectName	MO_DataHandler_Default	☐	agent restart	
3	⊞ ProtocolListenerFramework		☐	agent restart	
4	⊟ ProxyServer		☐	agent restart	
5	HttpProxyHost	proxyHostHttp	☐	agent restart	
6	HttpProxyPort	80	☐	agent restart	
7	HttpNonProxyHosts		☐	agent restart	
8	HttpsNonProxyHosts		☐	agent restart	
9	HttpsProxyHost	proxyHostHttps	☐	agent restart	
10	HttpsProxyPort	443	☐	agent restart	
11	SocksProxyHost		☐	agent restart	
12	SocksProxyPort		☐	agent restart	
13	HttpProxyUsername	httpProxyUsername	☐	agent restart	
14	HttpProxyPassword	********	☑	agent restart	
15	HttpsProxyUsername	httpsProxyUsername	☐	agent restart	
16	HttpsProxyPassword	********	☑	agent restart	
17	⊞ SSL		☐	agent restart	

Figure 19. 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 request is an HTTP URL and you specify ProxyServer ->HttpProxyUsername, the 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 request is an HTTPS URL and you specify ProxyServer ->HttpsProxyUsername, the 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.

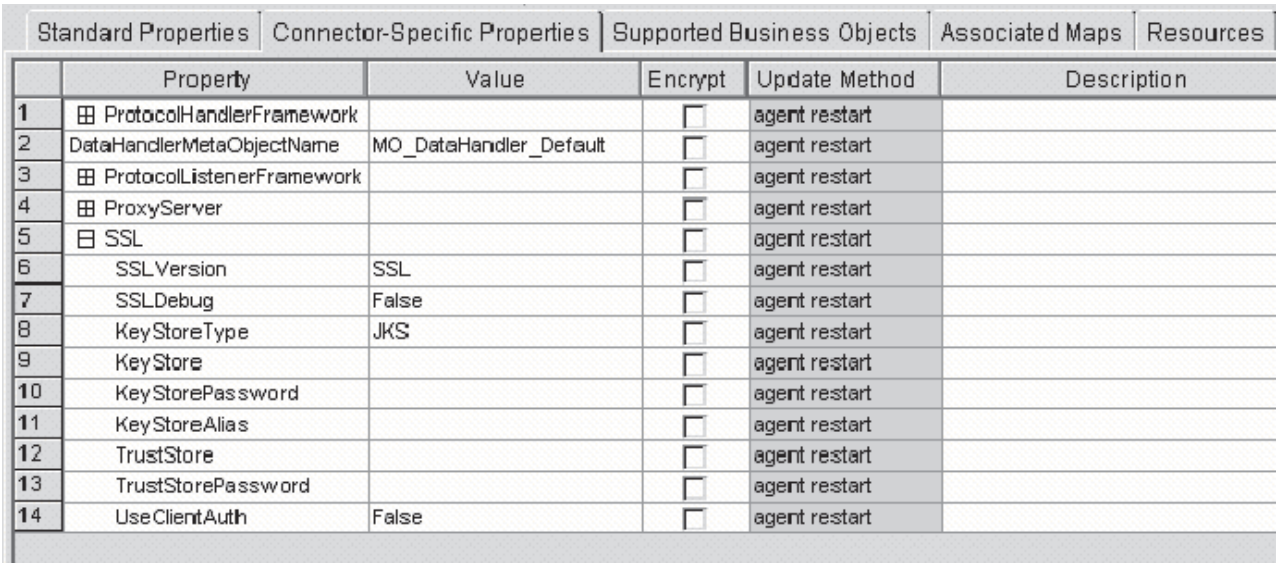
Default = none

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 20 shows the SSL properties as displayed in Connector Configurator.



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

Figure 20. 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. HTTPS listeners use this private key from the KeyStore. Also, the HTTP-HTTPS protocol handler uses this alias from the KeyStore when invoking HTTPS 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`, HTTPS listeners use client authentication.

Default = `false`

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 (`http` or `https`), 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 HTTP or HTTPS listener, be sure to specify different Port and Host properties 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 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 55.

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 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.
- 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 request business object during event notification. Trace the business object returned by the data handler.
Level 4	Trace the transport headers associated with:

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

Chapter 5. 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 HTTP or HTTPS protocol listener cannot bind to the port specified in connector properties. Solution: check the ports specified for all of the HTTP and 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 `http` or `https`. 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 `http` and `https`. Solution: This is not an error condition. However, if you want connector to use this handler, specify a valid Protocol property value.

