

IBM WebSphere Business Integration



Adapter for TCP/IP User Guide

Adapter Version 1.04

IBM WebSphere Business Integration



Adapter for TCP/IP User Guide

Adapter Version 1.04

Note!

Before using this information and the product it supports, read the information in Appendix F, "Notices," on page 109.

15December2005

This edition of this document applies to IBM WebSphere Business Integration Adapter for TCP/IP (5724-i47), WebSphere Business Integration Adapter Framework, Version 2.6.

To send us your comments about IBM WebSphere Business Integration documentation, e-mail 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.

© **Copyright International Business Machines Corporation 2004, 2005. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this document	vii
What this document includes	vii
What this document does not include	vii
Audience	vii
Related documents.	vii
Typographic conventions	viii
New in this release.	ix
New in release 1.0.x.	ix
New in release 1.0.0.	ix
Chapter 1. Overview	1
Task roadmap	1
Terminology	1
Adapter overview	3
The TCP/IP protocol	3
Connector architecture	3
Common Event Infrastructure	6
Application Response Measurement	7
Chapter 2. Installing the connector	9
The adapter environment	9
Broker compatibility	9
Adapter platforms	9
Processing locale-dependent data	10
Installing the connector	11
Verifying the installation	11
Installed Windows file structure	11
Installed UNIX file structure.	11
Chapter 3. Business objects and meta-objects in the TCP/IP adapter	13
Processing flow	13
PIMO framework processing	14
Service call request processing	18
Event processing	24
Internal business objects	26
BIA_ContentBO	26
BIA_InputMessage	27
BIA_ApplicationMessage	27
BIA_ResponseMessage.	28
BIA_FinalMessage	29
General meta-objects	29
Configuration meta-object (BIA_Static_MO).	30
Service Registration meta-object (BIA_MO_Service)	31
Data handler meta-object (BIA_MO_DataHandler_Default)	33
PIMO framework meta-objects	33
BIA_MO_Tcpip_MapSubscriptions.	33
BIA_Map_InputMessage_to_LLPMessagesList	35
Map usage rules.	36
Chapter 4. Configuring the connector	39
Overview of Connector Configurator	39
Running connectors on UNIX	40
Starting Connector Configurator	40

Running Configurator in stand-alone mode.	40
Running Configurator from System Manager	40
Creating a connector-specific property template	41
Creating a new template	41
Creating a new configuration file	43
Creating a configuration file from a connector-specific template	44
Using an existing file	45
Completing a configuration file.	46
Setting the configuration file properties	46
Setting standard connector properties.	47
Setting connector-specific configuration properties	48
Specifying supported business object definitions	49
Associated maps.	51
Resources	52
Messaging.	52
Security	52
Setting trace/log file values	53
Data handlers	54
Saving your configuration file	54
Changing a configuration file	54
Completing the configuration	55
Using Connector Configurator in a globalized environment	55
Chapter 5. Running the connector	57
Starting the connector	57
Stopping the connector	58
Creating multiple instances of connectors on one server	59
Create a new directory	59
Create business object definitions	59
Create a connector definition	59
Create a start-up script	59
Chapter 6. Maintaining the connector	61
Error handling in the connector.	61
Tracing messages	62
Using tracing with the connector	62
Appendix A. Standard configuration properties	65
Standard connector properties overview.	65
Starting Connector Configurator	65
Configuration property values overview.	66
Standard properties quick-reference	67
Standard properties.	72
AdapterHelpName	72
AdminInQueue	73
AdminOutQueue	73
AgentConnections	73
AgentTraceLevel.	73
ApplicationName	73
BiDi.Application.	73
BiDi.Broker	74
BiDi.Metadata	74
BiDi.Transformation	74
BrokerType	74
CharacterEncoding	74
CommonEventInfrastructure.	74
CommonEventInfrastructureContextURL	75
ConcurrentEventTriggeredFlows	75
ContainerManagedEvents.	75
ControllerEventSequencing	76

ControllerStoreAndForwardMode	76
ControllerTraceLevel	77
DeliveryQueue	77
DeliveryTransport	77
DuplicateEventElimination	78
EnableOidForFlowMonitoring	78
FaultQueue	79
jms.FactoryClassName	79
jms.ListenerConcurrency	79
jms.MessageBrokerName	79
jms.NumConcurrentRequests	79
jms.Password	80
jms.TransportOptimized	80
jms.UserName	80
JvmMaxHeapSize	80
JvmMaxNativeStackSize	80
JvmMinHeapSize	80
ListenerConcurrency	81
Locale	81
LogAtInterchangeEnd	81
MaxEventCapacity	82
MessageFileName	82
MonitorQueue	82
OADAutoRestartAgent	82
OADMaxNumRetry	83
OADRetryTimeInterval	83
PollEndTime	83
PollFrequency	83
PollQuantity	84
PollStartTime	84
RepositoryDirectory	84
RequestQueue	85
ResponseQueue	85
RestartRetryCount	85
RestartRetryInterval	85
ResultsSetEnabled	85
ResultsSetSize	86
RHF2MessageDomain	86
SourceQueue	86
SynchronousRequestQueue	86
SynchronousRequestTimeout	87
SynchronousResponseQueue	87
TivoliMonitorTransactionPerformance	87
WireFormat	87
WsifSynchronousRequestTimeout	87
XMLNamespaceFormat	87

Appendix B. Connector-specific properties and required business-object properties . . 89

Connector-specific properties	89
Connector-specific configuration property descriptions	91
ServerConfiguration	91
ClientConfiguration	92
ConfigurationMetaObject	94
ServiceRegistrationMO	94
DataHandlerMimeType	94
ErrorClassName	94
DataHandlerMetaObjectName	94
Supported business objects	94
Performance tuning	95
Event processing	95
Request processing	95

Appendix C. Error handling	97
Error handler for event processing.	97
Checkpoint examples	97
Interface details	97
Error handler for request processing	98
Checkpoint examples	98
Interface details	99
Guidelines for the message handler and data handler	99
Appendix D. Common Event Infrastructure.	101
Required software.	101
Enabling Common Event Infrastructure	101
Obtaining Common Event Infrastructure adapter events	101
For more information.	102
Common Event Infrastructure event catalog definitions	102
XML format for "start adapter" metadata	102
XML format for "stop adapter" metadata	104
XML format for "timeout adapter" metadata	104
XML format for "request" or "delivery" metadata	105
Appendix E. Application Response Measurement	107
Required software.	107
Enabling Application Response Measurement	107
Transaction monitoring	107
For more information.	108
Appendix F. Notices	109
Programming interface information	111
Trademarks and service marks.	111
Index	113

About this document

The IBM^(R) WebSphere^(R) Business Integration Adapters portfolio supplies connectivity for leading e-business technologies, enterprise applications, and legacy and mainframe systems. The product set includes tools and templates for customizing, creating, and managing components for business process integration.

What this document includes

This document describes installation, connector property configuration, business object development, and troubleshooting for this IBM WebSphere Business Integration adapter.

What this document does not include

This document does not describe deployment metrics and capacity planning issues such as server load balancing, number of adapter processing threads, maximum and minimum throughputs, and tolerance thresholds.

Such issues are unique to every customer deployment and must be measured within or close to the exact environment where the adapter is to be deployed. You should contact your IBM services representative to discuss the configuration of your deployment site, and for details on planning and evaluating these kinds of metrics, given your specific configuration.

Audience

This document is for consultants, developers, and system administrators who use the adapter at customer sites.

Related documents

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

You can install related documentation from the following sites:

- For general adapter information; for using adapters with WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, WebSphere Business Integration Message Broker); and for using adapters with WebSphere Application Server, see the IBM WebSphere Business Integration Adapters information center:
<http://www.ibm.com/websphere/integration/wbiadapters/infocenter>
- For using adapters with WebSphere InterChange Server, see the IBM WebSphere InterChange Server information centers:
<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/>
- For more information about WebSphere Application Server:
<http://www.ibm.com/software/webservers/appserv/library.html>

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.

Typographic conventions

This document uses the following conventions:

<code>courier font</code>	Indicates a literal value, such as a command name, file name, information that you type, or information that the system prints on the screen.
<i>italic</i>	Indicates a new term the first time that it appears, 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, a pipe separates 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, <code>option[,...]</code> means that you can enter multiple, comma-separated options.
< >	Angle brackets surround individual elements of a name to distinguish them from each other, as in <code><server_name><connector_name>tmp.log</code> .
/ , \	In this document, backslashes (\) are used as the convention for directory paths. For UNIX ^(R) installations, substitute slashes (/) for backslashes. All product path names are relative to the directory where the connector is installed on your machine.
<i>ProductDir</i>	Represents the directory where the IBM WebSphere Business Integration Adapters product is installed. The default directory is <code>WebSphereAdapters</code> .

New in this release

New in release 1.0.x

This document describes the following updates to the Adapter for TCP/IP in releases 1.0.1, 1.0.2, 1.0.3, and 1.0.4.

- An error handler is invoked whenever an error or exception occurs. The error handler is configurable, and custom code can be written.
- Connection pooling is available to improve performance during synchronous request processing.
- Support for the BO_Verb attribute has been provided during request processing.
- Support for Base64 encoding of incoming data and decoding of outgoing data has been added.
- The adapter was migrated to Adapter Framework 2.6.
- Request processing has been enhanced:
 - Both asynchronous and synchronous processing is available.
 - Message handlers can be bypassed.
- BIA_ApplicationMessage is now part of both the Inbound and Outbound sections of a map.

New in release 1.0.0

Version 1.0.0 is the first release of the WebSphere Business Integration Adapter for TCP/IP.

Chapter 1. Overview

This chapter provides a brief overview of the TCP/IP adapter, explaining terms you need to know and describing adapter processing. It is important that you have a basic understanding of the adapter before installing, configuring, and using it. The following topics are covered:

- “Task roadmap”
- “Terminology”
- “Adapter overview” on page 3

Task roadmap

To use the adapter for TCP/IP, you must perform the tasks described in Table 1.

Table 1. Using the adapter: task roadmap

Task	Associated procedure (see...)	For more information (see...)
Installing the adapter	Chapter 2, “Installing the connector,” on page 9	<i>Installing WebSphere Business Integration Adapters</i>
Configuring business and meta-objects	Chapter 3, “Business objects and meta-objects in the TCP/IP adapter,” on page 13	<i>Business Object Development Guide</i>
Configuring the connector	Chapter 4, “Configuring the connector,” on page 39 Appendix A, “Standard configuration properties,” on page 65 Appendix B, “Connector-specific properties and required business-object properties,” on page 89	<i>Connector Development Guide</i>
Running the connector	Chapter 5, “Running the connector,” on page 57	<i>Connector Development Guide</i>
Maintaining the connector	Chapter 6, “Maintaining the connector,” on page 61	

Terminology

To understand the adapter, you must know these terms:

adapter

The component in the WebSphere business integration system that provides components to support communication between an integration broker and either an application or a technology. An adapter always includes a connector, message files, and configuration tools. It can also include an Object Discovery Agent (ODA) or a data handler.

adapter framework

The software that IBM provides to configure and run an adapter. The runtime components of the adapter framework include the Java^(TM) runtime environment, the connector framework, and the Object Discovery Agent (ODA) runtime. This connector framework includes the connector

libraries (C++ and Java) needed to develop new connectors. The ODA runtime includes the library in the Object Development Kit (ODK) needed to develop new ODAs. The configuration components include the following tools:

- Business Object Designer
- Connector Configurator
- Log Viewer
- System Manager
- Adapter Monitor
- Test Connector
- Any Object Discovery Agents (ODAs) associated with an adapter (optional)

Adapter Development Kit (ADK)

A development kit that provides some samples for adapter development, including sample connectors and Object Discovery Agents (ODAs).

connector

The component of an adapter that uses business objects to send information about an event to an integration broker (event notification) or receive information about a request from the integration broker (request processing). A connector consists of the connector framework and the connector's application- or technology-specific component.

connector framework

The component of a connector that manages interactions between a connector's application- or technology-specific component and the integration broker. This component provides all required management services and retrieves the meta-data that the connector requires from the repository. The connector framework, whose code is common to all connectors, is written in Java and includes a C++ extension to support application-specific components written in C++.

connector controller

A subcomponent of the connector framework used when the integration broker in the system is the WebSphere InterChange Server. This subcomponent interacts with collaborations, a feature of WebSphere InterChange Server. A connector controller runs within InterChange Server and initiates mapping between application-specific and generic business objects and manages collaboration subscriptions to business object definitions.

integration broker

The component in the WebSphere business integration system that integrates data among heterogeneous applications. An integration broker typically provides a variety of services that include: the ability to route data, a repository of rules that govern the integration process, connectivity to a variety of applications, and administrative capabilities that facilitate integration.

WebSphere business integration system

An enterprise solution that moves information among diverse sources to perform business exchanges and that processes and routes information among disparate applications in the enterprise environment. The business integration system consists of an integration broker and one or more adapters.

Adapter overview

This section provides a brief description of the TCP/IP protocol and describes the components of the adapter for TCP/IP.

The TCP/IP protocol

TCP/IP is a set of protocols for the transmission of any sort of data over heterogeneous physical networks. It is the basis of the Internet and many other networks large and small. Data is encapsulated in packets that the stateless IP protocol carries through the network. Control of the stream of IP packets is taken care of by TCP, the Transmission Control Protocol. It is responsible for establishing a "virtual circuit" between the sending host and the receiving host and providing for reliable transmission.

Software that implements the TCP protocol comes in two varieties: TCP clients, which initiate the connection, and TCP servers, which listen on assigned ports for initiation requests and then accept them. The structure of TCP is inherently full duplex, meaning that both TCP clients and TCP servers are capable of sending and receiving data concurrently.

Connector architecture

The WebSphere Business Integration Adapter for TCP/IP provides a way of routing data sent over raw TCP/IP connections into and out of the WebSphere business integration system. Certain industry-standard protocols for the transmission of domain-specific messages, such as the HL7 protocol in the health-care industry, are designed to be sent directly over TCP/IP connections. The TCP/IP adapter provides a robust, highly scalable way to route such messages into or out of the WebSphere business integration system, regardless of the nature of the TCP application at the other end of the connection.

Figure 1 and Figure 2 show the connector components and their relationships to the WebSphere business integration system and to the TCP/IP network to which they are connected. In Figure 1 on page 4, an inbound message coming from the network (represented by the cloud) over a TCP/IP socket is transformed and then delivered to the broker.

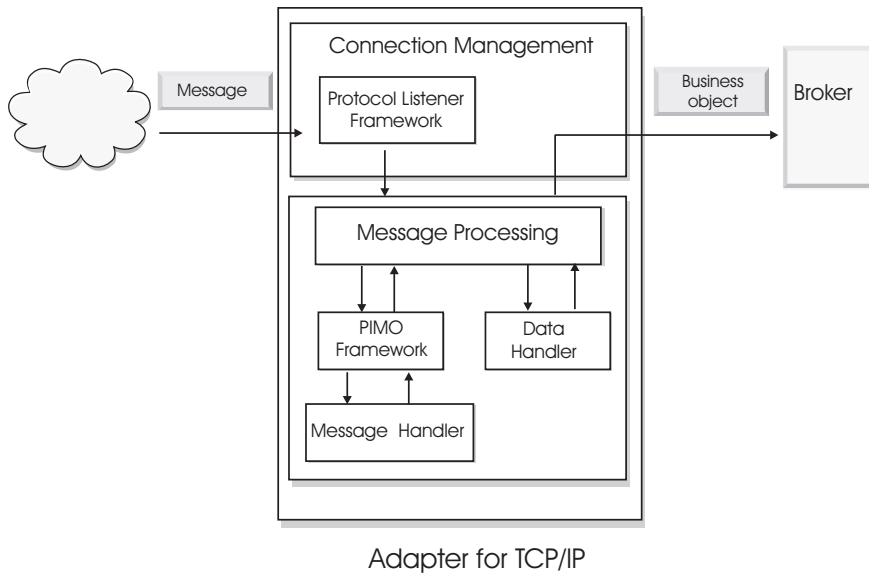


Figure 1. Architecture of the connector - Event (inbound) processing

Similarly, in Figure 2, an outbound message coming from a broker is transformed and then delivered over a TCP/IP socket to the network.

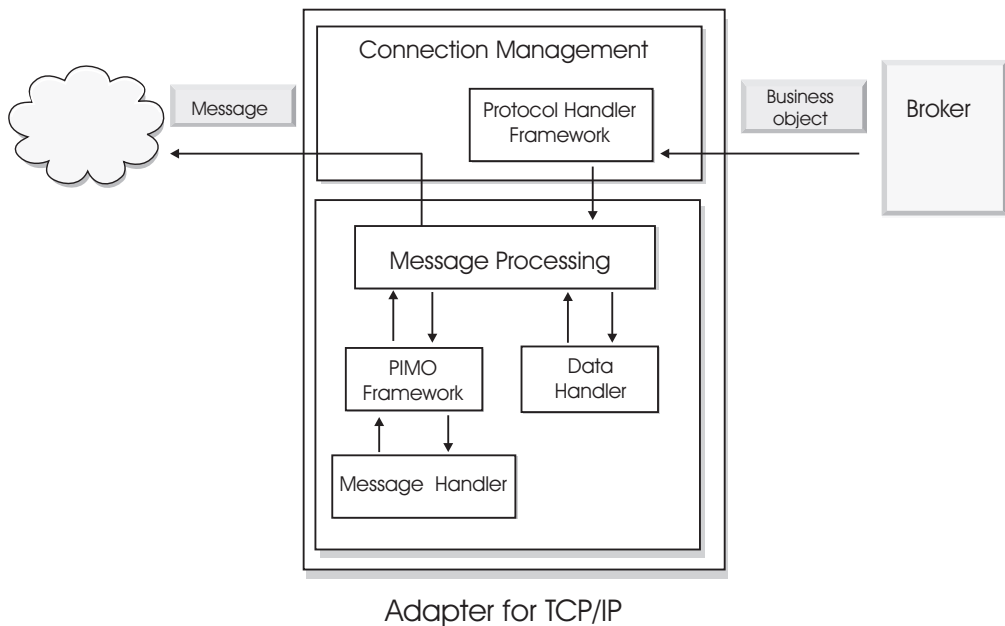


Figure 2. Architecture of the connector - Request (outbound) processing

Connection management

The connector, the runtime component of the adapter for TCP/IP, is capable of serving as either a TCP server (in inbound, or event processing, mode) or a TCP client (in outbound, or service request, mode). The module in the connector that implements the actual TCP protocol is called the Connection Manager.

The Connection Manager has two parts: the Protocol Listener Framework and the Protocol Handler Framework.

Protocol Listener Framework: The Protocol Listener Framework serves as a TCP server, based on properties set in the ServerConfiguration section of the connector configuration file, the CFG. See Appendix B, “Connector-specific properties and required business-object properties,” on page 89 for more information on the CFG.

The Framework listens for incoming requests and provides for parallel processing of multiple requests through a thread management component called the *request pool*. Every new incoming request becomes a new request object in the request pool. The size of the pool is set by the MaxRequestPoolSize configuration property.

Requests in the pool are processed by worker threads, up to the number configured by the MaxRequestProcessors property in the CFG. At any given moment, the connector can process at most (MaxRequestPoolSize + MaxRequestProcessors) requests. Thread management and load balancing are also taken care of here. The worker threads package the incoming requests into byte arrays and hand them off to the Message Processing Framework.

Protocol Handler Framework: The Protocol Handler Framework serves as the TCP client, taking service call request data and sending it to remote hosts based on properties set in the ClientConfiguration part of the CFG.

The Protocol Handler Framework uses *connection pooling* to improve performance by reusing client connections to the external TCP/IP socket server. To use connection pooling, set ConnectionPoolingOn to True. You can set the number of connections with the ConnectionPoolSize property.

Message processing management

The Message Processing Framework manages the transformation of incoming event data into WebSphere business integration-usable business objects and of outgoing service request business objects into various supported message structures that can be sent over TCP/IP. It is made up of three parts:

- The PIMO framework
- Message handlers, which are configured in PIMO maps
- Data handlers

The following sections provide an overview of the PIMO framework and describe how the components are used in both inbound and outbound processing. Details of the PIMO framework, including the business objects and meta-objects used in PIMO processing, are described in Chapter 3, “Business objects and meta-objects in the TCP/IP adapter,” on page 13.

The PIMO framework: The PIMO framework is a highly flexible, general-purpose abstraction for effecting business logic processing at the connector level. The PIMO framework is capable of a wide range of operations, and is used, in differing forms, in other adapters.

Message structures that come in over a TCP/IP connection might require a certain amount of pre-processing before they can be sent on to a data handler to be turned into WebSphere business integration business objects. In the TCP/IP connector, the PIMO framework has been customized specifically to take care of pre- and post-processing issues.

Take, for example, the HL7 health care data standard. The details of network transmission error detection and correction are handled by the lower levels of most modern network protocols, so the main standard does not include specifications

covering them. However, many mini and mainframe computer systems, which might be part of the HL7 data flow, operate in communication environments that do not provide sufficient lower-layer functionality. In these cases, HL7 offers several alternate lower-layer protocols, such as the Hybrid Low Layer Protocol and the Minimal Low Layer Protocol, to suit different environments. Messages sent using these protocols must be pre-processed to remove this protocol-related information before the main body of data can be extracted.

The PIMO framework is designed to do exactly this sort of pre-processing by invoking one or more PIMO *maps*. The maps for the adapter are configured in the BIA_MO_Tcpip_MapSubscriptions PIMO meta-object. Each map has a message handler associated with it. These maps are provided to the PIMO framework when a message or business object is sent to the framework.

Inbound processing flow: In event (inbound) processing, the adapter's connector acts as a TCP server, listening on the designated socket, and setting up and managing the load on the socket once the connection with a client is established.

The data goes from the Connection Management component to the Message Processing component, where some basic pre-processing is done by the PIMO.

1. Event data coming in from the request pool is run through the PIMO framework, where certain pre-processing operations are done using functionality found in message handlers.
2. The data is then handed over to a data handler, an independent plug-in used by many different adapters, to be built into a WebSphere business integration-usable business object. The choice of data handlers is based on the DataHandlerMimeType and related properties in the CFG.
3. The business object is passed on to the broker. The structure of the business object is completely determined by the data handler, so the adapter for TCP/IP is capable of handling any sort of message data for which there is a related data handler.
4. If this is a synchronous request, a response is returned from the broker.

One TCP server can be configured per connector instance.

Outbound processing flow: In request (outbound) processing, the integration broker sends the connector a business object that represents the message it wishes to send to a pre-configured target host or application.

1. A data handler converts the object into a message, which might also, if necessary, be sent for post-processing by the PIMO subcomponent.
2. The connector, acting as a TCP client, contacts the appropriate server, establishes the connection, and manages sending the data.
3. If required, the connector also handles any responses that the target server might provide.

Multiple TCP clients can be configured per connector instance.