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 an HTTP service. It occurs because your target HTTP service sent an incorrect HTTP response. Solution: make sure your target HTTP service 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 HTTP service. It occurs because you have specified an incorrect end point address for the HTTP service. Solution: make sure you have specified the correct address for the HTTP 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 HTTPConnector` .	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 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. Make sure this collaboration exists and is running.

Problem	Potential solution / explanation
Failed to transform a request into a request business object. 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: make sure you have specified `ws_verb` object-level ASI for this TLO. Specify the verb as the value of this ASI.

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 33 on page 73 below.

Summary of standard properties

Table 33 on page 73 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 33 on page 73, 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 33. 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 33. 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 33. 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 33. 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 78.

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 90).

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 91 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 95.)

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 91.
- 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 98..
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 72.

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 HTTP 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 HTTP adapter for an HTTP sample
- illustrate how to configure the HTTP adapter for an HTTPS sample

About the tutorial

This tutorial is intended to demonstrate the asynchronous and synchronous event transmission for both the request and event processing facets of the Adapter for HTTP with both of the supported protocols: HTTP and HTTPS. In each scenario, the adapters act as:

- an HTTP client that invokes an external URL
- a proxy that listens for HTTP requests on a URL and routes them to a WebSphere ICS collaboration.

The tutorial is designed to show the basic functionality of the adapter in sample scenarios:

- **An asynchronous scenario** that illustrates an asynchronous (request-only) HTTP POST. There are two samples in this scenario—for configuration simplicity, the same HTTP adapter is used to used to listen for HTTP requests and invoke a URL as an HTTP client.
  - **A proxy that listens for HTTP requests on a URL** In this sample, the incoming request is routed to the collaboration `SERVICE_ASYNCH_Order_Collab` within WebSphere ICS. The collaboration is referred to as Asynch Order. If the adapter is properly configured, this collaboration can be invoked using either one of the protocols: HTTP or HTTPS. `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 HTTPConnector. The service port (To) is bound to SampleSiebelConnector.
  - **An HTTP client that invokes an external URL** In this sample, the HTTP client is another collaboration `CLIENT_ASYNCH_Order_Collab` within WebSphere ICS that will invoke an external URL asynchronously using the HTTP adapter. If the adapter is configured properly, this HTTP client can invoke the external URL over either one of the protocols: HTTP or HTTPS. `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 HTTPConnector.

  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) HTTP Post. There are two samples in this scenario—for configuration simplicity, the same HTTP adapter is used to listen for HTTP requests and invoke a URL as an HTTP client.
  - **A proxy that listens for HTTP requests on a URL** In this sample, the incoming request is routed to the collaboration `SERVICE_SYNCH_OrderStatus_Collab` within WebSphere ICS. The collaboration is referred to as Synch OrderStatus. If the adapter is properly configured, this collaboration can be invoked using either one of the protocols: HTTP or HTTPS. `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 HTTPConnector. The service port (To) is bound to SampleSiebelConnector.
  - **An HTTP client that invokes an external URL** In this sample, the HTTP client is another collaboration `CLIENT_SYNCH_OrderStatus_Collab` within WebSphere ICS that will invoke an external URL using the HTTP adapter. If the adapter is properly configured, this HTTP client can invoke the external URL over either one of the protocols: HTTP or HTTPS. `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 HTTPConnector.

  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 4.2.2 or later.
- You have installed the WebSphere Business Integration Adapter for HTTP in the WebSphere ICS home directory.
- You are experienced with HTTP technology.
- You are experienced with XML 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 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 `HTTPSample.jar` located in: WBI_folder\connectors\HTTP\samples\WebSphereICS\

Compile the collaboration templates

Using WebSphere Business Integration System Manager:

- **Compile All** of the Collaboration Templates that were imported from the `HTTPSample.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 HTTPConnector in Connector Configurator.
3. You must also configure HTTPConnector for the protocol you want to use with the sample:
   - If you want to use HTTP, see "Configuring for the HTTP protocol scenario" to configure the connector for HTTP.
   - If you want to use HTTPS, see "Configuring for the HTTPS protocol scenario" to configure the connector for HTTPS.

Configuring for the HTTP protocol scenario

This section shows you how to configure the connector for the HTTP sample scenario. As described in the body of this document, the connector includes an HTTP protocol listener and HTTP-HTTPS 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 HTTP protocol listener for this sample:

1. In Connector Configurator, click on **Connector-Specific Properties** for the HTTPConnector.
2. Expand the **ProtocolListenerFramework** property to display the ProtocolListeners child property.
3. Expand the **ProtocolListeners** child property to display the **HTTPListener1** child property.
4. Check the value of **HTTPListener1->Host** and **HTTPListener1->Port** properties. 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 **HTTPListener1->Host** to the machine name on which you will run the connector.