Common Event Infrastructure

This adapter is compatible with the IBM Common Event Infrastructure, a standard for event management that permits interoperability with other IBM WebSphere event-producing applications. If Common Event Infrastructure support is enabled, events produced by the adapter can be received (or used) by another Common Event Infrastructure-compatible application.

For more information, refer to Appendix D, “Common Event Infrastructure,” on page 101.

Application Response Measurement

This adapter is compatible with the Application Response Measurement application programming interface (API), an API that allows applications to be managed for availability, service level agreements, and capacity planning. An ARM-instrumented application can participate in IBM Tivoli^(R) Monitoring for Transaction Performance, allowing collection and review of data concerning transaction metrics.

For more information, refer to Appendix E, “Application Response Measurement,” on page 107.

Chapter 2. Installing the connector

This chapter provides a description of the tasks you must complete to install the TCP/IP connector.

- “The adapter environment”
- “Installing the connector” on page 11
- “Verifying the installation” on page 11

The adapter environment

Before installing the adapter, you must understand its environmental requirements as described in the following sections:

- “Broker compatibility”
- “Adapter platforms”
- “Processing locale-dependent data” on page 10

Broker compatibility

This adapter runs with the WebSphere Business Integration Adapter FrameworkV2.6 and requires one of the following:

- WebSphere InterChange ServerV4.2.2, V4.3
- WebSphere MQ IntegratorV2.1
- WebSphere MQ Integrator BrokerV2.1
- WebSphere Business Integration Message BrokerV5.0.1
- WebSphere Application Server EnterpriseV5.0.2, in conjunction with WebSphere Studio Application Developer Integration EditionV5.0.1
- WebSphere Business Integration Server FoundationV5.1.1

See the *Release Notes* for any exceptions.

Note: For instructions on installing the integration broker and its prerequisites, see the following documentation.

- For WebSphere InterChange Server, see the *System Installation Guide for UNIX or for Windows*.
- For message brokers (WebSphere MQ Integrator Broker, WebSphere MQ Integrator, and WebSphere Business Integration Message Broker), see *Implementing Adapters with WebSphere Message Brokers* and the installation documentation for the message broker. Some of this can be found at the following Web site:
<http://www.ibm.com/software/integration/mqfamily/library/manualsa/>.
- For WebSphere Application Server, see *Implementing Adapters with WebSphere Application Server* and the documentation at
<http://www.ibm.com/software/webservers/appserv/library.html>.

Adapter platforms

In addition to a broker, this adapter requires one of the following operating systems:

Note: All operating system environments require the Java compiler (IBM JDK 1.4.2 for Windows^(R) 2000) for compiling custom adapters.

- AIX^(R):
 - AIX 5.1 with Maintenance Level 4.
 - AIX 5.2 with Maintenance Level 1.
This adapter supports 32-bit JVM on a 64-bit platform.
 - AIX 5.3.
- Solaris:
 - Solaris 8 (2.8) with Solaris Patch Cluster dated Feb. 11, 2004 or later.
 - Solaris 9 (2.9) with Solaris Patch Cluster dated February 11, 2004 or later.
This adapter supports 32-bit JVM on a 64-bit platform.
- HP-UX: HP-UX 11.i (11.11) with June 2003 GOLDBASE11i and June 2003 GOLDAPPS11i bundles.
- Linux^(TM):
 - Red Hat Enterprise Linux AS 3.0 with Update 1.
 - Red Hat Enterprise Linux ES 3.0 with Update 1.
 - Red Hat Enterprise Linux WS 3.0 with Update 1.
 - SUSE Linux Enterprise Server x86 8.1 with SP3.
 - SUSE Linux Standard Server x86 8.1 with SP3 v 16.

Note: The TMTP (Tivoli Monitoring for Transaction Performance) component of the WebSphere Business Integration Adapter FrameworkV2.6 is not supported on Linux Red Hat.

- Windows:
 - Windows 2000 (Professional, Server, or Advanced Server) with Service Pack 4.
 - Windows XP with Service Pack 1A, for WebSphere Business Integration Adapter Framework (administrative tools only).
 - Windows 2003 (Standard Edition or Enterprise Edition).

Processing locale-dependent data

The connector has been internationalized so that it can support delivery of double-byte character sets (DBCS) going into an interface that also supports double-byte character sets and to deliver message text in the specified language. 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 run time 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.

Installing the connector

For information on installing WebSphere Business Integration adapter products, refer to the *Installing WebSphere Business Integration Adapters* guide located in the WebSphere Business Integration Adapters Information Center at the following site:

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

Verifying the installation

Once you have installed the connector, the files in the following tables should be installed on your machine. In the tables, *ProductDir* represents the directory in which you have installed the IBM WebSphere Business Integration Adapter Software.

Installed Windows file structure

The table below describes the file structure used by the connector as installed in a Windows machine.

Table 2. Files installed with the connector - Windows

Directory	Installed files
<i>ProductDir</i> \bin\Data\App	BIA_TCPIPConnectorTemplate The connector's configuration file template
<i>ProductDir</i> \connectors\TCPIP	BIA_TCPIP.jar and other resources
<i>ProductDir</i> \connectors\messages	BIA_TCPIPAdapter.txt The connector message file, containing error messages and codes
<i>ProductDir</i> \repository\TCPIP\sample	BIA_TCPIPAdapterRepository.zip contains the samples

Installed UNIX file structure

The table below describes the file structure used by the connector as installed on a UNIX machine.

Table 3. Files installed with the connector - UNIX

Directory	Installed files
<i>ProductDir</i> /bin/Data/App	BIA_TCPIPConnectorTemplate The connector's configuration file template
<i>ProductDir</i> /connectors/TCPIP	BIA_TCPIP.jar and other resources
<i>ProductDir</i> /connectors/messages	BIA_TCPIPAdapter.txt The connector message file, containing error messages and codes
<i>ProductDir</i> /repository/TCPIP/sample	BIA_TCPIPAdapterRepository.zip contains the samples

Chapter 3. Business objects and meta-objects in the TCP/IP adapter

This chapter describes the business objects and meta-objects that are used by the adapter for TCP/IP.

The first part of this chapter, “Processing flow,” describes the PIMO framework and shows how the business objects and meta-objects are used by the adapter for TCP/IP.

The second half of this chapter provides reference material for the following types of objects:

- Internal business objects
These are used as transitional data wrappers, as data is pulled off the network (in event mode) and as it is being fed to the network (in service call request mode). These objects and their structure are described in detail in “Internal business objects” on page 26.
- General meta-objects
These meta-objects correspond to, and are set by, properties in the connector configuration file. These objects and their structure are described in detail in “General meta-objects” on page 29.
- PIMO framework meta-objects
These meta-objects are used to perform object manipulation inside the connector. In the adapter for TCP/IP, this mechanism is used to provide pre- and post-processing of message data.
The PIMO framework is described in “PIMO framework processing” on page 14. The PIMO framework objects and their structure are described in “PIMO framework meta-objects” on page 33.

Processing flow

As described in “Connector architecture” on page 3, the connector, the runtime component of the adapter for TCP/IP, is designed as a general purpose conduit to route data transmitted directly over TCP/IP networks under well-known protocols, such as the HL7 protocol in the health care industry, into and out of the WebSphere business integration system.

- Inbound (or *event*) information is captured from the network message stream by the connector, transformed into a WebSphere business integration business object, and published to the integration broker. This process is described in “Event processing” on page 24.
- Outbound (or *service call request*) information is received from the integration broker as a WebSphere business integration business object, transformed into a network message stream, and sent back over the network. This process is described in “Service call request processing” on page 18.

The nature of the WebSphere business integration business object in this flow is completely dependent on the data handler, a data transformation plug-in that the connector calls based on settings in the connector configuration file. The data handler, and not the adapter itself, translates the messages to and from the appropriate WebSphere business integration business object form.

PIMO framework processing

The PIMO framework is used in both service call request processing and event processing.

The PIMO framework provides an abstract mechanism for performing certain kinds of object manipulation inside the connector. In the adapter for TCP/IP, this mechanism is used to provide pre- and post-processing of message data.

Overview

The PIMO framework uses a set of specially designed meta-objects to do its work. At the top of the adapter's PIMO hierarchy is the BIA_MO_Tcpip_MapSubscriptions object. Figure 3 shows a sample BIA_MO_Tcpip_MapSubscriptions meta-object. Figure 26 on page 34 shows what this object looks like in the Business Object Designer.

This object designates both the inbound (event pre-processing) and outbound (service call request post-processing) paths that data will take. It contains two objects:

- BIA_MO_Tcpip_MapSubscriptions_In
- BIA_MO_Tcpip_MapSubscriptions_Out

Each object contains a reference to one or more PIMO maps.

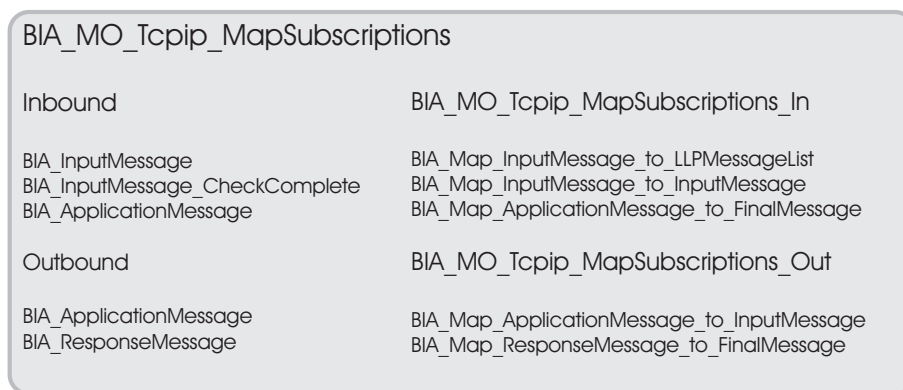


Figure 3. BIA_MO_Tcpip_MapSubscriptions object

PIMO objects consist of three basic attributes: Port, Declaration, and Action.

Figure 4 on page 15 shows the structure of a BIA_MO_Tcpip_MapSubscriptions meta-object that contains three inbound maps and two outbound maps.

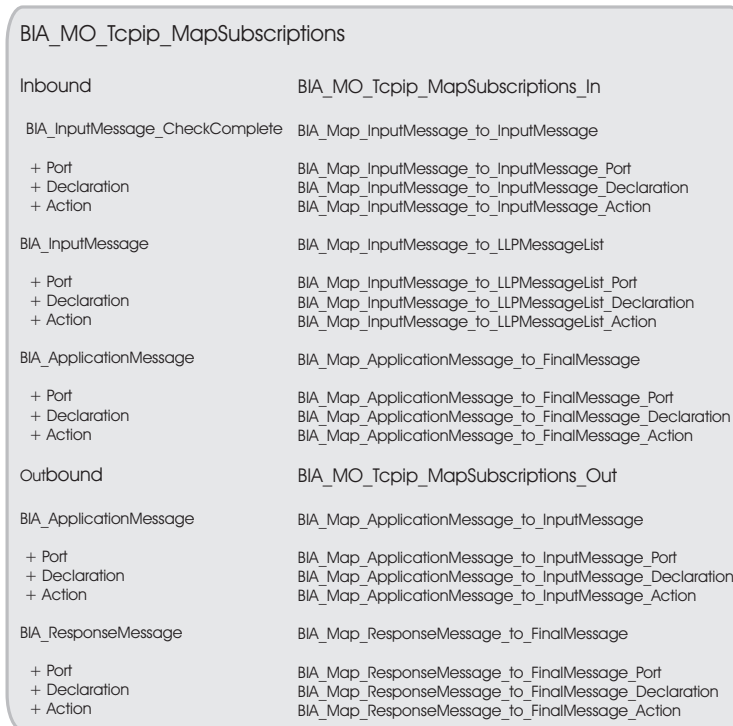


Figure 4. PIMO map structure

Port: Each Port is made up of two attributes:

- IPort
- OPort

These attributes indicate the expected type of the source object (for example, a BIA_InputMessage object, which is the internal wrapper object for event processing) and the destination object (for example, a BIA_LLPMessagesList object).

Note: The BIA_LLPMessagesList map is a sample map provided by default with the connector; however, the adapter for TCP/IP can be used for more than just HL7 message processing. Customize the provided sample maps to meet the needs of your industry.

Figure 5 shows two input maps chained together, the first separating multiple messages into single messages, and the second stripping away the LLP information from the actual HL7 message.

Separating the process into multiple steps can produce more complex types of processing. If chained maps are used, it is essential that the oPort type of the first step be exactly the same as the iPort type of the second step, and so forth.

BIA_MO_Tcpip_MapSubscriptions

Inbound	BIA_MO_Tcpip_MapSubscriptions_In
BIA_InputMessage_CheckComplete	BIA_Map_InputMessage_to_InputMessage
- Port	BIA_Map_InputMessage_to_InputMessage_Port
+ iPort	BIA_InputMessage
+ oPort	BIA_InputMessage
+ Declaration	BIA_Map_InputMessage_to_InputMessage_Declaration
+ Action	BIA_Map_InputMessage_to_InputMessage_Action
BIA_InputMessage	BIA_Map_InputMessage_to_LLPMessagesList
- Port	BIA_Map_InputMessage_to_LLPMessagesList_Port
+ iPort	BIA_InputMessage
+ oPort	BIA_LLPMessagesList
+ Declaration	BIA_Map_InputMessage_to_LLPMessagesList_Declaration
+ Action	BIA_Map_InputMessage_to_LLPMessagesList_Action

Figure 5. iPort and oPort declarations

Declaration: The declaration attribute is optional. It contains names for temporary variables to be used during processing.

Action: Each Action attribute consists of one or more defined actions. The Application Specific Information (ASI) for each defined action provides the information the PIMO needs to invoke the message handler, a native Java class designed to perform the actual data transform that is required. The message handler provides methods for parsing (unwrapping) or constructing (wrapping) the TCP/IP messages based on a certain protocol.

An example ASI follows:

```
type=nativeStatic;
class=com.ibm.adapters.tcpip.messagehandlers.LLPMessagingProtocolHandler;
method=parseInputMessageToLLPMessages; target=contentText; iPort; oPort
```

The ASI is described in “PIMO framework meta-objects” on page 33. This section provides a general description of the actions.

The method at the heart of this set, `parseInputMessageToLLPMessages`, separates the LLP-specific wrapper data and then returns a list that stores the individual parts of the `LLPMessage` (the header, the message itself, and the trailer) so that the message (as a `BIA_ContentBO`) can be handed off to the data handler. A set of these HL7 message handlers is included in the TCP/IP installation, but others can be developed as the need arises.

Note: LLP and HL7 are examples of protocols that can be processed by the adapter for TCP/IP. The adapter can actually be customized to process many protocols over the TCP/IP stack. All the presentation/protocol layers above TCP/IP can be processed by message handlers and data handlers.

Service call request processing follows the same stages except in reverse: the PIMO framework wraps messages in their protocol-specific data before sending them to be forwarded to the remote host.

PIMO maps

This section lists the default PIMO maps that are provided with the adapter for TCP/IP. If you need to write a custom message handler, you can use a default map and change the value against the Action attribute.

The way PIMO maps are used by the adapter for TCP/IP is described in “Service call request processing” on page 18 and “Event processing” on page 24. Reference information about these maps is provided in “PIMO framework meta-objects” on page 33.

Inbound: The Inbound attribute indicates the set of maps used by PIMO in event mode processing. Three inbound maps are provided with the adapter:

- **BIA_InputMessage**
This map is invoked in event processing. It is used for complete messages.
- **BIA_InputMessage_CheckComplete**
This map is invoked in event processing. This map is used when, on a single socket read call, multiple messages arrive.
To switch on this map, set the `SupportMultipleMessages` in `BIA_MO_Service` to true for the mime type configured in the connector configuration file.
`BIA_InputMessage_CheckComplete` map filters complete messages from incomplete messages. The complete messages are passed to `BIA_InputMessage` for processing. For incomplete messages, the adapter waits on the socket for further data. The `BIA_InputMessage_CheckComplete` is then invoked again.
If `SupportMultipleMessages` is false, only `BIA_InputMessage` map is invoked. The `BIA_InputMessage` map is for processing complete messages.
- **BIA_ApplicationMessage**
This map is invoked in synchronous event processing, when the response from the broker comes to the adapter for the synchronous event.

Outbound: The Outbound attribute indicates the set of maps used by PIMO in service call request mode processing. Two outbound maps are provided with the adapter:

- **BIA_ApplicationMessage**
This map is invoked to process the data returned by the data handler. The processed data is then written by the adapter to the TCP/IP socket.
- **BIA_ResponseMessage**
This map is invoked in synchronous request processing on the response data received by the adapter from the TCP/IP server during a synchronous request call.

Note that the `iPort` for any of the default maps provided with the adapter is predetermined. The `iPorts` are the business objects (for example, `BIA_InputMessage`) shipped with the adapter. The `oPorts` usually do not have any

requirements except that they wrap the content in the Content attribute of type BIA_ContentBO. See “Map usage rules” on page 36 for details.

You can add to this list of maps, and you can bypass maps that are not needed. See “Bypassing all maps” on page 38.

Service call request processing

In service call request processing, a business object is sent by the broker to the adapter. After it is transformed, it is sent over the network.

The business object sent by the broker to the adapter in synchronous processing typically has two attributes:

- The first attribute (Request in Figure 6) is populated by the broker and sent to the adapter.
- The second attribute (Response in Figure 6) is populated by the adapter and sent back to the broker. This attribute would be empty in the business object initially sent to the connection from the broker.

This structure is not required, however. You can have the broker send information in the first and second attribute or any other attributes. You can have other attributes in the business object structure that can be ignored during request processing.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Ap
1	1	Request	BIA_ContentBO	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
2	2	Response	BIA_ContentBO	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
3	3	ObjectEventId	String							
4	4			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 6. Structure of the business object sent by the broker

When a request business object arrives, the Adapter Framework invokes the Protocol Handler Framework, and the following flow occurs:

1. The Protocol Handler Framework passes the incoming business object to Message Processor.
2. Message Processor reads the following properties from the connector configuration file:
 - DataHandlerMimeType
 - ServiceRegistrationMO
 - DataHandlerMetaObjectName
3. Message Processor invokes the appropriate data handler class, using information in DataHandlerMimeType, and the method of the data handler class, using information in ServiceRegistrationMO, for the mime type.
4. The data handler reads the business object content and forms the application data. The data handler then returns the application data.

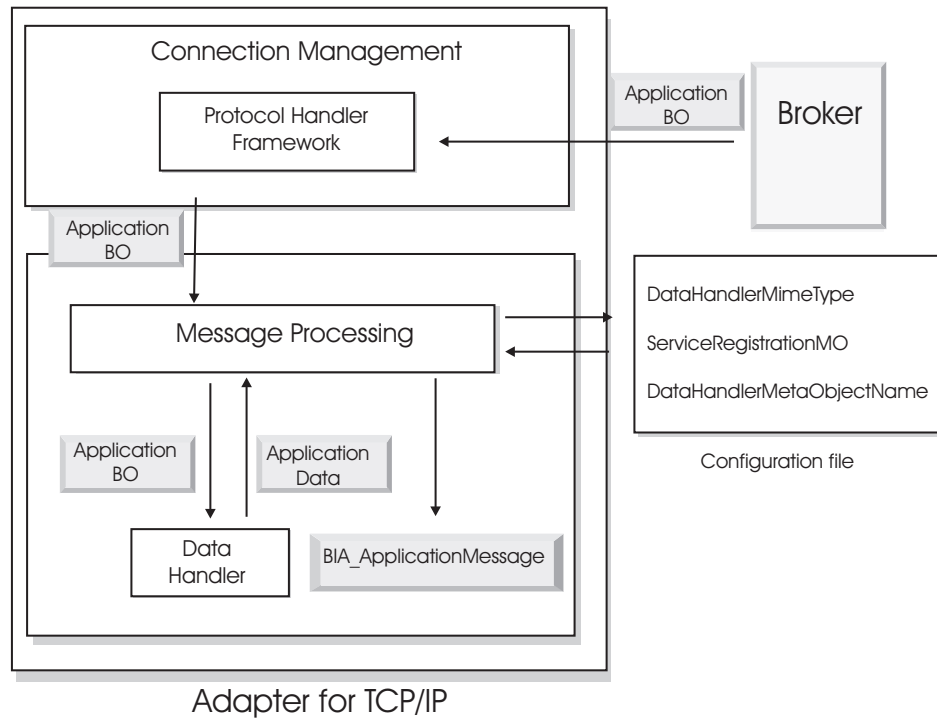
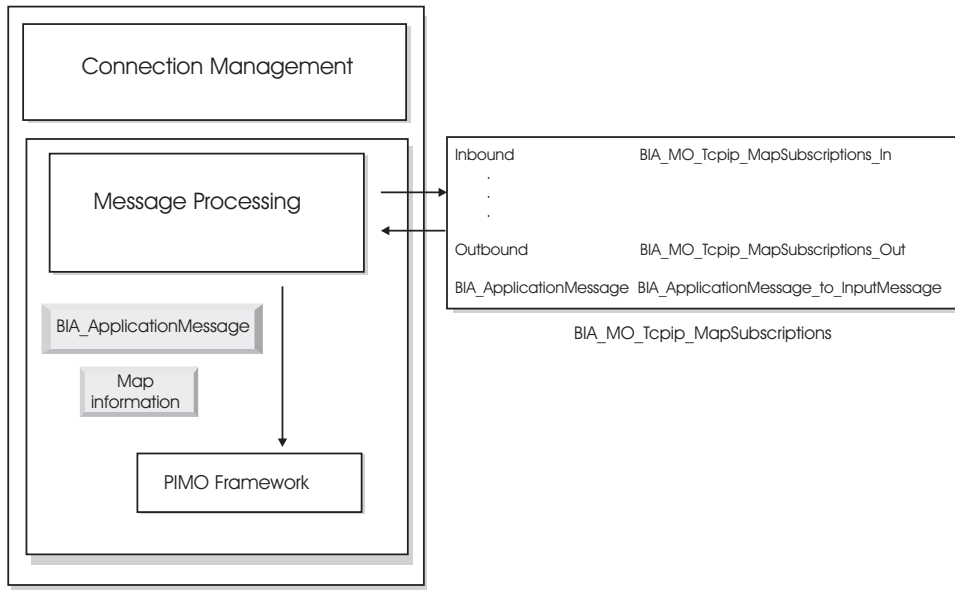


Figure 7. Request processing flow - Part 1

5. Message Processor wraps the application data returned by the data handler in a business object (BIA_ApplicationMessage).
6. Message Processor reads the map information contained in a BIA_ApplicationMessage child attribute contained in the Outbound attribute of BIA_MO_Tcpip_MapSubscriptions.
7. Message Processor invokes the PIMO framework, passing the map information related to the business object, BIA_ApplicationMessage.