You need not configure the HTTP-HTTPS protocol handler for the sample.

Configuring for the HTTPS protocol scenario

This section shows you how to configure the connector for the HTTPS sample scenario. The connector includes an HTTPS protocol listener and HTTP-HTTPS 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 106, you should also have created and tested your keystore and truststore using your Key and Certificate management software.

Configure SSL connector-specific properties: For 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 HTTPConnector.
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 HTTPS protocol listener:

1. In Connector Configurator, click on **Connector-Specific Properties** for the HTTPConnector.
2. Expand the **ProtocolListenerFramework** property to display the **ProtocolListeners** child property.
3. Expand the **ProtocolListeners** child property to display the **HTTPSListener1** child property. Check the value of the **HTTPSListener1->Host** and **HTTPSListener1->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 **HTTPSListener1->Host** to the machine name on which you are running the connector.

You need not configure the 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 HTTP 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 httpadapter -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 httpadapter` in the command line, you would enter `httpadapter` 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 <httpadapter> (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 httpadapter -keystore c:\security\keystore
   -file c:\security\httpadapter.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\httpadapter.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.

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 107.

Add and deploy the project

1. From the Server Instance view, add the **User Project** created in "Create user project" on page 109to 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 HTTP 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 HTTPConnector.
4. HTTPconnector then finds the `XML_Order` object that is a child of the `CLIENT_ASYNCH_TLO_Order` object.
5. The Request business object is converted into an XML message using the XML data handler. HTTPconnector sends the XML message to the URL 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`. This value should be set to `http` or `https`.
6. The XML request is POSTed to the URL. As mentioned earlier, the same HTTPConnector is listening for the XML request on the same URL. The connector's protocol listener receives the XML message.
7. The connector converts the XML message into XML_Order and then creates a `SERVICE_ASYNCH_TLO_Order` object. The XML_Order object is set as a child of the SERVICE__ASYNCH_TLO_Order object.
8. HTTPConnector now asynchronously posts the SERVICE_TLO_Order object to ICS. This completes the asynchronous URL invocation.

Because this is an asynchronous invocation (request-only), no response is sent back to the HTTP client. When `SERVICE_ASYNCH_Order_Collab` receives this object, the collaboration then sends the business object to the application named`SampleSiebel`, which is running as the second instance of Test Connector. The object is displayed in the Test Connector. When Reply Success is selected from the`SampleSiebel` 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 HTTP 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\HTTP\samples\WebSphereICS\OrderStatus\CLIENT_ASYNCH_TLO_Order.bo