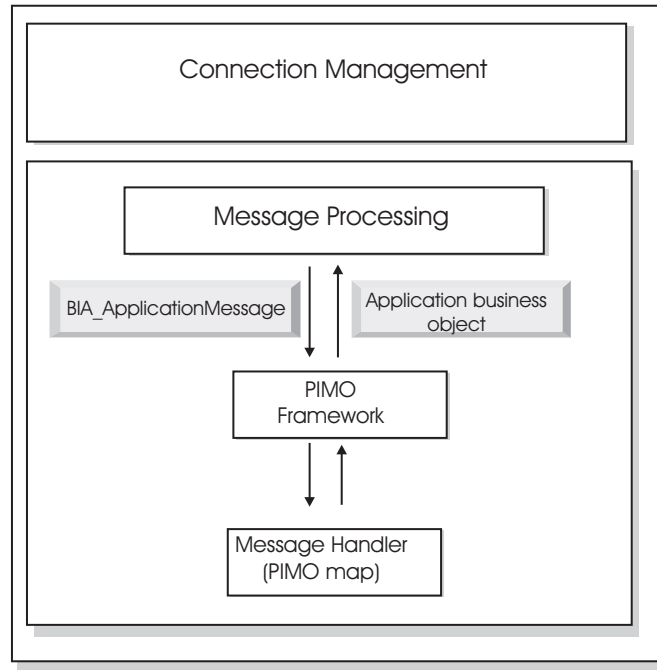
Adapter for TCP/IP

Figure 8. Request processing flow - Part 2

8. The PIMO framework uses the map information to invoke the appropriate message handler, which transforms the `BIA_ApplicationMessage` to the appropriate business object.

The advantage of using the PIMO framework and maps is that you can control different application data. For example, the PIMO framework could invoke an LLP message handler that accepts `BIA_ApplicationMessage` as an input parameter and wraps the data contained in the `BIA_ApplicationMessage` in LLP protocol headers and trailers. The LLP message handler could then wrap the resultant content into a business object and return the business object to the PIMO framework.

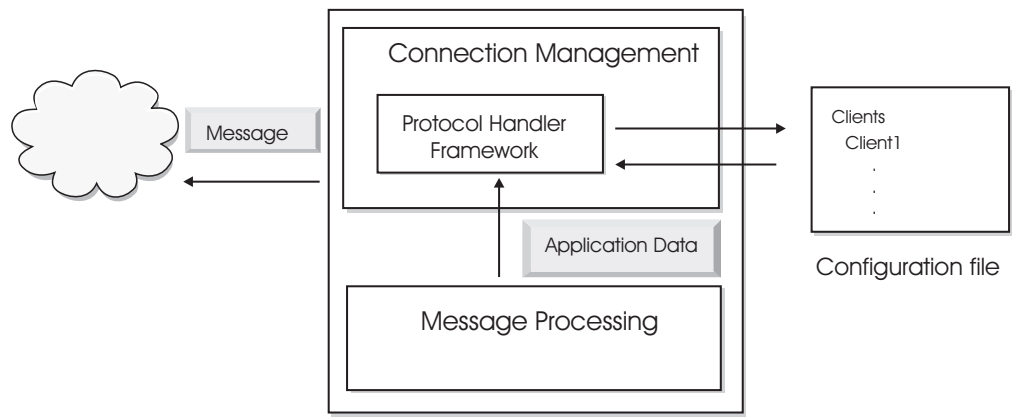
9. The PIMO framework returns the business object, returned by the message handler configured in the map, to the Message Processor.



Adapter for TCP/IP

Figure 9. Request processing flow - Part 3

10. The Message Processor receives the business object from the PIMO framework and again reads the map information contained in the Outbound attribute of BIA_MO_Tcpip_MapSubscriptions for the particular business object name. If further maps are configured for the business object, the Message Processor invokes the PIMO framework again.
11. The final application data is returned to the Protocol Handler Framework. The Protocol Handler Framework, using the client configuration details in the connector configuration file, connects to the remote server and sends the application data to the remote server.



Adapter for TCP/IP

Figure 10. Request processing flow - Part 4

The way response from the server is processed depends on whether this is an asynchronous or a synchronous transaction.

Asynchronous processing

In asynchronous request processing, no response is expected by the adapter from the destination TCP/IP socket server. If the message can be delivered over the socket to the destination, the adapter sends back a success indicator to the broker.

Synchronous processing

In synchronous processing, the response from the server is processed, and the response is sent back to the broker, according to the following steps:

1. The TCP/IP server returns the response to the connector. The connector passes this response to a configured message handler (the BIA_ResponseMessage map in the Outbound attribute of BIA_MO_Tcpip_MapSubscriptions).

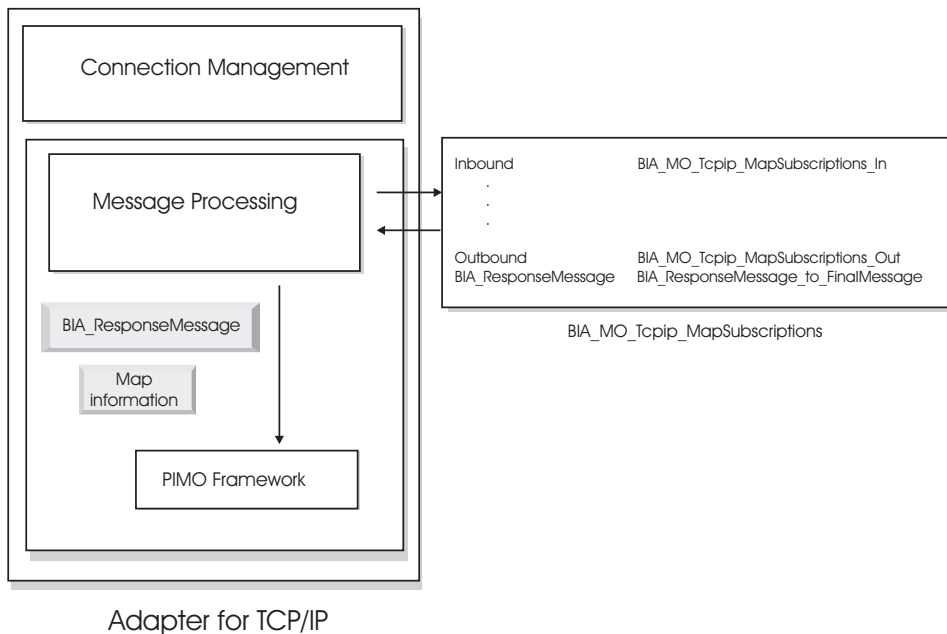


Figure 11. Response processing (synchronous mode)

2. The message handler processes the response. This message handler could invoke a data handler to form the Response attribute business object.

Note: The oPort of the BIA_ResponseMessage map conveys information to the connector (as described in steps 2a through 2f) and should have the following attributes:

BIA_MO_Tcpip_MapSubscriptions	
Inbound	BIA_MO_Tcpip_MapSubscriptions_In
Outbound	BIA_MO_Tcpip_MapSubscriptions_Out
+ BIA_ApplicationMessage	BIA_Map_ApplicationMessage_to_InputMessage
- BIA_ResponseMessage	BIA_Map_ResponseMessage_to_FinalMessage
- Port	BIA_Map_ResponseMessage_to_FinalMessage_Port
+ iPort	BIA_ResponseMessage
- oPort	BIA_FinalMessage
CharSet	String
Status	String
+ Content	BIA_ContentBO
ResponseDataMimeType	String
ResponseAttributeName	String
+ Declaration	BIA_Map_ResponseMessage_to_FinalMessage_Declaration
+ Action	BIA_Map_ResponseMessage_to_FinalMessage_Action

Figure 12. BIA_ResponseMessage map

- a. The message handler serializes, to a string, the response business object that will be sent back to the broker.
- b. The message handler populates the ResponseDataMimeType attribute of the oPort business object to convey the mime type used for serialization.
- c. The message handler populates the Content attribute of the oPort business object with the serialized string.
- d. The message handler uses the ResponseAttributeName of the oPort business object to convey to the adapter which attribute of the incoming request business object, from the broker, has to be populated with the de-serialized data.
- e. The message handler uses the CharSet attribute of the oPort business object to convey the character set of the data.
- f. The message handler communicates a status of VALCHANGE, in the Status attribute of the oPort business object of the BIA_ResponseMessage map, to the connector. This indicates to the connector that it needs to send back a status of VALCHANGE to the broker, indicating that the business object value, initially sent by the broker, has been changed.

Note: Note that the processing logic of the Response business object is up to the message handler and the data handler that have been custom written for your business requirement. If the message handler is written to ignore the Response from the TCP/IP server, SUCCESS can be returned instead, in the Status attribute, to the broker.

If you need to pass the Response back to the broker, the message handler needs to be written to handle the response from the TCP/IP server.

3. The connector reads the oPort and de-serializes the data in the Content attribute of the oPort. The de-serialization is done using the mime type populated in the ResponseDataMimeType attribute.

The de-serialized data is populated by the adapter in the ResponseAttributeName attribute of the request business object. The request business object is the business object that comes from the broker to the adapter.

Event processing

The Protocol Listener Framework receives data on a TCP/IP socket, and the following flow occurs:

1. The Protocol Listener Framework passes the application data, incoming over a socket, to the Message Processor.
2. The Message Processor forms a business object, `BIA_InputMessage`, from the application data. The application data contained in `BIA_InputMessage` at this moment might contain complete as well as incomplete application messages.
3. This step is performed if the following two items are true:
 - The `BIA_InputMessage_CheckComplete` map is specified in `BIA_MO_Tcpip_MapSubscriptions`.
 - The context `SupportMultipleMessages` in `BIA_MO_Service` is set to true for the mime type configured in the connector configuration file.

`BIA_InputMessage_CheckComplete` map filters complete messages from incomplete messages.

- The complete messages are passed to `BIA_InputMessage` for processing.
- For incomplete messages, the adapter waits on the socket for further data. The `BIA_InputMessage_CheckComplete` is then invoked again. The complete messages are then transformed into a `BIA_InputMessage` business object.

If `SupportMultipleMessages` is false, only `BIA_InputMessage` map is invoked. The `BIA_InputMessage` map is for processing complete messages.

4. The business object `BIA_InputMessage` is sent to the PIMO framework, which determines the appropriate message handler and invokes it. The message handler parses for complete messages and returns them to the Message Processor to be further processed by other message handlers.

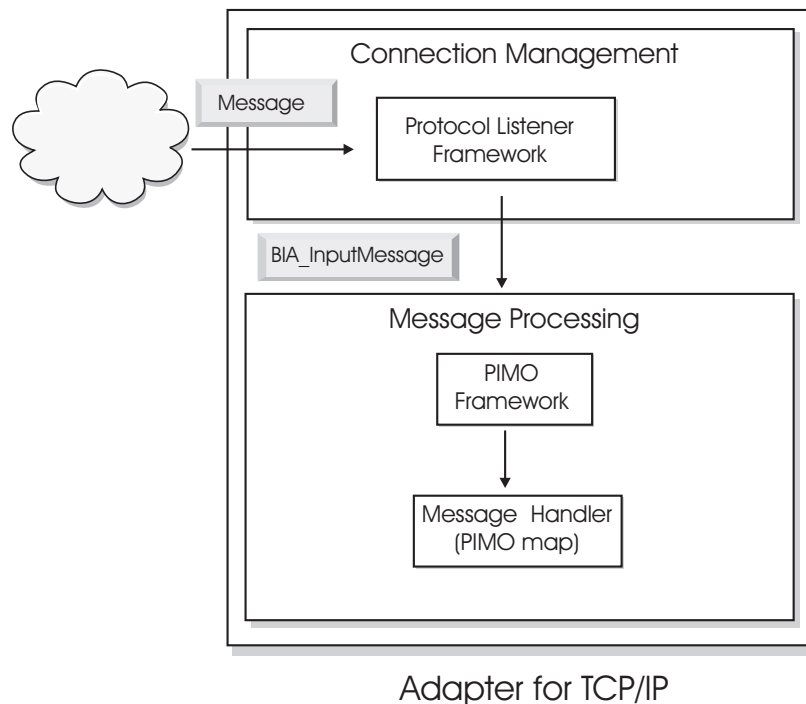
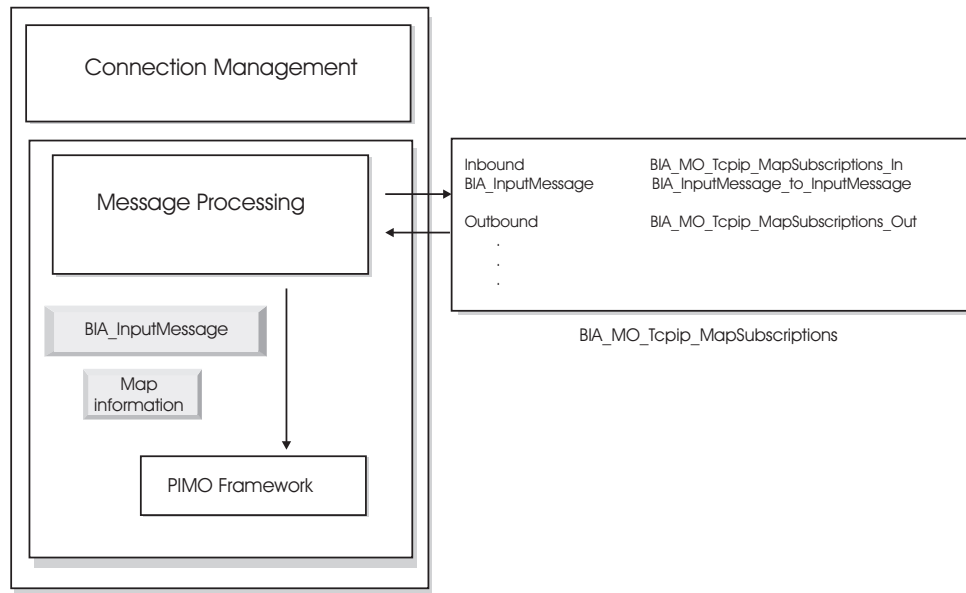


Figure 13. Event processing - Part 1

5. The Message Processor receives the business object from the PIMO framework and reads the map information contained in the Inbound attribute of

BIA_MO_Tcpip_MapSubscriptions for the particular business object name. If further maps are configured for the business object, the Message Processor invokes the PIMO framework again.



Adapter for TCP/IP

Figure 14. Event processing - Part 2

6. The Message Processor reads the DataHandlerMimeType, ServiceRegistrationMO, and DataHandlerMetaObjectName properties from the connector configuration file. The Message Processor invokes the appropriate data handler class, using information in the data handler meta-object for the mime type, and the method of the data handler class, using information in the Service Registration Meta Object for the mime type. The data handler returns the application business object for the application data sent to it.

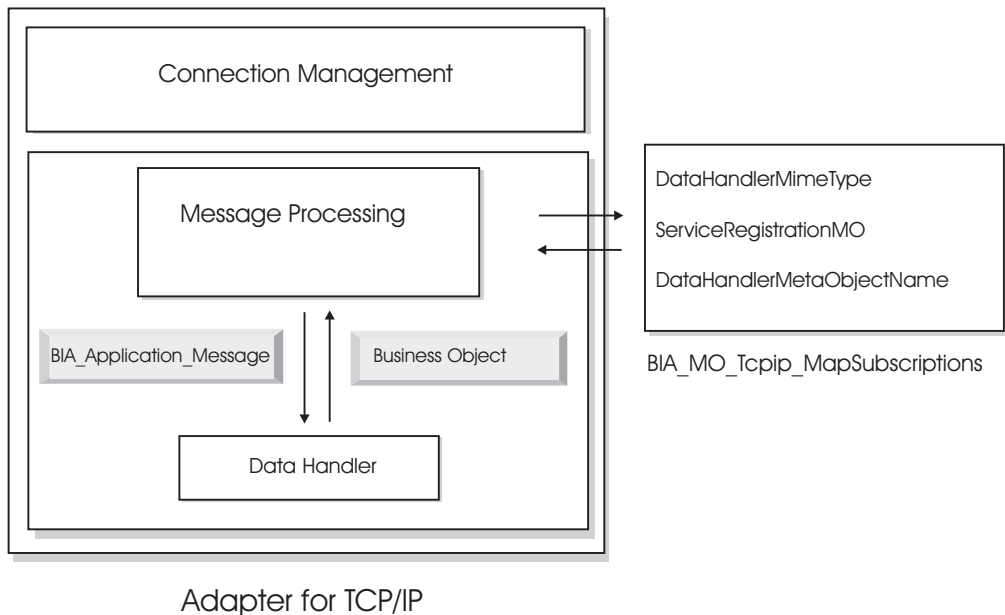


Figure 15. Event processing - Part 3

7. The final application business object is sent to the broker in either synchronous or asynchronous mode. The response from the broker is sent to the data handler. The response from the data handler is wrapped in BIA_ApplicationMessage, and the BIA_ApplicationMessage map in the Inbound attribute is invoked. The Protocol Listener Framework sends the final application data back to the application server.

Internal business objects

This section provides reference information about internal business objects.

The connector's internal business objects are used as transitional data wrappers, as data is pulled off the network (in event mode) and as it is being fed to the network (in service call request mode).

Note: The definition files for all internal business objects and meta-objects are stored as XML schema files (.xsd) in the following directories:
ProductDir\connectors\TCPIP\Samples for Windows and
ProductDir/connectors/TCPIP/Samples for UNIX. They can be viewed using either the Business Object Designer or an XML-capable browser.

BIA_ContentBO

In event mode, the connector functions as a TCP server, listening on a socket for requests from remote applications to establish a channel to transmit data. The Connection Management subcomponent establishes the connection and manages the incoming data stream from the network, including load balancing and setting up parallel processes to handle multiple requests. As the data flows in, it is passed to the Message Processing component, where it is held in a BIA_ContentBO, the basic data wrapper.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information
1	1	Content	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		
2	2	ObjectEventId	String							
3	3			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 16. BIA_ContentBO in the Business Object Designer

The pertinent attribute of the BIA_ContentBO is as follows:

Business-object level attribute	Description
Content	Stores application or protocol data.

BIA_InputMessage

The content object is contained in a BIA_InputMessage business object. The input message object may initially include complete and incomplete messages from the remote application. The connector separates complete and incomplete messages, queuing the incomplete messages until they are complete, and sending completed messages, wrapped in their BIA_InputMessage objects, to the PIMO framework, where some forms of pre-processing may be done before they are passed on to the data handler for final processing.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information
1	1	CharSet	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		
2	2	Content	BIA_ContentBO	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	N			
2.1	2.1	Content	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		
2.2	2.2	ObjectEventId	String							
3	3	ObjectEventId	String							
4	4			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 17. BIA_InputMessage in the Business Object Designer

The pertinent attributes of the BIA_InputMessage object are as follows:

Business object-level attributes	Description
CharSet	Encoding applied to the incoming bytes.
Content	Stores content coming in the socket as BIA_ContentBOs. It is of N cardinality because attribute length is limited compared to the length of the incoming message.

BIA_ApplicationMessage

In service call request mode, the WebSphere business integration business objects that represent messages to be transmitted to the remote application are sent from the integration broker. These objects are translated into the appropriate message form by the data handler. The connector wraps this message data in a BIA_ContentBO, which is contained in a BIA_ApplicationMessage object. The message data may be subject to PIMO post-processing, after which the connector, acting as a TCP client, sends the message data back out over the TCP/IP network to a target specified in the connector configuration file.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information
1	1	CharSet	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		
2	2	<input type="checkbox"/> Content	BIA_ContentBO	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	N			
2.1	2.1	Content	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		
2.2	2.2	ObjectEventId	String							
3	3	ObjectEventId	String							
4	4			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 18. BIA_ApplicationMessage in the Business Object Designer

The pertinent attributes of the BIA_ApplicationMessage object are as follows:

Business object-level attributes	Description
Charset	Encoding applied on incoming bytes.
Content	Stores content coming in from the data handler as BIA_ContentBOs. It is of N cardinality because attribute length is limited compared to the length of the incoming message.

BIA_ResponseMessage

In asynchronous processing, the BIA_ResponseMessage object contains any acknowledgment message--wrapped in a content object--from the remote application as a result of a service call request.

In synchronous processing, the BIA_ResponseMessage object contains the response from the remote application.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information
1	1	CharSet	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
2	2	<input type="checkbox"/> Content	BIA_ContentBO	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
2.1	2.1	Content	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
2.2	2.2	ObjectEventId	String							
3	3	ObjectEventId	String							

Figure 19. BIA_ResponseMessage in the Business Object Designer

The pertinent attributes of the BIA_ResponseMessage object are as follows:

Business object-level attributes	Description
Charset	Encoding applied on incoming bytes.
Content	Stores content coming in from the data handler as BIA_ContentBOs. It is of N cardinality because attribute length is limited compared to the length of the incoming message.

BIA_FinalMessage

The response received by the adapter in synchronous request processing is wrapped in BIA_ResponseMessage, and the BIA_ResponseMessage map is invoked. BIA_FinalMessage is the business object that is returned from this map to the adapter. BIA_FinalMessage encapsulates the final data that is to be returned to the broker.

Note: BIA_FinalMessage is useful only in synchronous request processing.


General		Attributes			
	Pos	Name	Type	Key	Foreign Key
1	1	CharSet	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	2	Status	String	<input type="checkbox"/>	<input type="checkbox"/>
3	3	ResponseAttributeName	String	<input type="checkbox"/>	<input type="checkbox"/>
4	4	ResponseDataMimeType	String	<input type="checkbox"/>	<input type="checkbox"/>
5	5	 Content	BIA_ContentBO	<input type="checkbox"/>	<input type="checkbox"/>
5.1	5.1	Content	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5.2	5.2	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>

Figure 20. BIA_FinalMessage in the Business Object Designer

The pertinent attributes of the BIA_FinalMessage object are as follows:

Business object-level attributes	Description
Charset	Encoding applied on incoming bytes.
Status	Indicates to the connector whether to return a SUCCESS, VALCHANGE, or FAILURE status to the broker.
ResponseAttributeName	Conveys to the connector which attribute of the incoming request business object, from the broker, has to be populated with the de-serialized data.
ResponseDataMimeType	Conveys the mime type used for serialization.
Content	Stores content coming in from the data handler as BIA_ContentBOs. It is of N cardinality because attribute length is limited compared to the length of the incoming message.

General meta-objects

There are three groups of general meta-objects in the connector that correspond to, and are set by, three properties in the connector configuration file:

- The Configuration meta-object (BIA_Static_MO)
- The Service Registration meta-object (BIA_MO_Service), which is a nested set of objects
- The Data handler meta-object (BIA_MO_DataHandler_Default), which is a nested set of objects

This section describes the objects and shows you how the objects look in the Business Object Designer. Note that these are *examples* of the types of information the business objects can contain.

Configuration meta-object (BIA_Static_MO)

The configuration meta-object is a BIA_Static_MO. It stores static meta information for processing various message types as Application Specific Information (ASI) values.

General		Attributes									
Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information		Comments
1	Default	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		requestMode=async;client=Client1;		
2	HL7_340	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		mode=sync;collabName=xyzCollab		
3	HL7_520	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		mode=sync;collabName=xyzCollab		
4	HL7_MESSAGE	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		mode=sync;collabName=PassThruCollab		
5	NCPDP_ClaimOrService	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		mode=sync;collabName=PassThruCollab		
6	LLPMessage	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		mode=async;client=Client1		
7	MyLocalBO	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		mode=sync;collabName=TCPIPToVTC;		
8	MyLocalBO_Create	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		mode=sync;collabName=TCPIPToVTC;client=Client1,requestMode=async;		
9	Employee	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		mode=sync;collabName=TCPIPToVTC		
10	Employee_Create	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		mode=sync;collabName=TCPIPToVTC		
11	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>						
12			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255				

Figure 21. BIA_Static_MO in the Business Object Designer

The pertinent attributes of this example object are as follows:

Meta-object-level attributes	Description
Default	Describes the default values.
<object>_<verb>	Describes the static meta attribute values for the verb associated with an object (for example, in Figure 21, MyLocalBO_Create and Employee_Create).
<object>	Describes the static meta attribute values associated with an object (for example, in Figure 21, MyLocalBO and Employee).

The types of static meta information for processing message types that are stored as ASI values for *each* specified message type are as follows:

Application-specific information for each message type	Description
mode =	Possible values are sync or async. When the value is sync, the connector in event processing contacts the broker in a synchronous manner. When it is async, the contact is asynchronous.
collabName =	Relevant only in synchronous event processing. Names the collaboration that the connector needs to invoke.

Application-specific information for each message type	Description
client = Clientx	Relevant for service call request processing. Names the remote server configuration stored in the connector configuration attribute Clientx.
requestMode =	Possible values are sync (for synchronous request processing) and async (for asynchronous request processing).

Service Registration meta-object (BIA_MO_Service)

The service registration meta-object is a BIA_MO_Service. This is the top-level object in a set of objects that describe multiple types of "services" that can be used in processing message types. A service is any reusable functionality. Only a data handler service is defined for this release, but others, such as a Database Service for database access, could also be defined.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific
1	1	DataHandlerService	BIA_MO_DataHandlerServiceDetails	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1			
2	2	ObjectEventId	String							
3	3			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 22. BIA_MO_Service in the Business Object Designer

The pertinent attribute of the object is as follows:

Meta-object-level attribute	Description
DataHandlerService	References a BIA_MO_DataHandlerServiceDetails, which contains further information.

The BIA_MO_DataHandlerServiceDetails object provides additional information for a data handler service.

General		Attributes								
	Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific
1	1	ServiceType		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	mime	
2	2	ServiceName		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	hl7	
3	3	ServiceInformation	BIA_MO_DataHandlerService	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
4	4	ObjectEventId	String							
5	5			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 23. BIA_MO_DataHandlerServiceDetails in the Business Object Designer

The pertinent attributes of the object are as follows:

Meta-object-level attribute	Description
Service Type	Contains additional information for specifying the service type. It is not used for connector processing.

Meta-object-level attribute	Description
ServiceName	Specifies the service name. In the case of the DataHandlerService, this is the mime type of the message to be processed as designated in the connector configuration file in the DataHandlerMimeType attribute.
ServiceInformation	References a BIA_MO_DataHandlerService, which contains further information.

The BIA_MO_DataHandlerService object provides specific information for invoking the appropriate data handler and its methods.

General		Attributes							
Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	
1	hl7	BIA_MO_DataHandlerService_HL7	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
1.1	EventMethodFormat	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		255	BusinessObjectInterface getBO(byte[], Object)	
1.2	RequestMethodFormat	String	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		255	InputStream getStreamFromBO(BusinessObjectInterface, Object)	
1.3	CharSet	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	US-ASCII	
1.4	SupportMultipleMessages	Boolean	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			true	
1.5	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
2	ncpop	BIA_MO_DataHandlerService_NCPDP	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
3	text_namevalue	BIA_MO_DataHandlerService_TextNameValue	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
4	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
5			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 24. BIA_MO_DataHandlerService in the Business Object Designer

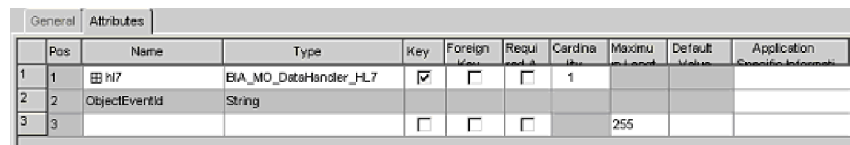
The pertinent attributes of the object are as follows:

Meta-object-level attributes	Description
hl7	Defines the handling for a specific mime type. The object (in Figure 24, BIA_MO_DataHandlerService_HL7) referenced in this attribute contains specific information used by the data handler. The related child attributes in this table (indicated by a +) belong to the contained object. They are used if the mime type listed here matches the ServiceName attribute in the BIA_MO_DataHandlerServiceDetails object.
+EventMethodFormat	Stores the HL7 DataHandler method to be invoked for event processing
+RequestMethodFormat	Stores the HL7 DataHandler method to be invoked for service call request processing
+CharSet	Stores the CharSet of the message data coming into the socket. This setting should match the CharacterEncoding property in the connector configuration file.
+SupportMultipleMessages	Tells the connector whether the data coming in contains single or multiple messages.

Meta-object-level attributes	Description
ncpdp	Defines the handling for a second mime type through a reference to another type-specific object (in Figure 24, BIA_MO_DataHandlerService_NCPDP).

Data handler meta-object (BIA_MO_DataHandler_Default)

The data handler meta-object is a BIA_MO_DataHandler_Default. This is the top-level object in a hierarchy that stores information used by the designated data handler. This information is distinct from the information stored in the Service Registration object hierarchy.



Pos	Name	Type	Key	Foreign Key	Required	Cardinality	Maximum Length	Default Value	Application Specific Information
1	hl7	BIA_MO_DataHandler_HL7	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
2	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
3			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 25. BIA_MO_DataHandlerDefault in the Business Object Designer

The pertinent attribute of the object is as follows:

Meta-object-level attribute	Description
hl7	Defines the handling for a mime type, as designated in the connector configuration file in the DataHandlerMimeType attribute. The referenced object (in Figure 25, BIA_MO_DataHandler_HL7) contains type specific information used by the data handler.

PIMO framework meta-objects

This section describes the PIMO meta-objects.

BIA_MO_Tcpip_MapSubscriptions

Processing in the PIMO is based on a series of transformations ordered by a set of maps. At the top of the map hierarchy is the BIA_MO_Tcpip_MapSubscriptions object.

Figure 26 is an example of what the BIA_MO_Tcpip_MapSubscriptions object looks like in the Business Object Designer.

General		Attributes				
	Pos	Name	Type	Key	Foreign	Required
1	1	Inbound	BIA_MO_Tcpip_MapSubscriptions_In	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1.1	1.1	BIA_InputMessage	BIA_Map_InputMessage_to_LLPMessagesList	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1.2	1.2	BIA_InputMessage_CheckComplete	BIA_Map_InputMessage_to_InputMessage	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1.3	1.3	BIA_ApplicationMessage	BIA_Map_ApplicationMessage_to_FinalMessage	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
1.4	1.4	ObjectEventId	String			
2	2	Outbound	BIA_MO_Tcpip_MapSubscriptions_Out	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2.1	2.1	BIA_ApplicationMessage	BIA_Map_ApplicationMessage_to_InputMessage	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2.2	2.2	BIA_ResponseMessage	BIA_Map_ResponseMessage_to_FinalMessage	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2.3	2.3	ObjectEventId	String			
3	3	ObjectEventId	String			
4	4			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 26. BIA_MO_Tcpip_MapSubscriptions in the Business Object Designer

The pertinent attributes of the example object are as follows:

MO Level Attributes	Description
Inbound	The set of maps used by PIMO in event mode processing. In Figure 26, three maps are used.
+BIA_InputMessage_CheckComplete	Child attribute, which contains the map BIA_Map_InputMessage_to_InputMessage. See Figure 29 on page 37 for information about the requirements for BIA_InputMessage_CheckComplete.
+BIA_InputMessage	Child attribute, which contains the map BIA_Map_InputMessage_to_LLPMessagesList, used to remove LLP headers and trailers from the main HL7 data. See Figure 28 on page 36 for information about the requirements for BIA_InputMessage.
+BIA_ApplicationMessage	Child attribute, which contains a map BIA_Map_ApplicationMessage_to_FinalMessage. See Figure 30 on page 37 for information about the requirements for BIA_ApplicationMessage.
Outbound	The set of maps used by PIMO in service call request processing. In Figure 26, two maps are used.
+BIA_ApplicationMessage	Child attribute, which contains the map BIA_Map_ApplicationMessage_to_InputMessage. See Figure 31 on page 38 for information about the requirements for BIA_ApplicationMessage.
+BIA_ResponseMessage	Child attribute, which contains the map BIA_Map_ResponseMessage_to_FinalMessage. See Figure 32 on page 38 for information about the requirements for BIA_ResponseMessage.

BIA_Map_InputMessage_to_LLPMessagesList

This section describes the attributes and ASI that are used in PIMO maps. A specific map instance--BIA_Map_InputMessage_to_LLPMessagesList--is used as an example.

General		Attributes								
Pos		Name	Type	Key	Foreign	Required	Cardinal	Maxi	De	Application Specific Information
1	1	Port	BIA_Map_InputMessage_to_LLPMessagesList_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
1.1	1.1	IPort	BIA_InputMessage	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
1.1.1	1.1.1	CharSet	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		
1.1.2	1.1.2	Content	BIA_ContentBO	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	N			
1.1.3	1.1.3	ObjectEventId	String							
1.2	1.2	OPort	BIA_LLPMessagesList	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
1.2.1	1.2.1	LLPMessages	BIA_LLPMessages	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	N			
1.2.2	1.2.2	ObjectEventId	String							
1.3	1.3	ObjectEventId	String							
2	2	Declaration	BIA_Map_InputMessage_to_LLPMessagesList_Declaration	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
2.1	2.1	DummyKey	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		
2.2	2.2	contentText	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		
2.3	2.3	ObjectEventId	String							
3	3	Action	BIA_Map_InputMessage_to_LLPMessagesList_Action	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1			
3.1	3.1	Action1	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		type=NativeStatic;class=com.ibm.adapters.tcpip.messagehandlers.LLPMessagingProtocolHandler;method=parseInputMessageToLLPMessages;target=contentText;IPort;Oport
3.2	3.2	ObjectEventId	String							
4	4	ObjectEventId	String							
5	5			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		

Figure 27. BIA_Map_InputMessage_to_LLPMessagesList in the Business Object Designer

The pertinent attributes of the object are as follows:

Map BO Level Attributes	Description
Port	Each PIMO map has two ports: the IPort and the OPort, defined in the example by the BIA_InputMessage_to_LLPMessagesList_Port. These indicate the expected type of the source object (in the example, a BIA_InputMessage) and the destination object (in the example, a BIA_LLPMessagesList).
Declaration	A PIMO map may include declaration objects containing names for temporary variables to be used during processing. In the example, the declaration object (a BIA_Map_InputMessage_to_LLPMessagesList_Declaration) contains one such name, "contentText"
Action	Each PIMO map has at least one action object (here a BIA_Map_InputMessage_to_LLPMessagesList) that stores information pointing to the actual class and method that does the processing

The ASI contains the following information:

```
type=NativeStatic;
class=com.ibm.adapters.tcpip.messagehandlers.LLPMessagingProtocolHandler;
method=parseInputMessageToLLPMessages; target=contentText;IPort;Oport
```

type = In this release, this value is always nativeStatic. The present PIMO structure only supports public static methods.

- class =**
This value is the fully qualified name of the Java class that contains the method. In the example this is `com.ibm.adapters.tcpip.messagehandlers.LLPMessagingProtocolHandler`.
- method =**
This value is the method to be called. The example is `parseInputMessageToLLPMessages`.
- target =**
This value is where returned data should be stored. In the example, the data is to be stored in the variable `contentText` created in the declaration attribute.
- var1, var2 . . . varx**
An open list of parameters to be passed into the method. The order and number of variables on this list must exactly match the order and number of parameters that the method expects. In the example, `var1` is the `IPort` business object and `var2` is the `OPort` business object from the `Port` attribute.

Map usage rules

This section describes rules you should follow when using maps.

General rules

The following rules apply to all maps:

- The name of the attribute should be the iPort of the map, where map signifies the data type of the attribute.
- To achieve message handler chaining, the oPort of the previous map should match the iPort of the next map.

Inbound rules

The following rules apply to the Inbound attribute:

- The first attribute in Inbound should be `BIA_InputMessage`, because incoming event data is always wrapped by the adapter in the `BIA_InputMessage` business object.

For this map, the iPort has to be `BIA_InputMessage`. The oPort can be any business object with a `Content` attribute of type `BIA_ContentBO`.

General		Attributes		
Pos	Name	Type	Key	
1	Inbound	BIA_MO_Tcpip_MapSubscriptions_In	<input checked="" type="checkbox"/>	
.1	BIA_InputMessage	BIA_Map_InputMessage_to_LLPMessagesList	<input checked="" type="checkbox"/>	
.1.1	Port	BIA_Map_InputMessage_to_LLPMessagesList_Port	<input checked="" type="checkbox"/>	
.1.1.1	IPort	BIA_InputMessage	<input checked="" type="checkbox"/>	
.1.1.1	OPort	BIA_LLPMessagesList	<input type="checkbox"/>	

Figure 28. `BIA_InputMessage`

- If `supportMultipleMessages` is set to `true` in `BIA_MO_DataHandlerService`, there needs to be an attribute `BIA_InputMessage_CheckComplete` of type `BIA_Map_InputMessage_to_InputMessage`. This attribute invokes the `MessageHandler` method `checkForCompleteInputMessage()` to separate complete messages from incomplete ones in the same event data (one socket read).

For this map, the iPort has to be BIA_InputMessage. The oPort can be any business object with a Content attribute of type BIA_ContentBO.

General		Attributes			
	Pos	Name	Type	Key	Foreign
	1	Inbound	BIA_MO_Tcpsip_MapSubscriptions_In	<input checked="" type="checkbox"/>	<input type="checkbox"/>
1.1	1.1	BIA_InputMessage	BIA_Map_InputMessage_to_LLPMessagesList	<input checked="" type="checkbox"/>	<input type="checkbox"/>
1.2	1.2	BIA_InputMessage_CheckComplete	BIA_Map_InputMessage_to_InputMessage	<input type="checkbox"/>	<input type="checkbox"/>
1.2.1	1.2.1	Port	BIA_Map_InputMessage_to_InputMessage_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>
1.2.1	1.2.1	iPort	BIA_InputMessage	<input checked="" type="checkbox"/>	<input type="checkbox"/>
1.2.1	1.2.1	oPort	BIA_InputMessage	<input type="checkbox"/>	<input type="checkbox"/>

Figure 29. BIA_InputMessage_CheckComplete

- For synchronous event handling, one attribute should be BIA_ApplicationMessage. The business object returned from the broker as the result of an executeCollaboration() call is first subjected to the data handler, and the output application data from the data handler is wrapped in BIA_ApplicationMessage. The map signified by the data type of BIA_ApplicationMessage is invoked for message formatting.

For this map, the iPort has to be BIA_ApplicationMessage. The oPort can be any business object with a Content attribute of type BIA_ContentBO.

General		Attributes			
	Pos	Name	Type	Key	Foreign
	1	Inbound	BIA_MO_Tcpsip_MapSubscriptions_In	<input checked="" type="checkbox"/>	<input type="checkbox"/>
1.1	1.1	BIA_InputMessage	BIA_Map_InputMessage_to_LLPMessagesList	<input checked="" type="checkbox"/>	<input type="checkbox"/>
1.2	1.2	BIA_InputMessage_CheckComplete	BIA_Map_InputMessage_to_InputMessage	<input type="checkbox"/>	<input type="checkbox"/>
1.3	1.3	BIA_ApplicationMessage	BIA_Map_ApplicationMessage_to_FinalMessage	<input type="checkbox"/>	<input type="checkbox"/>
1.3.1	1.3.1	Port	BIA_Map_ApplicationMessage_to_FinalMessage_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>
1.3.1	1.3.1	iPort	BIA_ApplicationMessage	<input checked="" type="checkbox"/>	<input type="checkbox"/>
1.3.1	1.3.1	oPort	BIA_FinalMessage	<input type="checkbox"/>	<input type="checkbox"/>

Figure 30. BIA_ApplicationMessage (inbound)

Outbound rules

The following rules apply to the Outbound attribute:

- If maps are needed, the name of one attribute should be BIA_ApplicationMessage. The business object from the broker during request processing is converted to application data by the data handler. This data is wrapped in the content attribute of BIA_ApplicationMessage. The map signified by the data type of BIA_ApplicationMessage will be invoked on it.

For this map, the iPort has to be BIA_ApplicationMessage. The oPort can be any business object with a Content attribute of type BIA_ContentBO.

General		Attributes		
	Pos	Name	Type	Key
	1	<input type="checkbox"/> Inbound	BIA_MO_Tcpip_MapSubscriptions_In	<input checked="" type="checkbox"/>
1.1	1.1	<input type="checkbox"/> BIA_InputMessage	BIA_Map_InputMessage_to_LLPMessagesList	<input checked="" type="checkbox"/>
1.2	1.2	<input type="checkbox"/> BIA_InputMessage_CheckComplete	BIA_Map_InputMessage_to_InputMessage	<input type="checkbox"/>
1.3	1.3	<input type="checkbox"/> BIA_ApplicationMessage	BIA_Map_ApplicationMessage_to_FinalMessage	<input type="checkbox"/>
1.4	1.4	ObjectEventId	String	
	2	<input type="checkbox"/> Outbound	BIA_MO_Tcpip_MapSubscriptions_Out	<input type="checkbox"/>
2.1	2.1	<input type="checkbox"/> BIA_ApplicationMessage	BIA_Map_ApplicationMessage_to_InputMessage	<input checked="" type="checkbox"/>
2.1.1	2.1.1	<input type="checkbox"/> Port	BIA_Map_ApplicationMessage_to_InputMessage_Port	<input checked="" type="checkbox"/>
2.1.1	2.1.1	<input type="checkbox"/> iPort	BIA_ApplicationMessage	<input checked="" type="checkbox"/>
2.1.1	2.1.1	<input type="checkbox"/> OPort	BIA_InputMessage	<input type="checkbox"/>

Figure 31. BIA_ApplicationMessage (outbound)

- If maps are needed, the name of the attribute to support synchronous processing should be BIA_ResponseMessage. The response coming from the external application to the synchronous request is wrapped in the BIA_ResponseMessage business object Content attribute. The map signified by the data type of BIA_ReponseMessage is invoked for message formatting.

For this map, the iPort has to be BIA_ResponseMessage. The oPort can be any business object with a Content attribute of type BIA_ContentBO. Figure 32 shows the oPort as BIA_FinalMessage. The oPort should have other attributes, such as ResponseAttributeName, ResponseDataMimeType, Status, and CharSet, as in BIA_FinalMessage.

General		Attributes		
	Pos	Name	Type	Key
	1	<input type="checkbox"/> Inbound	BIA_MO_Tcpip_MapSubscriptions_In	<input checked="" type="checkbox"/>
1.1	1.1	<input type="checkbox"/> BIA_InputMessage	BIA_Map_InputMessage_to_LLPMessagesList	<input checked="" type="checkbox"/>
1.2	1.2	<input type="checkbox"/> BIA_InputMessage_CheckComplete	BIA_Map_InputMessage_to_InputMessage	<input type="checkbox"/>
1.3	1.3	<input type="checkbox"/> BIA_ApplicationMessage	BIA_Map_ApplicationMessage_to_FinalMessage	<input type="checkbox"/>
1.4	1.4	ObjectEventId	String	
	2	<input type="checkbox"/> Outbound	BIA_MO_Tcpip_MapSubscriptions_Out	<input type="checkbox"/>
2.1	2.1	<input type="checkbox"/> BIA_ApplicationMessage	BIA_Map_ApplicationMessage_to_InputMessage	<input checked="" type="checkbox"/>
2.2	2.2	<input type="checkbox"/> BIA_ResponseMessage	BIA_Map_ResponseMessage_to_FinalMessage	<input type="checkbox"/>
2.2.1	2.2.1	<input type="checkbox"/> Port	BIA_Map_ResponseMessage_to_FinalMessage_Port	<input checked="" type="checkbox"/>
2.2.1	2.2.1	<input type="checkbox"/> iPort	BIA_ResponseMessage	<input checked="" type="checkbox"/>
2.2.1	2.2.1	<input type="checkbox"/> OPort	BIA_FinalMessage	<input type="checkbox"/>

Figure 32. BIA_ResponseMessage

Bypassing all maps

In asynchronous request processing, you can bypass all maps by setting up BIA_MO_Tcpip_MapSubscriptions so that the Inbound and Outbound attributes are empty (without any children).

Note: For synchronous request processing, the map BIA_Map_ResponseMessage_to_FinalMessage must be plugged in, so you cannot bypass maps in synchronous request processing.

Chapter 4. Configuring the connector

The Connector Configurator is a tool supplied with the Adapter for TCP/IP that allows you to configure the supplied connector template.

This chapter describes how to install and configure the adapter using Connector Configurator.

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
- 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

If your adapter supports DB2^(R) Information Integrator, use the WMQI options and the DB2 II standard properties (see the Notes column in Appendix A, “Standard configuration properties.”)

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 WebSphere InterChange Server, 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 40).

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 41 to set up a new one.

Running connectors on UNIX

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.

Some properties in the Connector Configurator use directory paths, which default to the Windows convention for directory paths. If you use the configuration file in a UNIX environment, revise the directory paths to match the UNIX convention for these paths. Select the target operating system in the toolbar drop-list so that the correct operating system rules are used for extended validation.

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 without running System Manager and work with connector configuration files, irrespective of your broker.

To do so:

- From **Start>Programs**, click **IBM WebSphere Business Integration Adapters>IBM WebSphere Business Integration Toolset>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 46.)

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 41.
- 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
 - Property Subtype
 - Updated Method
 - Description
- **Flags**
 - Standard flags
- **Custom Flag**
 - Flag

The **Property Subtype** can be selected when **Property Type** is a String. It is an optional value which provides syntax checking when you save the configuration file. The default is a blank space, and means that the property has not been subtyped.

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. Right-click on the square to the left of the Value column heading.
2. From the pop-up menu, select **Add** to display the Property Value dialog box. Depending on the property type, the dialog box allows you to enter either a value, or both a value and a range.
3. Enter 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.

Setting path names

Some general rules for setting path names are:

- The maximum length of a file name in Windows and UNIX is 255 characters.
- In Windows, the absolute path name must follow the format `[Drive:][Directory]\filename` (for example, `C:\WebSphereAdapters\bin\Data\Std\StdConnProps.xml`).
In UNIX the first character should be `/`.
- Queue names may not have leading or embedded spaces.

Creating a new configuration file

When you create a new configuration file, you must name it and select an integration broker.