   The Test Connector should show that the CLIENT_ASYNCH_TLO_Order is loaded.
7. Verify the HTTP URL address:
   - To run the HTTP sample:
     a. In your Test Connector, make sure that the value of the Handler attribute for the CLIENT_ASYNCH_TLO_Order business object is set to http.
     b. Expand the Request attribute of CLIENT_ASYNCH_TLO_Order. This attribute is of type CLIENT_ASYNCH_Order business object.
     c. Expand the HTTPCfgMO attribute of XML_Order. This attribute is of type XML_Order_HTTP_CfgMO.
     d. Make sure the value of the Destination attribute of XML_Order_HTTP_CfgMO is set to http://localhost:8080/wbia/http/samples.
   - **To run the HTTPS sample**
     a. Make sure that the value of the Handler attribute for the CLIENT_ASYNCH_TLO_Order business object is set to http even though this is an HTTPS invocation.
     b. Expand the Request attribute of CLIENT_ASYNCH_TLO_Order. This attribute is of type XML_Order business object.
     c. Expand the HTTPCfgMO attribute of XML_Order. This attribute is of type XML_Order_HTTP_CfgMO.
     d. Make sure the value of the Destination attribute of XML_Order_HTTP_CfgMO is set to https://localhost:443/wbia/http/samples.
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, CustomarId 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 HTTP 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 HTTP connector.
4. The HTTP connector finds the XML_OrderStatus object, which is a request child of the CLIENT_SYNCH_TLO_OrderStatus object.
5. The HTTP connector invokes the XML data handler to convert the XML_OrderStatus business object into an XML message.
6. The XML Request is POSTed to the URL. As mentioned earlier, the same HTTPConnector is listening for the XML request on the same URL. The connector's protocol listener receives the XML message.
7. The connector's protocol listener converts the XML message into XML_OrderStatus and then creates a SERVICE_SYNCH_TLO_Order object. The XML_OrderStatus object is set as a child of the SERVICE_SYNCH_TLO_Order object.
8. The HTTP connector now synchronously posts the SERVICE_SYNCH_TLO_OrderStatus object to the SERVICE_SYNCH_OrderStatus_Collab collaboration running in WebSphere ICS. Since this is a synchronous execution, the HTTP connector remains blocked until the collaboration executes and returns the response.
9. The HTTP 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 HTTP connector remains blocked until the collaboration executes and returns the response.
10. After editing the values, and selecting Reply Success from the SampleSiebel application, the event is sent back to the SERVICE_SYNCH_OrderStatus_Collab collaboration.
11. SERVICE_SYNCH_OrderStatus_Collab receives the SERVICE_SYNCH_TLO_OrderStatus object. The collaboration then sends the business object to HTTPConnector.
12. HTTPConnector finds the XML_OrderStatus business object that is a child of the SERVICE_SYNCH_OrderStatus_TLO. This business object is converted into an XML response message by the XML data handler.
13. The XML response is sent back to the HTTP Client.
14. The HTTP client, which in this case is the HTTP connector's protocol handler, receives the response. The connector invokes the XML data handler with the response message. The XML data handler converts the response message into an XML_OrderStatus business object. HTTPConnector sets this object as the child of CLIENT_SYNCH_OrderStatus_TLO.
15. CLIENT_SYNCH_OrderStatus_TLO is returned to the CLIENT_SYNCH_OrderStatus_Collab collaboration.
16. 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 HTTP 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\HTTP\samples\WebSphereICS\OrderStatus\CLIENT_SYNCH_TLO_OrderStatus.bo

   The Test Connector should show that the CLIENT_SYNCH_TLO_OrderStatus is loaded.
7. Verify the HTTP URL address:
   - **To run the HTTP sample**:
     a. In your Test connector, make sure that the value of the Handler attribute for the CLIENT_SYNCH_TLO_OrderStatus business object is set to http.
     b. Expand the request attribute of CLIENT_SYNCH_TLO_OrderStatus. This attribute is of type XML_OrderStatus business object.
     c. Expand the HTTPCfgMO attribute of XML_OrderStatus. This attribute is of type XML_Order_HTTP_CfgMO.
     d. Make sure the value of the Destination attribute of XML_Order_HTTP_CfgMO is set to http://localhost:8080/wbia/http/samples.
   - **To run the HTTPS sample**:
     a. In your Test Connector, make sure that the value of the Handler attribute for the CLIENT_SYNCH_TLO_OrderStatus business object is set to http even though this is an https invocation.
     b. Expand the Request attribute of CLIENT_SYNCH_TLO_OrderStatus. This attribute is of type XML_OrderStatus business object
     c. Expand the HTTPCfgMO attribute of XML_OrderStatus. This attribute is of type XML_Order_HTTP_CfgMO.
     d. Make sure the value of the Destination attribute of XML_Order_HTTP_CfgMO is set to https://localhost:443/wbia/http/samples.
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. Inspect the values of the request to ensure that that the values sent across from the SampleSAPConnector are intact.
11. Populate the response attribute of this business object by selecting **LOAD BO**. Load the following file:
    - WBI_folder\connectors\HTTP\samples\WebSphereICS\SERVICE_SYNCH_TLO_OrderStatus.bo

    The Test Connector should show that the SERVICE_SYNCH_TLO_OrderStatus is loaded.
12. Select **REQUEST->REPLY->SUCCESS.**
13. An event labeled SERVICE_SYNCH_TLO_OrderStatus.Retrieve is displayed in the right panel of the Test Connector that is simulating SampleSAPConnector.

14. 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.
15. Once you have inspected the business object, close the window. Select **REQUEST->REPLY->SUCCESS**.

This completes the execution of synchronous scenario.

Appendix D. 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 HTTP 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 httpadapter -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 63.

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 httpadapter` 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 <httpadapter> (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 httpadapter -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 HTTPS protocol listener to authenticate the client, set the SSL " UseClientAuth connector configuration property to `true` . In this case, the HTTPS protocol listener expect s TrustStore to contain certificates for all trusted clients. Note that the connector uses the JSSE default mechanism to trust clients.

If you are invoking HTTPS services, the HTTP-HTTPS protocol handler requires that TrustStore trust the service. This means that TrustStore must contain the certificates of all trusted HTTP 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 httpadapter.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