You also select an operating system for extended validation on the file. The toolbar has a droplist called **Target System** that allows you to select the target operating system for extended validation of the properties. The available options are:

Windows, UNIX, Other (if not Windows or UNIX), and None-no extended validation (switches off extended validation). The default on startup is Windows.

To start Connector Configurator:

- In the System Manager window, select **Connector Configurator** from the **Tools** menu. Connector Configurator opens.
- In stand-alone mode, launch Connector Configurator.

To set the operating system for extended validation of the configuration file:

- Pull down the **Target System:** droplist on the menu bar.
- Select the operating system you are running on.

Then 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.
- Complete 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. Set the operating system for extended validation of the configuration file using the **Target System:** droplist on the menu bar (see “Creating a new configuration file” on page 43).
2. Click **File>New>Connector Configuration**.
3. 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**.
4. 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 complete the definition now, or you can save the file and complete the fields later.

5. 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.
6. 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).
- A WebSphere InterChange Server repository file.
Definitions used in a previous WebSphere InterChange Server 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)
 - WebSphere InterChange Server Repository (*.in, *.out)
Choose this option if a repository file was used to configure the connector in a WebSphere InterChange Server 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 connector properties associated with the selected broker. The table shows **Property name**, **Value**, **Type**, **Subtype** (if the Type is a string), **Description**, and **Update Method**.
3. You can save the file now or complete the remaining configuration fields, as described in “Specifying supported business object definitions” on page 49.
4. 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 you created the configuration file, you used the **Target System** droplist that allows you to select the target operating system for extended validation of the properties.

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.

If you have elected to use the extended validation feature by selecting a value of Windows, UNIX or Other from the **Target System** droplist, the system will validate the property subtype as well as the type, and it displays a warning message if the validation fails.

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 WebSphere InterChange Server, values for these properties are also required:

- Associated Maps
- Resources
- Messaging (where applicable)
- Security

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, “Standard configuration properties,” on page 65. 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.

Note: If the property has a Type of String, it may have a subtype value in the Subtype column. This subtype is used for extended validation of the property.

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.

To get more information on a particular standard property, left-click the entry in the Description column for that property in the Standard Properties tabbed sheet. If you have Extended Help installed, an arrow button will appear on the right. When you click on the button, a Help window will open and display details of the standard property.

Note: If the hot button does not appear, no Extended Help was found for that property.

If installed, the Extended Help files are located in
<ProductDir>\bin\Data\Std\Help\<RegionalSetting>\.

Setting connector-specific configuration properties

For connector-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.

Note: If the property has a Type of String, you can select a subtype from the Subtype droplist. This subtype is used for extended validation of the property.

3. To encrypt a property, select the **Encrypt** box.
4. To get more information on a particular property, left-click the entry in the Description column for that property. If you have Extended Help installed, a hot button will appear. When you click on the hot button, a Help window will open and display details of the standard property.

Note: If the hot button does not appear, no Extended Help was found for that property.

5. Choose to save or discard changes, as described for “Setting standard connector properties” on page 47.

If the Extended Help files are installed and the AdapterHelpName property is blank, Connector Configurator will point to the adapter-specific Extended Help files located in <ProductDir>\bin\Data\App\Help\<RegionalSetting>\. Otherwise,

Connector Configurator will point to the adapter-specific Extended Help files located in
<ProductDir>\bin\Data\App\Help\<AdapterHelpName>\<RegionalSetting>\. See “AdapterHelpName” on page 72.

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 Appendix A, “Standard configuration properties.”

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 WebSphere InterChange Server 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 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 WebSphere Application Server 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

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 Binding**

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 WebSphere InterChange Server 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 WebSphere InterChange Server.
5. Reboot the server for the changes to take effect.

Resources

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

The **Messaging** tab enables you to configure messaging properties. The messaging properties are available only if you have set MQ as the value of the `DeliveryTransport` standard property and WebSphere InterChange Server as the broker type. These properties affect how your connector will use queues.

Validating messaging queues

Before you can validate a messaging queue, you must:

- Make sure that WebSphere MQ Series is installed.
- Create a messaging queue with channel and port on the host machine.
- Set up a connection to the host machine.

To validate the queue, use the **Validate** button to the right of the **Messaging Type** and **Host Name** fields on the **Messaging** tab.

Security

You can use the **Security** tab in Connector Configurator to set various privacy levels for a message. You can only use this feature when the `DeliveryTransport` property is set to JMS.

By default, **Privacy** is turned off. Check the **Privacy** box to enable it.

The **Keystore Target System Absolute Pathname** is:

- For Windows:
`<ProductDir>\connectors\security\<connectorname>.jks`
- For UNIX:
`opt/IBM/WebSphereAdapters/connectors/security/<connectorname>.jks`

This path and file should be on the system where you plan to start the connector, that is, the target system.

You can use the **Browse** button at the right only if the target system is the one currently running. It is greyed out unless **Privacy** is enabled and the **Target System** in the menu bar is set to Windows.

The **Message Privacy Level** may be set as follows for the three messages categories (All Messages, All Administrative Messages, and All Business Object Messages):

- "" is the default; used when no privacy levels for a message category have been set.

- none
Not the same as the default: use this to deliberately set a privacy level of none for a message category.
- integrity
- privacy
- integrity_plus_privacy

The **Key Maintenance** feature lets you generate, import and export public keys for the server and adapter.

- When you select **Generate Keys**, the Generate Keys dialog box appears with the defaults for the keytool that will generate the keys.
- The keystore value defaults to the value you entered in **Keystore Target System Absolute Pathname** on the Security tab.
- When you select OK, the entries are validated, the key certificate is generated, and the output is sent to the Connector Configurator log window.

Before you can import a certificate into the adapter keystore, you must export it from the server keystore. When you select **Export Adapter Public Key**, the Export Adapter Public Key dialog box appears.

- The export certificate defaults to the same value as the keystore, except that the file extension is <filename>.cer.

When you select **Import Server Public Key**, the Import Server Public Key dialog box appears.

- The import certificate defaults to <ProductDir>\bin\ics.cer (if the file exists on the system).
- The import Certificate Association should be the server name. If a server is registered, you can select it from the droplist.

The **Adapter Access Control** feature is enabled only when the value of DeliveryTransport is IDL. By default, the adapter logs in with the guest identity. If the **Use guest identity** box is not checked, the **Adapter Identity** and **Adapter Password** fields are enabled.

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 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 WebSphere InterChange Server: *Implementation Guide for WebSphere InterChange Server*
- For WebSphere Message Brokers: *Implementing Adapters with WebSphere Message Brokers*
- For WebSphere Application Server: *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):

1. Open the existing configuration file in Connector Configurator.
2. Select the **Standard Properties** tab.
3. 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 in the properties window 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>
```

Chapter 5. Running the connector

This chapter describes the steps required to start and stop the connector and to run multiple instances of the connector on the same machine. It contains the following sections:

- “Starting the connector”
- “Stopping the connector” on page 58
- “Creating multiple instances of connectors on one server” on page 59

Starting the connector

A connector must be explicitly started using its **connector start-up script**. On Windows systems the startup script should reside in the connector’s runtime directory:

ProductDir\connectors*connName*

where *connName* identifies the connector.

On UNIX systems the startup script should reside in the *UNIX ProductDir/bin* directory.

The name of the startup script depends on the operating-system platform, as Table 4 shows.

Table 4. Startup scripts for a connector

Operating system	Startup script
UNIX-based systems	connector_manager
Windows	start_ <i>connName</i> .bat

When the startup script runs, it expects by default to find the configuration file in the *Productdir* (see the commands below). This is where you place your configuration file.

Note: You need a local configuration file if the adapter is using JMS transport.

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 [-cconfigFile ]
```

- On UNIX-based systems:

```
connector_manager -start connName brokerName [-cconfigFile ]
```

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, which is launched when you start System Manager running with the WebSphere Application Server or InterChange Server broker: You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Manager (available for all brokers): 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.
 - When using InterChange Server 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.

Creating multiple instances of connectors on one server

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 59.
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*.

Chapter 6. Maintaining the connector

This chapter describes how the adapter handles errors. The chapter contains the following sections:

- “Error handling in the connector”
- “Tracing messages” on page 62

Error handling in the connector

The connector logs any abnormal condition that it encounters during processing, regardless of the trace level. It writes the error text to the connector log file. See “Setting trace/log file values” on page 53 for information on specifying log information.

The message contains a detailed description of the condition and the outcome and may also include extra information that may aid in debugging, such as business object dumps or stack traces (for exceptions).

For a complete list of error messages, refer to the `BIA_TCPIPConnector.txt` message file installed in the `ProductDir\connectors\messages` directory. Table 5 describes some of the more common errors and how the connector handles those errors.

Table 5. Connector errors

Error description	Error type	Error handling	Steps to correct
Mandatory CFG properties not populated	Fatal	Error logged and connector terminates	Make sure properties are populated
Pre-requisite BO definitions not in repository	Fatal	Error logged and connector terminates	Make sure all .xsd files are in the repository
Only Client or only Server is configured in CFG	Warning	Warning logged and the connector processes only requests or events, respectively	If both functions are desired, make sure all properties in CFG are configured
CFG properties are of inappropriate type (for example, negative values where positive are required)	Fatal	Error logged and connector terminates	Make sure properties are of the appropriate type
Server that is configured for requests not available	Error	The request fails, but the connector does not terminate	Make sure a) the application server is running and b) the remote host is available from the connector machine
Socket on server configured for request times out before request sent.	Error	Socket closure error logged. Connector continues. Failure status sent to broker.	See above

Table 5. Connector errors (continued)

Error description	Error type	Error handling	Steps to correct
Socket closes before complete event is received	Error	Error logged, and connector continues	Do not close socket immediately after sending data to the connector
Bind exception trying to bind socket to local port	Fatal	Error logged, and connector terminates	Make sure port is free
Maps specified in map MO missing	Error	Error logged, and connector terminates	Make sure map .xsd files available in repository
PIMO level error	Error	PIMO infrastructure logs error and connector terminates	Make sure that the maps and the related action are configured properly

Tracing messages

Tracing is an optional debugging feature you can turn on to closely follow the connector's behavior. Tracing messages are configurable and can be changed dynamically. You set various levels depending on the desired detail. See "Setting trace/log file values" on page 53 for information on specifying tracing information.

The following sections describe tracing for the TCP/IP adapter.

Recommendation: Tracing should be turned off on a production system or set to the lowest possible level to improve performance and decrease file size.

Using tracing with the connector

Trace messages, by default, are written to STDOUT (screen). You can also configure tracing to write to a file.

Table 6 describes the types of tracing messages that the connector outputs at each trace level. These messages are in addition to any tracing messages output by the IBM WebSphere Business Integration Adapter architecture.

Table 6. Tracing messages for the connector

Tracing level	Tracing messages
Level 0	Trace adapter version
Level 1	<ul style="list-style-type: none"> Trace each time pollForEvents method is called. Trace each time the adapter accepts a new socket connection in TCP/IP server mode. Trace each time the adapter attempts a new socket connection in TCP/IP client mode. Trace ASBO/ISBO name created by adapter to deliver to broker. Trace request ASBO/ISBO name created by broker to deliver to adapter.
Level 2	<ul style="list-style-type: none"> Trace each time doVerbFor () is called. Trace which Protocol Handler is processing this request. Trace each time executeCollab() or gotApplEvent() is called.

Table 6. Tracing messages for the connector (continued)

Tracing level	Tracing messages
Level 3	<ul style="list-style-type: none">• Trace important ASI of the business object being processed.• Trace important attributes of the business object being processed.• Trace all data that passes through the TCP channel in both directions. Note: The data will be formatted as hexadecimal.
Level 4	<ul style="list-style-type: none">• Trace spawning of threads.• Trace all ASI processed.• Trace entry and exit of important functions.
Level 5	<ul style="list-style-type: none">• Trace entry and exit for each important method.• Trace all the Adapter properties.• Trace dumps of business object sent to the broker.• Trace dumps of business object sent by the broker.

Appendix A. Standard configuration properties

This appendix describes the standard configuration properties for the connector component of WebSphere Business Integration adapters. The information covers connectors running with the following integration brokers:

- WebSphere InterChange Server
- WebSphere MQ Integrator, WebSphere MQ Integrator Broker, and WebSphere Business Integration Message Broker, collectively referred to as the WebSphere Message Brokers (and shown as WMQI in the Connector Configurator).
- Information Integrator
- WebSphere Application Server

If your adapter supports DB2 Information Integrator, use the WMQI options and the DB2 II standard properties (see the Notes column in Table 7 on page 67.)

The properties you set for the adapter depend on which integration broker you use. You choose the integration broker using Connector Configurator. After you choose the broker, Connector Configurator lists the standard properties you must configure for the adapter.

For information about properties specific to this connector, see the relevant section in this guide.

Standard connector properties overview

Connectors have two types of configuration properties:

- Standard configuration properties, which are used by the framework
- Application, or connector-specific, configuration properties, which are used by the agent

These properties determine the adapter framework and the agent run-time behavior.

This section describes how to start Connector Configurator and describes characteristics common to all properties. For information on configuration properties specific to a connector, see its adapter user guide.

Starting 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.

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.

Configuration property values overview

The connector uses the following order to determine a property's value:

1. Default
2. Repository (valid only if WebSphere InterChange Server is the integration broker)
3. Local configuration file
4. Command line

The default length of a property field is 255 characters. There is no limit on the length of a STRING property type. The length of an INTEGER type is determined by the server on which the adapter is running.

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.

The update characteristics of a property, that is, how and when a change to the connector properties takes effect, depend on the nature of the property.

There are four update methods for standard connector properties:

- **Dynamic**
The new value takes effect immediately after the change is saved in System Manager. However, if the connector is in stand-alone mode (independently of System Manager), for example, if it is running with one of the WebSphere message brokers, you can change properties only through the configuration file. In this case, a dynamic update is not possible.
- **Agent restart (WebSphere InterChange Server only)**
The new value takes effect only after you stop and restart the connector agent.
- **Component restart**
The new value takes effect only after the connector is stopped and then restarted in System Manager. You do not need to stop and restart the agent or the server process.
- **System restart**
The new value takes effect only after you stop and restart the connector agent and the server.

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 7 on page 67.

There are three locations in which a standard property can reside. Some properties can reside in more than one location.

- **ReposController**
The property resides in the connector controller and is effective only there. If you change the value on the agent side, it does not affect the controller.
- **ReposAgent**
The property resides in the agent and is effective only there. A local configuration can override this value, depending on the property.
- **LocalConfig**
The property resides in the configuration file for the connector and can act only through the configuration file. The controller cannot change the value of the property, and is not aware of changes made to the configuration file unless the system is redeployed to update the controller explicitly.

Standard properties quick-reference

Table 7 provides a quick-reference to the standard connector configuration properties. Not all connectors require all of these properties, and property settings may differ from integration broker to integration broker.

See the section following the table for a description of each property.

Note: In the Notes column in Table 7, the phrase “RepositoryDirectory is set to <REMOTE>” indicates that the broker is InterChange Server. When the broker is WMQI or WAS, the repository directory is set to <ProductDir>\repository

Table 7. Summary of standard configuration properties

Property name	Possible values	Default value	Update method	Notes
AdapterHelpName	One of the valid subdirectories in <ProductDir>\bin\Data\App\Help\ that contains a valid <RegionalSetting> directory	Template name, if valid, or blank field	Component restart	Supported regional settings. Include chs_chn, cht_twn, deu_deu, esn_esp, fra_fra, ita_ita, jpn_jpn, kor_kor, ptb_bra, and enu_usa (default).
AdminInQueue	Valid JMS queue name	<CONNECTORNAME>/ADMININQUEUE	Component restart	This property is valid only when the value of DeliveryTransport is JMS
AdminOutQueue	Valid JMS queue name	<CONNECTORNAME>/ADMINOUTQUEUE	Component restart	This property is valid only when the value of DeliveryTransport is JMS
AgentConnections	1 through 4	1	Component restart	This property is valid only when the value of DeliveryTransport is MQ or IDL, the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
AgentTraceLevel	0 through 5	0	Dynamic if broker is ICS; otherwise Component restart	
ApplicationName	Application name	The value specified for the connector application name	Component restart	
BiDi.Application	Any valid combination of these bidirectional attributes: 1st letter: I, V 2nd letter: L, R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transforma tion is true

Table 7. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
BiDi.Broker	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transformation is true. If the value of BrokerType is ICS, the property is read-only.
BiDi.Metadata	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transformation is true.
BiDi.Transformation	true or false	false	Component restart	This property is valid only if the value of BrokerType is not WAS.
BrokerType	ICS, WMQI, WAS	ICS	Component restart	
CharacterEncoding	Any supported code. The list shows this subset: ascii7, ascii8, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437 .	ascii7	Component restart	This property is valid only for C++ connectors.
CommonEventInfrastructure	true or false	false	Component restart	
CommonEventInfrastructureURL	A URL string, for example, corbaloc:iop: host:2809.	No default value.	Component restart	This property is valid only if the value of CommonEvent Infrastructure is true.
ConcurrentEventTriggeredFlows	1 through 32,767	1	Component restart	This property is valid only if the value of RepositoryDirectory is set to <REMOTE> and the value of BrokerType is ICS.
ContainerManagedEvents	Blank or JMS	Blank	Component restart	This property is valid only when the value of Delivery Transport is JMS.
ControllerEventSequencing	true or false	true	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
ControllerStoreAndForwardMode	true or false	true	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.

Table 7. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
ControllerTraceLevel	0 through 5	0	Dynamic	This property is valid only if the value of RepositoryDirectory is set to <REMOTE> and the value of BrokerType is ICS.
DeliveryQueue	Any valid JMS queue name	<CONNECTORNAME>/DELIVERYQUEUE	Component restart	This property is valid only when the value of Delivery Transport is JMS.
DeliveryTransport	MQ, IDL, or JMS	IDL when the value of RepositoryDirectory is <REMOTE>, otherwise JMS	Component restart	If the value of RepositoryDirectory is not <REMOTE>, the only valid value for this property is JMS.
DuplicateEventElimination	true or false	false	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
EnableOidForFlowMonitoring	true or false	false	Component restart	This property is valid only if the value of BrokerType is ICS.
FaultQueue	Any valid queue name.	<CONNECTORNAME>/FAULTQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
jms.FactoryClassName	CxCommon.Messaging.jms.IBMMQSeriesFactory, CxCommon.Messaging.jms.SonicMQFactory, or any Java class name	CxCommon.Messaging.jms.IBMMQSeriesFactory	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
jms.ListenerConcurrency	1 through 32767	1	Component restart	This property is valid only if the value of jms.TransportOptimized is true.
jms.MessageBrokerName	If the value of jms.FactoryClassName is IBM, use crossworlds.queue.manager.	crossworlds.queue.manager	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
jms.NumConcurrentRequests	Positive integer	10	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
jms.Password	Any valid password		Component restart	This property is valid only if the value of DeliveryTransport is JMS.
jms.TransportOptimized	true or false	false	Component restart	This property is valid only if the value of DeliveryTransport is JMS and the value of BrokerType is ICS.
jms.UserName	Any valid name		Component restart	This property is valid only if the value of Delivery Transport is JMS.

Table 7. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
JvmMaxHeapSize	Heap size in megabytes	128m	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
JvmMaxNativeStackSize	Size of stack in kilobytes	128k	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
JvmMinHeapSize	Heap size in megabytes	1m	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
ListenerConcurrency	1 through 100	1	Component restart	This property is valid only if the value of DeliveryTransport is MQ.
Locale	This is a subset of the supported locales: en_US, ja_JP, ko_KR, zh_CN, zh_TW, fr_FR, de_DE, it_IT, es_ES, pt_BR	en_US	Component restart	
LogAtInterchangeEnd	true or false	false	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
MaxEventCapacity	1 through 2147483647	2147483647	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
MessageFileName	Valid file name	InterchangeSystem.txt	Component restart	
MonitorQueue	Any valid queue name	<CONNECTORNAME> /MONITORQUEUE	Component restart	This property is valid only if the value of DuplicateEventElimination is true and ContainerManagedEvents has no value.
OADAutoRestartAgent	true or false	false	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
OADMaxNumRetry	A positive integer	1000	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.

Table 7. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
OADRetryTimeInterval	A positive integer in minutes	10	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
PollEndTime	HH = 0 through 23 MM = 0 through 59	HH:MM	Component restart	
PollFrequency	A positive integer (in milliseconds)	10000	Dynamic if broker is ICS; otherwise Component restart	
PollQuantity	1 through 500	1	Agent restart	This property is valid only if the value of ContainerManagedEvents is JMS.
PollStartTime	HH = 0 through 23 MM = 0 through 59	HH:MM	Component restart	
RepositoryDirectory	<REMOTE> if the broker is ICS; otherwise any valid local directory.	For ICS, the value is set to <REMOTE> For WMQI and WAS, the value is <ProductDir \repository	Agent restart	
RequestQueue	Valid JMS queue name	<CONNECTORNAME>/REQUESTQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS
ResponseQueue	Valid JMS queue name	<CONNECTORNAME>/RESPONSEQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
RestartRetryCount	0 through 99	7	Dynamic if ICS; otherwise Component restart	
RestartRetryInterval	A value in minutes from 1 through 2147483647	1	Dynamic if ICS; otherwise Component restart	
ResultSetEnabled	true or false	false	Component restart	Used only by connectors that support DB2II. This property is valid only if the value of DeliveryTransport is JMS, and the value of BrokerType is WMQI.

Table 7. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
ResultsSetSize	Positive integer	0 (means the results set size is unlimited)	Component restart	Used only by connectors that support DB2II. This property is valid only if the value of ResultsSetEnabled is true.
RHF2MessageDomain	mrm or xml	mrm	Component restart	This property is valid only if the value of DeliveryTransport is JMS and the value of WireFormat is CwXML.
SourceQueue	Any valid WebSphere MQ queue name	<CONNECTORNAME>/SOURCEQUEUE	Agent restart	This property is valid only if the value of ContainerManagedEvents is JMS.
SynchronousRequest Queue	Any valid queue name.	<CONNECTORNAME>/SYNCHRONOUSREQUEST QUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
SynchronousRequest Timeout	0 to any number (milliseconds)	0	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
SynchronousResponse Queue	Any valid queue name	<CONNECTORNAME>/SYNCHRONOUSRESPONSE QUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
TivoliMonitorTransaction Performance	true or false	false	Component restart	
WireFormat	CwXML or CwB0	CwXML	Agent restart	The value of this property must be CwXML if the value of RepositoryDirectory is not set to <REMOTE>. The value must be CwB0 if the value of RepositoryDirectory is set to <REMOTE>.
WsifSynchronousRequest Timeout	0 to any number (milliseconds)	0	Component restart	This property is valid only if the value of BrokerType is WAS.
XMLNamespaceFormat	short or long or no	short	Agent restart	This property is valid only if the value of BrokerType is WMQI or WAS

Standard properties

This section describes the standard connector configuration properties.

AdapterHelpName

The AdapterHelpName property is the name of a directory in which connector-specific extended help files are located. The directory must be located in <ProductDir>\bin\Data\App\Help and must contain at least the language directory enu_usa. It may contain other directories according to locale.

The default value is the template name if it is valid, or it is blank.

AdminInQueue

The AdminInQueue property specifies the queue that is used by the integration broker to send administrative messages to the connector.

The default value is `<CONNECTORNAME>/ADMININQUEUE`

AdminOutQueue

The AdminOutQueue property specifies the queue that is used by the connector to send administrative messages to the integration broker.

The default value is `<CONNECTORNAME>/ADMINOUTQUEUE`

AgentConnections

The AgentConnections property controls the number of ORB (Object Request Broker) connections opened when the ORB initializes.

It is valid only if the value of the RepositoryDirectory is set to `<REMOTE>` and the value of the DeliveryTransport property is MQ or IDL.

The default value of this property is 1.

AgentTraceLevel

The AgentTraceLevel property sets the level of trace messages for the application-specific component. The connector delivers all trace messages applicable at the tracing level set and lower.

The default value is 0.

ApplicationName

The ApplicationName property uniquely identifies the name of the connector application. This name is used by the system administrator to monitor the integration environment. This property must have a value before you can run the connector.

The default is the name of the connector.

BiDi.Application

The BiDi.Application property specifies the bidirectional format for data coming from an external application into the adapter in the form of any business object supported by this adapter. The property defines the bidirectional attributes of the application data. These attributes are:

- Type of text: implicit or visual (I or V)
- Text direction: left-to-right or right-to-left (L or R)
- Symmetric swapping: on or off (Y or N)
- Shaping (Arabic): on or off (S or N)
- Numerical shaping (Arabic): Hindi, contextual, or nominal (H, C, or N)

This property is valid only if the BiDi.Transformation property value is set to true.

The default value is ILYNN (implicit, left-to-right, on, off, nominal).

BiDi.Broker

The BiDi.Broker property specifies the bidirectional script format for data sent from the adapter to the integration broker in the form of any supported business object. It defines the bidirectional attributes of the data, which are as listed under BiDi.Application above.

This property is valid only if the BiDi.Transformation property value is set to true. If the BrokerType property is ICS, the property value is read-only.

The default value is ILYNN (implicit, left-to-right, on, off, nominal).

BiDi.Metadata

The BiDi.Metadata property defines the bidirectional format or attributes for the metadata, which is used by the connector to establish and maintain a link to the external application. The attribute settings are specific to each adapter using the bidirectional capabilities. If your adapter supports bidirectional processing, refer to the section on adapter-specific properties for more information.

This property is valid only if the BiDi.Transformation property value is set to true.

The default value is ILYNN (implicit, left-to-right, on, off, nominal).

BiDi.Transformation

The BiDi.Transformation property defines whether or not the system performs a bidirectional transformation at run time.

If the property value is set to true, the BiDi.Application, BiDi.Broker, and BiDi.Metadata properties are available. If the property value is set to false, they are hidden.

The default value is false.

BrokerType

The BrokerType property identifies the integration broker type that you are using. The possible values are ICS, WMQI (for WMQI, WMQIB or WBIMB), or WAS.

CharacterEncoding

The CharacterEncoding property 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. C++ connectors use the value `ascii7` for this property.

By default, only a subset of supported character encodings is displayed. To add other supported values to the list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory (`<ProductDir>`). For more information, see the Connector Configurator appendix in this guide.

CommonEventInfrastructure

The Common Event Infrastructure (CEI) is a simple event management function handling generated events. The CommonEventInfrastructure property specifies whether the CEI should be invoked at run time.

The default value is false.

CommonEventInfrastructureContextURL

The CommonEventInfrastructureContextURL is used to gain access to the WAS server that executes the Common Event Infrastructure (CEI) server application. This property specifies the URL to be used.

This property is valid only if the value of CommonEventInfrastructure is set to true.

The default value is a blank field.

ConcurrentEventTriggeredFlows

The ConcurrentEventTriggeredFlows property determines how many business objects can be concurrently processed by the connector for event delivery. You set the value of this attribute to the number of business objects that are mapped and delivered concurrently. For example, if you set the value of this property to 5, five business objects are processed concurrently.

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), the following properties must be configured:

- The collaboration must be configured to use multiple threads by setting its Maximum number of concurrent events property high enough to use multiple threads.
- The destination application's application-specific component must be configured to process requests concurrently. That is, it must be multithreaded, or it must be able to use connector agent parallelism and be configured for multiple processes. The Parallel Process Degree configuration property must be set to a value larger than 1.

The ConcurrentEventTriggeredFlows property has no effect on connector polling, which is single-threaded and is performed serially.

This property is valid only if the value of the RepositoryDirectory property is set to <REMOTE>.

The default value is 1.

ContainerManagedEvents

The ContainerManagedEvents 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 one JMS transaction.

When this property is set to JMS, the following properties must also be set to enable guaranteed event delivery:

- PollQuantity = 1 to 500

- `SourceQueue = /SOURCEQUEUE`

You must also configure a data handler with the `MimeType` and `DHClass` (data handler class) properties. You can also add `DataHandlerConfigMOName` (the meta-object name, which is optional). To set those values, use the **Data Handler** tab in Connector Configurator.

Although these properties are adapter-specific, here are some example values:

- `MimeType = text/xml`
- `DHClass = com.crossworlds.DataHandlers.text.xml`
- `DataHandlerConfigMOName = MO_DataHandler_Default`

The fields for these values in the **Data Handler** tab are displayed only if you have set the `ContainerManagedEvents` property to the value `JMS`.

Note: When `ContainerManagedEvents` is set to `JMS`, the connector does not call its `pollForEvents()` method, thereby disabling that method's functionality.

The `ContainerManagedEvents` property is valid only if the value of the `DeliveryTransport` property is set to `JMS`.

There is no default value.

ControllerEventSequencing

The `ControllerEventSequencing` property enables event sequencing in the connector controller.

This property is valid only if the value of the `RepositoryDirectory` property is set to `<REMOTE>` (`BrokerType` is `ICS`).

The default value is `true`.

ControllerStoreAndForwardMode

The `ControllerStoreAndForwardMode` property 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 WebSphere InterChange Server, 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.

This property is valid only if the value of the `RepositoryDirectory` property is set to `<REMOTE>` (the value of the `BrokerType` property is `ICS`).

The default value is true.

ControllerTraceLevel

The ControllerTraceLevel property sets the level of trace messages for the connector controller.

This property is valid only if the value of the RepositoryDirectory property is set to set to <REMOTE>.

The default value is 0.

DeliveryQueue

The DeliveryQueue property defines the queue that is used by the connector to send business objects to the integration broker.

This property is valid only if the value of the DeliveryTransport property is set to JMS.

The default value is <CONNECTORNAME>/DELIVERYQUEUE.

DeliveryTransport

The DeliveryTransport property 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 value of the RepositoryDirectory property is set to <REMOTE>, the value of the DeliveryTransport property can be MQ, IDL, or JMS, and the default is IDL.
- If the value of the RepositoryDirectory property is a local directory, the value can be only JMS.

The connector sends service-call requests and administrative messages over CORBA IIOP if the value of the RepositoryDirectory property is MQ or IDL.

If the value of the DeliveryTransport property is MQ, you can set the command-line parameter WhenServerAbsent in the adapter start script to indicate whether the adapter should pause or shut down when the InterChange Server is shut down.

- Enter WhenServerAbsent=pause to pause the adapter when WebSphere InterChange Server is not available.
- Enter WhenServerAbsent=shutdown to shut down the adapter when WebSphere InterChange Server is not available.

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 prevents writing 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 polling thread picks up an event, places it in the connector queue, then picks up the next event. This is faster than IDL, which requires the connector polling thread to pick up an event, go across the network into the server process, store the event persistently in the repository database, then pick up the next event.

JMS

The JMS transport mechanism 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` are listed in Connector Configurator. The properties `jms.MessageBrokerName` and `jms.FactoryClassName` are required for this transport.

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
- WebSphere InterChange Server 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 768MB of process heap size, set the following variable and property:

- Set the `LDR_CNTRL` environment variable in the `CWSharedEnv.sh` script.

This script is located in the `\bin` directory below the product directory (`<ProductDir>`). Using 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 larger than this limit, page swapping can occur, which can adversely affect the performance of your system.

- Set the value of the `IPCCBaseAddress` property to 11 or 12. For more information on this property, see the *System Installation Guide for UNIX*.

DuplicateEventElimination

When the value of this property is `true`, a JMS-enabled connector can ensure that duplicate events are not delivered to the delivery queue. To use this feature, during connector development, the connector must have a unique event identifier set as the business object `ObjectEventId` attribute in the application-specific code.

Note: When the value of this property is `true`, the `MonitorQueue` property must be enabled to provide guaranteed event delivery.

The default value is `false`.

EnableOidForFlowMonitoring

When the value of this property is `true`, the adapter runtime will mark the incoming `ObjectEventID` as a foreign key for flow monitoring.

This property is only valid if the `BrokerType` property is set to `ICS`.

The default value is false.

FaultQueue

If the connector experiences an error while processing a message, it moves the message (and a status indicator and description of the problem) to the queue specified in the FaultQueue property.

The default value is `<CONNECTORNAME>/FAULTQUEUE`.

jms.FactoryClassName

The `jms.FactoryClassName` property specifies the class name to instantiate for a JMS provider. This property must be set if the value of the `DeliveryTransport` property is JMS.

The default is `CxCommon.Messaging.jms.IBMMQSeriesFactory`.

jms.ListenerConcurrency

The `jms.ListenerConcurrency` property specifies the number of concurrent listeners for the JMS controller. It specifies the number of threads that fetch and process messages concurrently within a controller.

This property is valid only if the value of the `jms.OptimizedTransport` property is true.

The default value is 1.

jms.MessageBrokerName

The `jms.MessageBrokerName` specifies the broker name to use for the JMS provider. You must set this connector property if you specify JMS as the delivery transport mechanism (in the `DeliveryTransport` property).

When you connect to a remote message broker, this property requires the following values:

QueueMgrName:Channel:HostName:PortNumber

where:

QueueMgrName is the name of the queue manager.

Channel is the channel used by the client.

HostName is the name of the machine where the queue manager is to reside.

PortNumber is the port number used by the queue manager for listening

For example:

```
jms.MessageBrokerName = WBIMB.Queue.Manager:CHANNEL1:RemoteMachine:1456
```

The default value is `crossworlds.queue.manager`. Use the default when connecting to a local message broker.

jms.NumConcurrentRequests

The `jms.NumConcurrentRequests` property 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 are blocked and must wait for another request to complete before proceeding.

The default value is 10.

jms.Password

The `jms.Password` property specifies the password for the JMS provider. A value for this property is optional.

There is no default value.

jms.TransportOptimized

The `jms.TransportOptimized` property determines if the WIP (work in progress) is optimized. You must have a WebSphere MQ provider to optimize the WIP. For optimized WIP to operate, the messaging provider must be able to:

1. Read a message without taking it off the queue
2. Delete a message with a specific ID without transferring the entire message to the receiver's memory space
3. Read a message by using a specific ID (needed for recovery purposes)
4. Track the point at which events that have not been read appear.

The JMS APIs cannot be used for optimized WIP because they do not meet conditions 2 and 4 above, but the MQ Java APIs meet all four conditions, and hence are required for optimized WIP.

This property is valid only if the value of `DeliveryTransport` is `JMS` and the value of `BrokerType` is `ICS`.

The default value is `false`.

jms.UserName

The `jms.UserName` property specifies the user name for the JMS provider. A value for this property is optional.

There is no default value.

JvmMaxHeapSize

The `JvmMaxHeapSize` property specifies the maximum heap size for the agent (in megabytes).

This property is valid only if the value for the `RepositoryDirectory` property is set to `<REMOTE>`.

The default value is `128m`.

JvmMaxNativeStackSize

The `JvmMaxNativeStackSize` property specifies the maximum native stack size for the agent (in kilobytes).

This property is valid only if the value for the `RepositoryDirectory` property is set to `<REMOTE>`.

The default value is `128k`.

JvmMinHeapSize

The `JvmMinHeapSize` property specifies the minimum heap size for the agent (in megabytes).

This property is valid only if the value for the RepositoryDirectory property is set to <REMOTE>.

The default value is 1m.

ListenerConcurrency

The ListenerConcurrency property supports multithreading in WebSphere MQ Listener when WebSphere InterChange Server is the integration broker. It enables batch writing of multiple events to the database, thereby improving system performance.

This property is valid only with connectors that use MQ transport. The value of the DeliveryTransport property must be MQ.

The default value is 1.

Locale

The Locale property specifies the language code, country or territory, and, optionally, the associated character code set. The value of this property determines cultural conventions such 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 is a two-character language code (in lowercase letters)

TT is a two-letter country or territory code (in uppercase letters)

codeset is the name of the associated character code set (may be optional).

By default, only a subset of supported locales are listed. To add other supported values to the list, you modify the \Data\Std\stdConnProps.xml file in the <ProductDir>\bin directory. For more information, refer to the Connector Configurator appendix in this guide.

If the connector has not been internationalized, the only valid value for this property is en_US. To determine whether a specific connector has been globalized, refer to the user guide for that adapter.

The default value is en_US.

LogAtInterchangeEnd

The LogAtInterchangeEnd property specifies whether to log errors to the log destination of the integration broker.

Logging to the log destination also turns on e-mail notification, which generates e-mail messages for the recipient specified as the value of MESSAGE_RECIPIENT in the InterchangeSystem.cfg file when errors or fatal errors occur. For example, when a connector loses its connection to the application, if the value of LogAtInterChangeEnd is true, an e-mail message is sent to the specified message recipient.

This property is valid only if the value of the RepositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

The default value is false.

MaxEventCapacity

The MaxEventCapacity property specifies maximum number of events in the controller buffer. This property is used by the flow control feature.

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

The value can be a positive integer between 1 and 2147483647.

The default value is 2147483647.

MessageFileName

The MessageFileName property specifies the name of the connector message file. The standard location for the message file is \connectors\messages in the product directory. Specify the message file name 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 connector has its own message file, see the individual adapter user guide.

The default value is InterchangeSystem.txt.

MonitorQueue

The MonitorQueue property specifies the logical queue that the connector uses to monitor duplicate events.

It is valid only if the value of the DeliveryTransport property is JMS and the value of the DuplicateEventElimination is true.

The default value is <CONNECTORNAME>/MONITORQUEUE

OADAutoRestartAgent

the OADAutoRestartAgent property specifies whether the connector uses the automatic and remote restart feature. This feature uses the WebSphere 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 WebSphere MQ-triggered OAD feature, see the *Installation Guide for Windows* or *for UNIX*.

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

The default value is false.

OADMaxNumRetry

The OADMaxNumRetry property specifies the maximum number of times that the WebSphere MQ-triggered Object Activation Daemon (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.

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

The default value is 1000.

OADRetryTimeInterval

The OADRetryTimeInterval property specifies the number of minutes in the retry-time interval for the WebSphere MQ-triggered Object Activation Daemon (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.

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

The default value is 10.

PollEndTime

The PollEndTime property specifies the time to stop polling the event queue. The format is *HH:MM*, where *HH* is 0 through 23 hours, and *MM* represents 0 through 59 minutes.

You must provide a valid value for this property. The default value is HH:MM without a value, and it must be changed.

If the adapter runtime detects:

- PollStartTime set and PollEndTime not set, or
- PollEndTime set and PollStartTime not set

it will poll using the value configured for the PollFrequency property.

PollFrequency

The PollFrequency property 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 the PollQuantity property.
- Process these objects. For some connectors, this may be partly done on separate threads, which execute asynchronously to the next polling action.
- Delay for the interval specified by the PollFrequency property.
- Repeat the cycle.

The following values are valid for this property:

- The number of milliseconds between polling actions (a positive integer).

- The word `no`, which causes the connector not to poll. Enter the word in lowercase.
- The word `key`, which causes the connector to poll only when you type the letter `p` in the connector Command Prompt window. 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

The `PollQuantity` property designates the number of items from the application that the connector polls for. If the adapter has a connector-specific property for setting the poll quantity, the value set in the connector-specific property overrides the standard property value.

This property is valid only if the value of the `DeliveryTransport` property is `JMS`, and the `ContainerManagedEvents` property has a value.

An e-mail message is also considered an event. The connector actions are as follows when it is polled for e-mail.

- When it is polled once, the connector detects the body of the message, which it reads as an attachment. Since no data handler was specified for this mime type, it will then ignore the message.
- The connector processes the first BO attachment. The data handler is available for this MIME type, so it sends the business object to Visual Test Connector.
- When it is polled for the second time, the connector processes the second BO attachment. The data handler is available for this MIME type, so it sends the business object to Visual Test Connector.
- Once it is accepted, the third BO attachment should be transmitted.

PollStartTime

The `PollStartTime` property specifies the time to start polling the event queue. The format is `HH:MM`, where `HH` is 0 through 23 hours, and `MM` represents 0 through 59 minutes.

You must provide a valid value for this property. The default value is `HH:MM` without a value, and it must be changed.

If the adapter runtime detects:

- `PollStartTime` set and `PollEndTime` not set, or
- `PollEndTime` set and `PollStartTime` not set

it will poll using the value configured for the `PollFrequency` property.

RepositoryDirectory

The `RepositoryDirectory` property is the location of the repository from which the connector reads the XML schema documents that store the metadata for business object definitions.

If the integration broker is WebSphere InterChange Server, this value must be set to 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 is set to <ProductDir>\repository by default. However, it may be set to any valid directory name.

RequestQueue

The RequestQueue property specifies the queue that is used by the integration broker to send business objects to the connector.

This property is valid only if the value of the DeliveryTransport property is JMS.

The default value is <CONNECTORNAME>/REQUESTQUEUE.

ResponseQueue

The ResponseQueue property specifies the JMS response queue, which delivers a response message from the connector framework to the integration broker. When the integration broker is WebSphere InterChange Server, the server sends the request and waits for a response message in the JMS response queue.

This property is valid only if the value of the DeliveryTransport property is JMS.

The default value is <CONNECTORNAME>/RESPONSEQUEUE.

RestartRetryCount

The RestartRetryCount property specifies the number of times the connector attempts to restart itself. When this property is used for a connector that is connected in parallel, it specifies the number of times the master connector application-specific component attempts to restart the client connector application-specific component.

The default value is 7.

RestartRetryInterval

The RestartRetryInterval property specifies the interval in minutes at which the connector attempts to restart itself. When this property is used for a connector that is linked in parallel, it specifies the interval at which the master connector application-specific component attempts to restart the client connector application-specific component.

Possible values for the property range from 1 through 2147483647.

The default value is 1.

ResultsSetEnabled

The ResultsSetEnabled property enables or disables results set support when Information Integrator is active. This property can be used only if the adapter supports DB2 Information Integrator.

This property is valid only if the value of the DeliveryTransport property is JMS, and the value of BrokerType is WMQI.

The default value is false.

ResultSetSize

The `ResultSetSize` property defines the maximum number of business objects that can be returned to Information Integrator. This property can be used only if the adapter supports DB2 Information Integrator.

This property is valid only if the value of the `ResultSetEnabled` property is true.

The default value is 0. This means that the size of the results set is unlimited.

RHF2MessageDomain

The `RHF2MessageDomain` property allows you to configure the value of the field domain name in the JMS header. When data is sent to a WebSphere message broker over JMS transport, the adapter framework writes JMS header information, with a domain name and a fixed value of `mrm`. A configurable domain name lets you track how the WebSphere message broker processes the message data.

This is an example header:

```
<mcd><Msd>mrm</Msd><Set>3</Set><Type>  
Retek_POPhyDesc</Type><Fmt>CwXML</Fmt></mcd>
```

This property is valid only if the value of `BrokerType` is `WMQI` or `WAS`. Also, it is valid only if the value of the `DeliveryTransport` property is `JMS`, and the value of the `WireFormat` property is `CwXML`.

Possible values are `mrm` and `xml`. The default value is `mrm`.

SourceQueue

The `SourceQueue` property 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 75.

This property is valid only if the value of `DeliveryTransport` is `JMS`, and a value for `ContainerManagedEvents` is specified.

The default value is `<CONNECTORNAME>/SOURCEQUEUE`.

SynchronousRequestQueue

The `SynchronousRequestQueue` property 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 synchronous request queue and waits for a response from the broker on the synchronous response queue. The response message sent to the connector has a correlation ID that matches the ID of the original message.

This property is valid only if the value of `DeliveryTransport` is `JMS`.

The default value is `<CONNECTORNAME>/SYNCHRONOUSREQUESTQUEUE`

SynchronousRequestTimeout

The SynchronousRequestTimeout property specifies the time in milliseconds that the connector waits for a response to a synchronous request. If the response is not received within the specified time, the connector moves the original synchronous request message (and error message) to the fault queue.

This property is valid only if the value of DeliveryTransport is JMS.

The default value is 0.

SynchronousResponseQueue

The SynchronousResponseQueue property delivers response messages in reply to a synchronous request from the broker to the connector framework. This queue is necessary only if the connector uses synchronous execution.

This property is valid only if the value of DeliveryTransport is JMS.

The default is <CONNECTORNAME>/SYNCHRONOUSRESPONSEQUEUE

TivoliMonitorTransactionPerformance

The TivoliMonitorTransactionPerformance property specifies whether IBM Tivoli Monitoring for Transaction Performance (ITMTP) is invoked at run time.

The default value is false.

WireFormat

The WireFormat property specifies the message format on the transport:

- If the value of the RepositoryDirectory property is a local directory, the value is CwXML.
- If the value of the RepositoryDirectory property is a remote directory, the value is CwB0.

WsifSynchronousRequestTimeout

The WsifSynchronousRequestTimeout property specifies the time in milliseconds that the connector waits for a response to a synchronous request. If the response is not received within the specified time, the connector moves the original synchronous request message (and an error message) to the fault queue.

This property is valid only if the value of BrokerType is WAS.

The default value is 0.

XMLNamespaceFormat

The XMLNamespaceFormat property specifies short or long namespaces in the XML format of business object definitions.

This property is valid only if the value of BrokerType is set to WMQI or WAS.

The default value is short.

Appendix B. Connector-specific properties and required business-object properties

This appendix lists the properties specific to the Adapter for TCP/IP and the required business-object properties for the adapter.

Connector-specific properties

Table 8 provides a summary of the properties and values that are specific to the Adapter for TCP/IP. Plus signs (+) indicate child properties. Definitions and discussion of the properties follow the table. The Connector Configurator is used to set these values.

Table 8. Connector Specific Properties

Properties	Possible values	Default values	Update Method	Required?
ServerConfiguration			agent restart	No, but if this set is not populated, event processing is not enabled. A warning message will be logged.
+Port	Integer greater than 0		agent restart	Yes
+TransportProtocol	For this release, only tcp supported. In the future, others, including secure tcp/ip, possible	tcp	agent restart	Yes
+MaxRequestProcessors	Integer greater than 0	2	agent restart	No
+MaxRequestPoolSize	Integer greater than 0	3	agent restart	No
+ServerQueueLength	Integer greater than 0		agent restart	No
+ReceiveBufferSize	Integer greater than 0		agent restart	No
+SendBufferSize	Integer greater than 0		agent restart	No
+KeepAlive	true or false	false	agent restart	No
+SocketTimeout	Integer greater than 0	10000	agent restart	No
+RetryInterval	Integer 0 or greater	0	agent restart	No
+NumberofRetries	Integer 0 or greater	0	agent restart	No

Table 8. Connector Specific Properties (continued)

Properties	Possible values	Default values	Update Method	Required?
+EnableBase64Encoding	true or false	false	agent restart	No
ClientConfiguration			agent restart	No, but if this set is not populated, service call request processing is not enabled. A warning message will be logged.
+Clients			agent restart	Yes. This is a hierarchical property that holds specific child client data. If it is not populated, a warning will be logged.
++Client1	Client name		agent restart	No
+++Host	Address of remote host		agent restart	Yes
+++Port	Port number of remote host		agent restart	Yes
+++TransportProtocol	Transport protocol this handler implements	tcp	agent restart	Yes
+++ReceiveBufferSize	Integer greater than 0	Determined by operating system and JVM	agent restart	No
+++SendBufferSize	Integer greater than 0	Determined by operating system and JVM	agent restart	No
+++KeepAlive	true or false	false	agent restart	No
+++SocketTimeout	Integer greater than 0		agent restart	No
+++MaxAttemptsToRead	Integer greater than 0	1	agent restart	No
+++NumberOfRetries	Integer 0 or greater	0	agent restart	No
+++RetryInterval	Integer 0 or greater	0	agent restart	No

Table 8. Connector Specific Properties (continued)

Properties	Possible values	Default values	Update Method	Required?
+++EnableBase64Decoding	true or false	false	agent restart	No
+++ConnectionPoolSize	true or false	20	agent restart	No
+++ConnectionPoolingOn	Integer 0 or greater	false	agent restart	No
++Client2	As above		agent restart	No
DataHandlerMimeType	Mime types	text/ name value	agent restart	Yes
ConfigurationMetaObject	Static Meta Object	BIA _Static _MO	agent restart	Yes
ServiceRegistrationMO	Service Meta Object	BIA _MO _Service	agent restart	Yes
DataHandlerMetaObjectName	DataHandler Configuration Meta Object	BIA _MO _Data Handler _Default	agent restart	Yes
ErrorClassName			agent restart	Yes

Connector-specific configuration property descriptions

The following is a list of definitions of the above properties.

ServerConfiguration

The set of properties used by the TCP/IP connector for event or inbound processing. In this case, the connector functions as a TCP server and listens for requests on the defined port. Only one server can be defined per connector.

Port

The local port on which the connector listens.

TransportProtocol

The transport protocol this listener implements. For this release, the only available value is "tcp". More values, such as secure TCP/IP, may be added in future releases.

MaxRequestProcessors

Sets the maximum number of threads to run concurrently for handling incoming requests on the defined port.

MaxRequestPoolSize

Sets the maximum number of incoming requests that are cached to be processed simultaneously. At any given moment, the connector can process at most (MaxRequestProcessors + MaxRequestPoolSize) requests.

ServerQueueLength

Sets the length of the server socket queue for incoming connection requests. This value specifies how many incoming requests can be stored at one time before the host starts refusing connections.

Note: The maximum queue length is operating-system-dependent.

ReceiveBufferSize

Sets the suggested network I/O buffer size. This value serves as a hint to the underlying platform's networking code. Increasing buffer size can increase the performance of network I/O for high-volume connections, while decreasing it can help reduce the backlog of incoming data.

SendBufferSize

Sets the suggested network I/O buffer size. This value serves as a hint to the underlying platform's networking code. Increasing buffer size can increase the performance of network I/O for high-volume connections, while decreasing it can help reduce the backlog of incoming data.

KeepAlive

Heartbeat probe. Periodically sends an empty data packet with its current sequence, acknowledgment, and window numbers.

SocketTimeout

Sets base timeout blocking in milliseconds for the socket. With this option set to a non-zero timeout, a call to `read()` for this socket will block for only this amount of time. If the timeout expires, a `java.io.InterruptedIOException` is raised, though the socket is still valid. The option must be enabled prior to entering the blocking operation to have effect. A timeout of zero is interpreted as an infinite timeout.

RetryInterval

Sets the suggested interval the connector in TCP server mode will wait before retrying an operation that has failed. Such situations may include errors that take place while accepting the connection, opening streams for read/write, reading or writing to these streams, and so on.

NumberOfRetries

Sets the suggested number of retries the server will make in the error conditions described in "RetryInterval."

EnableBase64Encoding

Specifies whether Base64 encoding is enabled or disabled. If the event data that comes into the adapter needs to be Base64-encoded, set this property to true. The data could be Base-64 encoded to preserve the byte values of incoming event data.

ClientConfiguration

The set of properties used by the TCP/IP connector for Service Call Request or outbound processing. In this case, the connector functions as a TCP client and initiates connections with remote hosts defined in the configuration. Multiple clients can be defined per connector.

Clients

This is a hierarchical property that functions only to hold children that define client configurations.

Client1

Specifies the name of the client. Correlates with the ASI specified in the Configuration Meta Object.

Host

Sets the address of the remote host.

Port

Sets the port number of the remote host to which the client needs to connect.

TransportProtocol

Sets the supported transport protocol. For this release, "tcp" is the only available value.

ReceiveBufferSize

Sets the suggested network I/O buffer size. This value serves as a hint to the underlying platform's networking code. Increasing buffer size can increase the performance of network I/O for high-volume connections, while decreasing it can help reduce the backlog of incoming data.

SendBufferSize

Sets the suggested network I/O buffer size. This value serves as a hint to the underlying platform's networking code. Increasing buffer size can increase the performance of network I/O for high-volume connections, while decreasing it can help reduce the backlog of incoming data.

KeepAlive

Heartbeat probe. Periodically sends an empty data packet with its current sequence, acknowledgment, and window numbers.

SocketTimeout

Sets timeout blocking in milliseconds for this socket. When this is set to a non-zero value, a read() call on the InputStream associated with this socket will block for only this amount of time. If the timeout expires, a `java.io.InterruptedIOException` is raised, though the socket is still valid. The option must be enabled prior to entering the blocking operation to have effect. A timeout of zero is interpreted as an infinite timeout.

MaxAttemptsToRead

Sets the maximum number of times the connector will read from the socket once it starts receiving data. This property allows for the reception of acknowledgment data that is sent in more than one stream. This approach differs from event processing because it is assumed that the amount of data received in an acknowledgment will be small.

RetryInterval

Sets the suggested interval the connector in TCP client mode will wait before retrying an operation that has failed. Such situations may include errors that take place while opening streams for read/write, reading or writing to these streams, and so on.

NumberOfRetries

Sets the suggested number of retries the connector in TCP client mode will make in the error conditions described in "RetryInterval."

EnableBase64Decoding

Specifies whether Base64 decoding is enabled or disabled. If the request data needs to be Base64-decoded before it is sent to the remote TCP/IP server, set this property to true.

ConnectionPoolingOn

Indicates whether connection pooling is switched on when the adapter connects to the remote TCP/IP server during request processing. If this property is set to false, a new connection is opened for each request.

ConnectionPoolSize

Indicates the number of connections to be created in the connection pool. ConnectionPoolingOn must be set to true for this property to have any effect.

Client2

The name of the next client configuration.

ConfigurationMetaObject

The meta-object that holds static configuration information. See the description of BIA_Static_MO in “General meta-objects” on page 29.

ServiceRegistrationMO

The top-level meta-object that holds service information. See the description of BIA_MO_Service in “General meta-objects” on page 29.

DataHandlerMimeType

Sets the expected MIME type of the incoming data. Used to specify the appropriate data handler.

ErrorClassName

The error class that should be invoked by the adapter when an error occurs during event or request processing. The value of ErrorClassName is the fully qualified name of the class (for example, com.ibm.adapters.protocol.util.TestErrorClass). See Appendix C, “Error handling,” on page 97 for detailed information.

DataHandlerMetaObjectName

The top-level meta-object that holds data-handler configuration information other than that contained in the service object. See the description of BIA_MO_DataHandler_Default in “General meta-objects” on page 29.

Supported business objects

The Connector Configurator should also be used to set up the Supported Business Objects tab. Table 9 shows an example of values that can be entered.

Table 9. Supported Business Objects Properties

Business Object Name	Message Set ID
BIA_MO_DataHandler_Default	1
BIA_MO_Service	2
BIA_MO_Tcpip_MapSubscriptions	3
BIA_STATIC_MO	4

The following configuration properties are associated with the business objects:

Table 10. Configuration properties and associated business objects

Configuration property	Business object
DataHandlerMetaObjectName	BIA_MO_DataHandler_Default
ServiceRegistrationMO	BIA_MO_Service
ConfigurationMetaObject	BIA_STATIC_MO

For more information on these objects, see “General meta-objects” on page 29.

Performance tuning

This section describes how to set properties to tune the performance of the adapter for TCP/IP.

Event processing

Set `MaxRequestProcessors` to the number of events that are expected to be simultaneously processed by the adapter. For example, set the value to 20 when 20 events are expected to be processed simultaneously.

The value of `MaxRequestPoolSize` is ideally 1.5 to 2 times the value of the `MaxRequestProcessors` property. For example, if the value of `MaxRequestProcessors` is 20, set the value of `MaxRequestPoolSize` to 30.

Request processing

You can set connection pooling on for better performance during request processing, because connections to the remote TCP/IP server are pooled and reused. Set `ConnectionPoolSize` to the number of requests that need to be processed in parallel. For example, set a value of 20 to process 20 requests in parallel.

`RetryInterval` and `MaxAttemptsToRead` also affect performance. When you enter values for these properties, the adapter will continue to try to read from the remote TCP/IP server, at intervals specified by `RetryInterval`, up to the value specified by `MaxAttemptsToRead`. This is to collate all possible responses from the remote TCP/IP Server (network delays might cause data to come in multiple reads). After the adapter receives the initial response, it will retry for `MaxAttemptsToRead` times, sleeping every `RetryInterval`, to see if any more data has come from the TCP/IP server, because the initial response might be incomplete. The adapter then returns the data received in this interval as a response to the request. At the same time, one connection is freed from the connection pool. Otherwise, the adapter will keep waiting for a response until `SocketTimeout`.

Set these values appropriately to receive a response from the TCP/IP server consistently. Do not keep a high value for `MaxAttemptsToRead`. Also, do not keep a very high value for `Socket Timeout`, because it might tie up connections from the connection pool.

Appendix C. Error handling

The `ErrorClassName` property in the connector configuration file is required to indicate to the adapter which error handler class to invoke. The value of `ErrorClassName` is the fully qualified name of the class (for example, `com.ibm.adapters.protocol.util.TestErrorClass`).

Use the information in this appendix to develop the error handler. To implement its interfaces, put it in the class path of the adapter (in the start script).

Error handler for event processing

If an error occurs (for example, event data is corrupted or the broker is unavailable) that causes event delivery to fail, the error must be communicated to the external application. The external application can then decide whether to correct the error (for example, by correcting the data and resending it or by attempting broker re-start).

An interface error handler is provided with the adapter. It has methods for handling errors in event as well as request processing. Use the information in this appendix to implement the methods.

These methods collect the error-related information from the adapter runtime (for example, whether the error occurred during “event” or “response to event”, which component caused the error (data handler, message handler, or broker), what was the exact exception thrown, what was the originating event, and so on).

This feature is currently used in at message handler, data handler, and broker error checkpoints. The method invoked during inbound flow/event processing is `handleEventError`.

Checkpoint examples

The following is a list of checkpoints for the HL7 sample that is provided with the adapter. They translate to equivalent checkpoints in custom implementations.

- The incoming event is supposed to be HL7 LLP. The message handler detects that it does not have appropriate demarcation characters, such as SOB and EOB.
- The incoming event violates data-handler parsing rules (for example, it is not a valid HL7 message).
- The event data is valid and can be transformed into a business object, but the broker becomes unavailable before the data can be delivered to the broker.
- In the case of synchronous event processing, the response business object cannot be converted into a stream or string because of errors in the business-object data.
- In the case of synchronous event processing, the response business object data, after being processed by the data handler, cannot be processed by the message handler.

Interface details

The following section provides information you can use to implement the interface. Note that `handleEventError` is the method of `Error Handler` that is invoked in event processing.

This section also lists some predefined values provided for the input parameters. The predefined values are underlined.

```
package com.ibm.adapters.protocol.util;
import com.crossworlds.cwconnectorapi.CWConnectorBusObj;

public interface ErrorHandler {

    //output string will be written to the client socket's output stream
    // the content of this string is the responsibility of error handler
    //implementer
    public String handleEventError(
        byte[] event, //data over input stream of the socket
        byte[] errorData, //portion of input bytes which caused error,
        as the input data can contain multiple messages (see SupportMultipleMessages
        in Service M0), out of which only some portion can be invalid. This parameter
        tells which exact portion contains invalid data.

        String errorLocation, //MessageHandler/DataHandler/Broker

        Object exceptionObj, //actual exception
        String flow //event/responseToEvent );

    public String handleRequestError(
        CWConnectorBusObj currBusObj, //BO from broker
        String errorLocation, //MessageHandler/DataHandler/Broker
        Object exceptionObj, //actual exception
        String flow //request/responseToRequest
    );
}
```

The return type of the method `handleEventError` should be the string that is written back to the calling TCP/IP client.

Error handler for request processing

If an error occurs during the outbound (request) flow, the adapter invokes the Error Handler. The Error Handler performs actions, such as archiving the incoming business object, and then the adapter returns a -1 failure status to the broker.

An interface error handler is provided with the adapter. It has methods for handling errors in event as well as request processing. Use the information in this section to implement the methods.

These methods collect the error-related information from the adapter runtime (for example, whether the error occurred during “event” or “response to event”, which component caused the error (data handler, message handler, or broker), what was the exact exception thrown, what was the originating event, and so on).

This feature is currently used at message handler, data handler and broker error checkpoints. `ErrorLocation` is a variable passed by the adapter to the method invoked during Request processing (`handleRequestError`).

Checkpoint examples

The following are examples of checkpoints in request processing.

- The incoming request cannot be converted to application data because of a data handler exception. The value of `errorLocation` passed to the error handler would be `DataHandler`.

- The message handlers cannot handle the application data returned from the data handler. The value of `errorLocation` passed to the error handler would be `MessageHandler`.
- The application data cannot be sent to the remote TCP/IP server (for example, the server may be down). The value of `errorLocation` passed to the error handler would be `SendRequestToServer`.
- The response from the TCP/IP server is invalid. The value of `errorLocation` passed to the error handler would be `ProcessResponseFromServer`.
- Unexpected exceptions occur during request processing. The value of `errorLocation` passed to the error handler would be `InDoVerbFor`.

Interface details

The following section provides information you can use to implement the interface. Note that `handleRequestError` is the method of Error Handler invoked in request processing.

This section also lists some predefined values provided for the input parameters. The predefined values are underlined.

```
package com.ibm.adapters.protocol.util;
import com.crossworlds.cwconnectorapi.CWConnectorBusObj;

public interface ErrorHandler {

    public String handleEventError(
        byte[] event, //data over input stream of the socket
        byte[] errorData, //portion of input bytes which caused error,
        as the input data can contain multiple messages (see SupportMultipleMessages
        in Service MO), out of which only some portion can be invalid. This parameter
        tells which exact portion contain invalid data.

        String errorLocation, //MessageHandler/DataHandler/Broker
        Object exceptionObj, //actual exception
        String flow //event/responseToEvent );

    public String handleRequestError(
        CWConnectorBusObj currBusObj, //BO from broker
        String errorLocation,
        //DataHandler/MessageHandler/SendRequestToServer/ProcessResponseFromServer/
        InDoVerbFor
        Object exceptionObj, //actual exception
        String flow //request/responseToRequest );
}
```

The return type of the method `handleEventError` should be the string that has to be written back to the calling TCP/IP client.

Guidelines for the message handler and data handler

To invoke the error handler from the message handler (in the case of an error in the format of the message or other map-related exception condition), throw an exception with an appropriate error message, or return the string "FAILURE" from the map method.

To invoke the error handler from the data handler (when parsing a message), throw an exception with an appropriate error message.

Appendix D. Common Event Infrastructure

WebSphere Business Integration Server Foundation includes the Common Event Infrastructure Server Application, which is required for Common Event Infrastructure to operate. The WebSphere Application Server Foundation can be installed on any system (it does not have to be the same machine on which the adapter is installed.)

The WebSphere Application Server Application Client includes the libraries required for interaction between the adapter and the Common Event Infrastructure Server Application. You must install WebSphere Application Server Application Client on the same system on which you install the adapter. The adapter connects to the WebSphere Application Server (within the WebSphere Business Integration Server Foundation) by means of a configurable URL.

Common Event Infrastructure support is available using any integration broker supported with this release.

Required software

In addition to the software prerequisites required for the adapter, you must have the following installed for Common Event Infrastructure to operate:

- WebSphere Business Integration Server Foundation 5.1.1
- WebSphere Application Server Application Client 5.0.2, 5.1, or 5.1.1.
(WebSphere Application Server Application Client 5.1.1 is provided with WebSphere Business Integration Server Foundation 5.1.1.)

Note: Common Event Infrastructure is not supported on any HP-UX or Linux platform.

Enabling Common Event Infrastructure

Common Event Infrastructure functionality is enabled with the standard properties `CommonEventInfrastructure` and `CommonEventInfrastructureContextURL`, configured with Connector Configurator. By default, Common Event Infrastructure is not enabled. The `CommonEventInfrastructureContextURL` property enables you to configure the URL of the Common Event Infrastructure server. (Refer to Appendix A, “Standard configuration properties,” on page 65 for more information.)

Obtaining Common Event Infrastructure adapter events

If Common Event Infrastructure is enabled, the adapter generates Common Event Infrastructure events that map to the following adapter events:

- Starting the adapter
- Stopping the adapter
- An application response to a timeout from the adapter agent
- Any `doVerbFor` call issued from the adapter agent
- A `gotAppEvent` call from the adapter agent

For another application (the “consumer application”) to receive the Common Event Infrastructure events generated by the adapter, the application must use the Common Event Infrastructure event catalog to determine the definitions of appropriate events and their properties. The events must be defined in the event catalog for the consumer application to be able to consume the sending application’s events.

“Common Event Infrastructure event catalog definitions” contains XML format metadata showing, for WebSphere Business Information adapters, the event descriptors and properties the consumer application should search for.

For more information

For more information about Common Event Infrastructure, refer to the Common Event Infrastructure information in the WebSphere Business Integration Server Foundation documentation, available at the following URL:

<http://publib.boulder.ibm.com/infocenter/ws51help>

For sample XML metadata showing the adapter-generated event descriptors and properties a consumer application should search for, refer to “Common Event Infrastructure event catalog definitions.”

Common Event Infrastructure event catalog definitions

The Common Event Infrastructure event catalog contains event definitions that can be queried by other applications. The following are event definition samples, using XML metadata, for typical adapter events. If you are writing another application, your application can use event catalog interfaces to query against the event definition. For more information about event definitions and how to query them, refer to the Common Event Infrastructure documentation that is available from the online IBM WebSphere Server Foundation Information Center.

For WebSphere Business Integration adapters, the extended data elements that need to be defined in the event catalog are the keys of the business object. Each business object key requires an event definition. So for any given adapter, various events such as start adapter, stop adapter, timeout adapter, and any doVerbFor event (create, update, or delete, for example) must have a corresponding event definition in the event catalog.

The following sections contain examples of the XML metadata for start adapter, stop adapter, and event request or delivery.

XML format for “start adapter” metadata

```
<eventDefinition name="startADAPTER"
  parent="event">
  <property name="creationTime" //Comment: example value would be
    "2004-05-13T17:00:16.319Z"
    required="true" />
  <property name="globalInstanceId" //Comment: Automatically generated
    by Common Event Infrastructure
    required="true"/>
  <property name="sequenceNumber" //Comment: Source defined number
    for messages to be sent/sorted logically
    required="false"/>
  <property name="version" //Comment: Version of the event
    required="false"
```

```

        defaultValue="1.0.1"/>
    <property name="sourceComponentId"
        path="sourceComponentId"
        required="true"/>
    <property name="application" //Comment: The name#version of the
source application generating the event. Example is "SampleConnector#3.0.0"
        path="sourceComponentId/application"        required="false"/>
    <property name="component" //Comment: This will be the name#version
of the source component.
        path="sourceComponentId/component"
        required="true"
        defaultValue="ConnectorFrameWorkVersion#4.2.2"/>
    <property name="componentIdType" //Comment: specifies the format
and meaning of the component
        path="sourceComponentId/componentIdType"
        required="true"
        defaultValue="Application"/>
    <property name="executionEnvironment"
//Comment: Identifies the environment the application is running
in...example is "Windows 2000#5.0"
        path="sourceComponentId/executionEnvironment"
        required="false" />
    <property name="location" //Comment: The value of this is the
server name...example is "WQMI"
        path="sourceComponentId/location"
        required="true"/>
    <property name="locationType" //Comment specifies the format and
meaning of the location
        path="sourceComponentId/locationType"
        required="true"
        defaultValue="Hostname"/>
    <property name="subComponent" //Comment:further distinction
of the logical component
        path="sourceComponentId/subComponent"
        required="true"
        defaultValue="AppSide_Connector.AgentBusinessObjectManager"/>
    <property name="componentType" //Comment: well-defined name
used to characterize all instances of this component
        path="sourceComponentId/componentType"
        required="true"
        defaultValue="ADAPTER"/>
    <property name="situation" //Comment: Defines the type of
situation that caused the event to be reported
        path="situation"
        required="true"/>
    <property name="categoryName=" //Comment: Specifies the type
of situation for the event
        path="situation/categoryName"
        required="true"
        defaultValue="StartSituation"/>
    <property name="situationType" //Comment: Specifies the type
of situation and disposition of the event
        path="situation/situationType"
        required="true"
    <property name="reasoningScope" //Comment: Specifies the scope
of the impact of the event
        path="situation/situationType/reasoningScope"
        required="true"
        permittedValue="INTERNAL"
        permittedValue="EXTERNAL"/>
    <property name="successDisposition" //Comment: Specifies the
success of event
        path="situation/situationType/successDisposition"
        required="true"
        permittedValue="SUCCESSFUL"
        permittedValue="UNSUCCESSFUL" />
    <property name="situationQualifier" //Comment: Specifies the

```

```

situation qualifiers for this event
  path="situation/situationType/situationQualifier"
  required="true"
  permittedValue="START_INITIATED"
  permittedValue="RESTART_INITIATED"
  permittedValue="START_COMPLETED" />
</eventDefinition>

```

XML format for "stop adapter" metadata

The metadata for "stop adapter" is the same as that for "start adapter" with the following exceptions:

- The default value for the categoryName property is StopSituation:

```

<property name="categoryName="
  //Comment: Specifies the type
  of situation for the event
    path="situation/categoryName"
    required="true"
    defaultValue="StopSituation"/>

```

- The permitted values for the situationQualifier property differ and are as follows for "stop adapter":

```

<property name="situationQualifier"
  //Comment: Specifies the situation qualifiers for this event
    path="situation/situationType/situationQualifier"
    required="true"
    permittedValue="STOP_INITIATED"
    permittedValue="ABORT_INITIATED"
    permittedValue="PAUSE_INITIATED"
    permittedValue="STOP_COMPLETED"
  />

```

XML format for "timeout adapter" metadata

The metadata for "timeout adapter" is the same as that for "start adapter" and "stop adapter" with the following exceptions:

- The default value for the categoryName property is ConnectSituation:

```

<property name="categoryName="
  //Comment: Specifies the type
  of situation for the event
    path="situation/categoryName"
    required="true"
    defaultValue="ConnectSituation"/>

```

- The permitted values for the situationQualifier property differ and are as follows for "timeout adapter":

```

<property name="situationQualifier" //Comment: Specifies
  the situation qualifiers for this event
    path="situation/situationType/situationQualifier"
    required="true"
    permittedValue="IN_USE"
    permittedValue="FREED"
    permittedValue="CLOSED"
    permittedValue="AVAILABLE"
  />

```

XML format for "request" or "delivery" metadata

At the end of this XML format are the extended data elements. The extended data elements for adapter request and delivery events represent data from the business object being processed. This data includes the name of the business object, the key (foreign or local) for the business object, and business objects that are children of parent business objects. The children business objects are then broken down into the same data as the parent (name, key, and any children business objects). This data is represented in an extended data element of the event definition. This data will change depending on which business object, which keys, and which child business objects are being processed. The extended data in this event definition is just an example and represents a business object named Employee with a key EmployeeId and a child business object EmployeeAddress with a key EmployeeId. This pattern could continue for as much data as exists for the particular business object.

```
<eventDefinition name="createEmployee" //Comment: This
extension name is always the business object verb followed by the business
object name
  parent="event">
  <property name="creationTime" //Comment: example value would be
"2004-05-13T17:00:16.319Z"
  required="true" />
  <property name="globalInstanceId" //Comment: Automatically generated
by Common Event Infrastructure
  required="true"/>
  <property name="localInstanceId" //Comment: Value is business
object verb+business object name+#+app name+ business object identifier
  required="false"/>
  <property name="sequenceNumber" //Comment: Source defined number
for messages to be sent/sorted logically
  required="false"/>
  <property name="version" //Comment: Version of the event...value is
set to 1.0.1
  required="false"
  defaultValue="1.0.1"/>
  <property name="sourceComponentId"
  path="sourceComponentId"
  required="true"/>
  <property name="application" //Comment: The name#version of the
source application generating the event...example is
"SampleConnector#3.0.0"
  path="sourceComponentId/application"
  required="false"/>
  <property name="component" //Comment: This will be the name#version
of the source component.
  path="sourceComponentId/component"
  required="true"
  defaultValue="ConnectorFrameWorkVersion#4.2.2"/>
  <property name="componentIdType" //Comment: specifies the format
and meaning of the component
  path="sourceComponentId/componentIdType"
  required="true"
  defaultValue="Application"/>
  <property name="executionEnvironment" //Comment: Identifies the
environment#version the app is running in...example is "Windows 2000#5.0"
  path="sourceComponentId/executionEnvironment"
  required="false" />
  <property name="instanceId" //Comment: Value is business object
verb+business object name+#+app name+ business object identifier
  path="sourceComponentId/instanceId"
  required="false"
  <property name="location" //Comment: The value of this is the
server name...example is "WQMI"
  path="sourceComponentId/location"
```

```

        required="true"/>
        <property name="locationType" //Comment specifies the format and
meaning of the location
        path="sourceComponentId/locationType"
        required="true"
        defaultValue="Hostname"/>
        <property name="subComponent" //Comment: further distinction of the
logical component-in this case the value is the name of the business
object
        path="sourceComponentId/subComponent"
        required="true"/>
        <property name="componentType" //Comment: well-defined name used
to characterize all instances of this component
        path="sourceComponentId/componentType"
        required="true"
        defaultValue="ADAPTER"/>
        <property name="situation" //Comment: Defines the type of
situation that caused the event to be reported
        path="situation"
        required="true"/>
        <property name="categoryName" //Comment: Specifies the type
of situation for the event
        path="situation/categoryName"
        required="true"
        permittedValue="CreateSituation"
        permittedValue="DestroySituation"
        permittedValue="OtherSituation" />
        <property name="situationType" //Comment: Specifies the type
of situation and disposition of the event
        path="situation/situationType"
        required="true"
        <property name="reasoningScope" //Comment: Specifies the scope
of the impact of the event
        path="situation/situationType/reasoningScope"
        required="true"
        permittedValue="INTERNAL"
        permittedValue="EXTERNAL"/>
        <property name="successDisposition" //Comment: Specifies the
success of event
        path="situation/situationType/successDisposition"
        required="true"
        permittedValue="SUCCESSFUL"
        permittedValue="UNSUCCESSFUL" />
        <extendedDataElements name="Employee" //Comment: name of business
object itself
        type="noValue"
        <children name="EmployeeId"
        type="string"/> //Comment: type is one of the
permitted values within Common Event Infrastructure documentation
        <children name="EmployeeAddress"
        type="noValue"/>
        <children name="EmployeeId"
        type="string"/>
        -
        -
        -
        </extendedDataElements
</eventDefinition>

```

Appendix E. Application Response Measurement

This adapter is compatible with the Application Response Measurement application programming interface (API), an API that allows applications to be managed for availability, service level agreements, and capacity planning. An ARM-instrumented application can participate in IBM Tivoli Monitoring for Transaction Performance, allowing collection and review of data concerning transaction metrics.

Required software

In addition to the software prerequisites required for the adapter, you must have the following installed for ARM to operate:

- WebSphere Application Server 5.0.1 (contains the IBM Tivoli Monitoring for Transaction Performance server). This does not have to be installed on the same system as the adapter.
- IBM Tivoli Monitoring for Transaction Performance v. 5.2 Fix 1. This must be installed on the same system on which the adapter is installed and configured to point to the system on which the IBM Tivoli Monitoring for Transaction Performance server resides.

Application Response Measurement support is available using any integration broker supported with this release.

Note: Application Response Measurement instrumentation is supported on all operating systems supported with this IBM WebSphere Business Integration Adapters release *except* HP-UX (any version) and Red Hat Linux 3.0.

Enabling Application Response Measurement

ARM instrumentation is enabled by setting the standard property `TivoliMonitorTransactionPerformance` in Connector Configurator to "True." By default ARM support is not enabled. (Refer to Appendix A, "Standard configuration properties," on page 65 for more information.)

Transaction monitoring

When ARM is enabled, the transactions that are monitored are service events and event deliveries. The transaction is measured from the start of a service request or event delivery to the end of the service request or event delivery. The name of the transaction displayed on the Tivoli Monitoring for Transaction Performance console will start with either SERVICE REQUEST or EVENT DELIVERY. The next part of the name will be the business object verb (such as CREATE, RETRIEVE, UPDATE, or DELETE). The final part of the name will be the business object name such as "EMPLOYEE." For example, the name of a transaction for an event delivery for creation of an employee might be EVENT DELIVERY CREATE EMPLOYEE. Another might be SERVICE REQUEST UPDATE ORDER.

The following metrics are collected by default for each type of service request or event delivery:

- Minimum transaction time
- Maximum transaction time

- Average transaction time
- Total transaction runs

You (or the system administrator of the WebSphere Application Server) can select which of these metrics to display, for which adapter events, by configuring Discovery Policies and Listener Policies for particular transactions from within the Tivoli Monitoring for Transaction Performance console. (Refer to “For more information.”)

For more information

Refer to the IBM Tivoli Monitoring for Transaction Performance documentation for more information. In particular, refer to the *IBM Tivoli Monitoring for Transaction Performance User's Guide* for information about monitoring and managing the metrics generated by the adapter.

Appendix F. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other 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 grant 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.*

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan*

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 the 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 Corporation
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 measurements 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 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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples 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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate 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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

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:

i5/OS
IBM
the IBM logo
AIX
AIX 5L
CICS
CrossWorlds
DB2
DB2 Universal Database
Domino
HelpNow
IMS
Informix
iSeries
Lotus
Lotus Notes
MQIntegrator
MQSeries
MVS
Notes
OS/400
Passport Advantage
pSeries
Redbooks
SupportPac
WebSphere
z/OS

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

This product includes software developed by the Eclipse Project (<http://www.eclipse.org/>).



WebSphere Business Integration Adapter Framework, version 2.6.0.3

Index

A

- Action 16
- AdapterHelpName 67, 72
- adapters, multiple instances 59
- AdminInQueue 67, 73
- AdminOutQueue 67, 73
- AgentConnections 67, 73
- AgentTraceLevel 67, 73
- Application Response Measurement 107
- Application Specific Information (ASI) 16
- ApplicationName 67, 73
- architecture overview 3
- ASI (Application Specific Information) 16
- asynchronous processing 22

B

- BIA_ApplicationMessage
 - business object 27
 - inbound processing 17, 37
 - outbound processing 17, 37
- BIA_ContentBO 26
- BIA_FinalMessage 29
- BIA_InputMessage
 - business object 27
 - definition 17
 - map usage 36
- BIA_InputMessage_CheckComplete 17, 36
- BIA_MO_DataHandler_Default 33
- BIA_MO_Service 31
- BIA_MO_Tcpip_MapSubscriptions 6, 14, 33
- BIA_ResponseMessage
 - business object 28
 - definition 17
 - map usage 38
- BIA_Static_MO 30
- BiDi.Application 67, 73
- BiDi.Broker 68, 74
- BiDi.Metadata 68, 74
- BiDi.Transformation 68, 74
- brokers supported 9
- BrokerType 68, 74
- business objects
 - BIA_ApplicationMessage 27
 - BIA_ContentBO 26
 - BIA_FinalMessage 29
 - BIA_InputMessage 27
 - BIA_ResponseMessage 28
- definitions, specifying 49
- internal 13, 26
- specifying 49

C

- chaining, map 15
- CharacterEncoding 68, 74

- Client1 93
- ClientConfiguration 92
- Clients 92
- Common Event Infrastructure
 - event catalog 102
 - metadata 102
 - overview 101
- CommonEventInfrastructure 68, 74
- CommonEventInfrastructure
 - ContextURL 68, 75
- ConcurrentEventTriggeredFlows 68, 75
- configuration file
 - changing 54
 - creating 39, 43
 - properties, setting 46
 - saving 54
- ConfigurationMetaObject 94
- Connection Manager 4
- connection pooling 5
- ConnectionPoolingOn 5, 94
- ConnectionPoolSize 5, 94, 95
- connector
 - starting 57
 - stopping 58
- Connector Configuration
 - globalized environment 55
- connector configuration file 39
- Connector Configurator
 - overview 39
 - starting 40
- connector properties, setting
 - connector-specific 48
 - standard 47
- connector-specific properties
 - Client1 93
 - ClientConfiguration 92
 - Clients 92
 - ConnectionPoolingOn 5, 94
 - ConnectionPoolSize 5, 94, 95
 - definition 39
 - descriptions 91
 - EnableBase64Decoding 94
 - EnableBase64Encoding 92
 - Host 93
 - KeepAlive 92, 93
 - MaxAttemptsToRead 93, 95
 - MaxRequestPoolSize 5, 91, 95
 - MaxRequestProcessors 5, 91, 95
 - NumberOfRetries 92, 93
 - Port 91, 93
 - ReceiveBufferSize 92, 93
 - RetryInterval 92, 93, 95
 - SendBufferSize 92, 93
 - ServerConfiguration 91
 - ServerQueueLength 92
 - setting 48
 - SocketTimeout 92, 93, 95
 - template 39, 41
 - TransportProtocol 91, 93
- ContainerManagedEvents 68, 75
- ControllerEventSequencing 68, 76

- ControllerStoreAndForwardMode 68, 76
- ControllerTraceLevel 69, 77
- conventions, typographic viii

D

- data handler 13
- DataHandlerMetaObjectName 18, 94
- DataHandlerMimeType 6, 18, 94
- Declaration 16
- DeliveryQueue 69, 77
- DeliveryTransport 69, 77
- DuplicateEventElimination 69, 78

E

- EnableBase64Decoding 94
- EnableBase64Encoding 92
- EnableOidForFlowMonitoring 69, 78
- error handler
 - event processing 97
 - request processing 98
- error logging 61
- ErrorClassName 94, 97
- event catalog, for Common Event Infrastructure 102
- event processing 3, 6, 24

F

- FaultQueue 69, 79
- file structure
 - UNIX 11
 - Windows 11

G

- general meta-objects 13, 29
- globalized environment 55
- glossary 1

H

- HL7 health care data standard 5
- Host 93

I

- IBM Tivoli Monitoring for Transaction Performance 107
- Inbound attribute 17
- inbound maps 17
- inbound processing 3, 6, 24
- installing the connector 11
- internal business objects 13, 26
- iPort 15, 17

J

- jms.FactoryClassName 69, 79
- jms.ListenerConcurrency 69, 79
- jms.MessageBrokerName 69, 79
- jms.NumConcurrentRequests 69, 79
- jms.Password 69, 80
- jms.TransportOptimized 69, 80
- jms.UserName 69, 80
- JvmMaxHeapSize 70, 80
- JvmMaxNativeStackSize 70, 80
- JvmMinHeapSize 70, 80

K

- KeepAlive 92, 93

L

- levels, trace 62
- ListenerConcurrency 70, 81
- Locale 70, 81
- locale-dependent processing 10
- log file values, setting 53
- LogAtInterchangeEnd 70, 81
- logging, error 61

M

- map chaining 15
- maps, PIMO
 - default 17
 - inbound 17
 - outbound 17
 - overview 6
 - usage rules 36
- MaxAttemptsToRead 93, 95
- MaxEventCapacity 70, 82
- MaxRequestPoolSize 5, 91, 95
- MaxRequestProcessors 5, 91, 95
- message handler 16
- Message Processing Framework 5
- MessageFileName 70, 82
- messages, tracing 62
- meta-objects
 - BIA_MO_DataHandler_Default 33
 - BIA_MO_Service 31
 - BIA_MO_Tcpip_MapSubscriptions 14
 - BIA_Static_MO 30
 - general 13, 29
 - PIMO 13, 33
- monitoring of transactions 107
- MonitorQueue 70, 82
- multiple instances of adapters 59

N

- NumberOfRetries 92, 93

O

- OADAutoRestartAgent 70, 82
- OADMaxNumRetry 70, 83
- OADRetryTimeInterval 71, 83
- operating systems supported 9

- oPort 15, 17, 22
- Outbound attribute 17
- outbound maps 17
- outbound processing 4, 6, 18

P

- performance tuning 95
- PIMO framework
 - BIA_MO_Tcpip_MapSubscriptions 14
 - meta-objects 13, 33
 - overview 5, 14
 - processing flow 14
- PIMO maps
 - chaining 15
 - default 17
 - overview 6
 - usage rules 36
- PollEndTime 71, 83
- PollFrequency 71, 83
- PollQuantity 71, 84
- PollStartTime 71, 84
- Port 15, 91, 93
- pre-installation requirements 9
- properties, connector-specific
 - Client1 93
 - ClientConfiguration 92
 - Clients 92
 - ConnectionPoolingOn 5, 94
 - ConnectionPoolSize 5, 94, 95
 - definition 39
 - EnableBase64Decoding 94
 - EnableBase64Encoding 92
 - Host 93
 - KeepAlive 92, 93
 - MaxAttemptsToRead 93, 95
 - MaxRequestPoolSize 5, 91, 95
 - MaxRequestProcessors 5, 91, 95
 - NumberOfRetries 92, 93
 - Port 91, 93
 - ReceiveBufferSize 92, 93
 - RetryInterval 92, 95
 - RetryIntervalProperties 93
 - SendBufferSize 92, 93
 - ServerConfiguration 91
 - ServerQueueLength 92
 - SocketTimeout 92, 93, 95
 - TransportProtocol 91, 93
- properties, setting
 - connector-specific 48
 - standard 47
- properties, standard
 - AdapterHelpName 67, 72
 - AdminInQueue 67, 73
 - AdminOutQueue 67, 73
 - AgentConnections 67, 73
 - AgentTraceLevel 67, 73
 - ApplicationName 67, 73
 - BiDi.Application 67, 73
 - BiDi.Broker 68, 74
 - BiDi.Metadata 68, 74
 - BiDi.Transformation 68, 74
 - BrokerType 68, 74
 - CharacterEncoding 68, 74
 - CommonEventInfrastructure 68, 74
 - CommonEventInfrastructure ContextURL 68, 75

- properties, standard (*continued*)
 - ConcurrentEventTriggeredFlows 68, 75
 - ContainerManagedEvents 68, 75
 - ControllerEventSequencing 68, 76
 - ControllerStoreAndForwardMode 68, 76
 - ControllerTraceLevel 69, 77
 - definition 39
 - DeliveryQueue 69, 77
 - DeliveryTransport 69, 77
 - DuplicateEventElimination 69, 78
 - EnableOidForFlowMonitoring 69, 78
 - FaultQueue 69, 79
 - jms.FactoryClassName 69, 79
 - jms.ListenerConcurrency 69, 79
 - jms.MessageBrokerName 69, 79
 - jms.NumConcurrentRequests 69, 79
 - jms.Password 69, 80
 - jms.TransportOptimized 69, 80
 - jms.UserName 69, 80
 - JvmMaxHeapSize 70, 80
 - JvmMaxNativeStackSize 70, 80
 - JvmMinHeapSize 70, 80
 - ListenerConcurrency 70, 81
 - Locale 70, 81
 - LogAtInterchangeEnd 70, 81
 - MaxEventCapacity 70, 82
 - MessageFileName 70, 82
 - MonitorQueue 70, 82
 - OADAutoRestartAgent 70, 82
 - OADMaxNumRetry 70, 83
 - OADRetryTimeInterval 71, 83
 - PollEndTime 71, 83
 - PollFrequency 71, 83
 - PollQuantity 71, 84
 - PollStartTime 71, 84
 - RepositoryDirectory 71, 84
 - RequestQueue 71, 85
 - ResponseQueue 71, 85
 - RestartRetryCount 71, 85
 - RestartRetryInterval 71, 85
 - ResultsSetEnabled 71, 85
 - ResultsSetSize 72, 86
 - RHF2MessageDomain 72, 86
 - SourceQueue 72, 86
 - SynchronousRequestQueue 72, 86
 - SynchronousRequestTimeout 72, 87
 - SynchronousResponseQueue 72, 87
 - TivoliMonitorTransaction
 - Performance 72, 87
 - WireFormat 72, 87
 - WsifSynchronousRequestTimeout 72, 87
 - XMLNamespaceFormat 72, 87
- property template 41
- Protocol Handler Framework 5
- Protocol Listener Framework 5

R

- ReceiveBufferSize 92, 93
- related documents vii
- RepositoryDirectory 71, 84
- request pool 5
- request processing 4, 6, 18
- RequestQueue 71, 85

ResponseQueue 71, 85
RestartRetryCount 71, 85
RestartRetryInterval 71, 85
ResultsSetEnabled 71, 85
ResultsSetSize 72, 86
RetryInterval 92, 93, 95
RHF2MessageDomain 72, 86

S

SendBufferSize 92, 93
ServerConfiguration 91
ServerQueueLength 92
service call request processing 4, 6, 18
ServiceRegistrationMO 18, 94
SocketTimeout 92, 93, 95
SourceQueue 72, 86
standard properties
 AdapterHelpName 67, 72
 AdminInQueue 67, 73
 AdminOutQueue 67, 73
 AgentConnections 67, 73
 AgentTraceLevel 67, 73
 ApplicationName 67, 73
 BiDi.Application 67, 73
 BiDi.Broker 68, 74
 BiDi.Metadata 68, 74
 BiDi.Transformation 68, 74
 BrokerType 68, 74
 CharacterEncoding 68, 74
 CommonEventInfrastructure 68, 74
 CommonEventInfrastructure
 ContextURL 68, 75
 ConcurrentEventTriggeredFlows 68,
 75
 ContainerManagedEvents 68, 75
 ControllerEventSequencing 68, 76
 ControllerStoreAndForwardMode 68,
 76
 ControllerTraceLevel 69, 77
 definition 39
 DeliveryQueue 69, 77
 DeliveryTransport 69, 77
 DuplicateEventElimination 69, 78
 EnableOidForFlowMonitoring 69, 78
 FaultQueue 69, 79
 jms.FactoryClassName 69, 79
 jms.ListenerConcurrency 69, 79
 jms.MessageBrokerName 69, 79
 jms.NumConcurrentRequests 69, 79
 jms.Password 69, 80
 jms.TransportOptimized 69, 80
 jms.UserName 69, 80
 JvmMaxHeapSize 70, 80
 JvmMaxNativeStackSize 70, 80
 JvmMinHeapSize 70, 80
 ListenerConcurrency 70, 81
 Locale 70, 81
 LogAtInterchangeEnd 70, 81
 MaxEventCapacity 70, 82
 MessageFileName 70, 82
 MonitorQueue 70, 82
 OADAutoRestartAgent 70, 82
 OADMMaxNumRetry 70, 83
 OADRetryTimeInterval 71, 83
 PollEndTime 71, 83
 PollFrequency 71, 83

standard properties (*continued*)
 PollQuantity 71, 84
 PollStartTime 71, 84
 RepositoryDirectory 71, 84
 RequestQueue 71, 85
 ResponseQueue 71, 85
 RestartRetryCount 71, 85
 RestartRetryInterval 71, 85
 ResultsSetEnabled 71, 85
 ResultsSetSize 72, 86
 RHF2MessageDomain 72, 86
 setting 47
 SourceQueue 72, 86
 SynchronousRequestQueue 72, 86
 SynchronousRequestTimeout 72, 87
 SynchronousResponseQueue 72, 87
 TivoliMonitorTransaction
 Performance 72, 87
 WireFormat 72, 87
 WsifSynchronousRequestTimeout 72,
 87
 XMLNameSpaceFormat 72, 87
Supported Business Objects tab 94
synchronous processing 22
SynchronousRequestQueue 72, 86
SynchronousRequestTimeout 72, 87
SynchronousResponseQueue 72, 87

T

task roadmap 1
TCP/IP protocol 3
template
 connector-specific 39, 41
 property 41
terminology 1
thread management 5
Tivoli Monitoring for Transaction
 Performance 107
TivoliMonitorTransaction
 Performance 72, 87
trace file values, setting 53
trace levels 62
tracing messages 62
transaction monitoring 107
TransportProtocol 91, 93
typographic conventions viii

U

UNIX
 Connector Configurator, and 40
 file structure on 11

W

Windows
 file structure on 11
 versions supported 9
WireFormat 72, 87
WsifSynchronousRequestTimeout 72, 87

X

XMLNameSpaceFormat 72, 87



Printed in USA