WebSphere®

IBM

Adapter for TCP/IP User Guide

WebSphere®

IBM

Adapter for TCP/IP User Guide

Note!
Before using this information and the product it supports, read the information in "Notices" on page 73.

First Edition (21Sept2005)

This edition of this document applies to IBM WebSphere Business Integration Server Express and Express Plus Adapter for TCP/IP (5724-i47) version 1.0.0.

To send us your comments about IBM WebSphere Business Integration Server Express and Express Plus 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.

© 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 v
Audience v
Related documents v
Typographic conventions v

New in this release vii

Chapter 1. Overview 1
Task roadmap 1
Terminology 1
Connector architecture 3

Chapter 2. Installing the Connector . . . 5
The adapter environment 5
Broker compatibility 5
Prerequisite software 5
Processing locale-dependent data 6
Installing the connector 6
Verifying the installation 6
Installed Windows File Structure 6
Installed Linux File Structure 7
Installed i5/OS File Structure 7

Chapter 3. Business Objects and Meta-Objects in the TCP/IP Adapter . . . 9
Internal Business Objects 9
BIA_ContentBO 9
BIA_InputMessage 10
BIA_ApplicationMessage 10
General Meta Objects 11
Configuration Meta Object 11
Service Registration Meta Object 12
Data Handler Meta Object 14
PIMO (Production Instruction Meta Object) Framework Meta Objects 15
BIA_MO_Tcpip_MapSubscriptions 15
BIA_Map_InputMessage_to_LLPMessageList . . 16

Chapter 4. Configuring the Connector 19
Overview of Connector Configurator Express . . . 19
Starting Connector Configurator Express 20
Running Configurator in stand-alone mode. . . 20
Running Configurator from System Manager . . . 20
Creating a connector-specific property template . . 20
Creating a new template 21
Creating a new configuration file 23
Creating a configuration file from a connector-specific template 23
Using an existing file 24
Completing a configuration file. 25
Setting the configuration file properties 25
Setting standard connector properties. 26
Setting application-specific configuration properties 27
Specifying supported business object definitions 28
Associated maps (InterChange Server Express only) 29
Setting trace/log file values 30
Data handlers 30
Saving your configuration file 30
Completing the configuration 31
Using Connector Configurator Express in a globalized environment 31

Chapter 5. Running the connector . . . 33
Starting the connector 33
Tasks performed during connector startup . . . 34
Stopping the connector 34
Creating multiple instances of connectors on one server 35
Create a new directory 35
Create business object definitions 36
Create a connector definition 36
Create a start-up script 36

Chapter 6. Maintaining the Connector 39
Error handling in the connector. 39
Tracing messages 40
Using tracing with the connector 40

Appendix A. Standard Configuration Properties 43
New properties 43
Standard connector properties overview 43
Starting Connector Configurator Express . . . 43
Configuration property values overview. . . . 43
Standard properties quick-reference 45
Standard properties. 49
AdapterHelpName 49
AdminInQueue 50
AdminOutQueue 50
AgentConnections 50
AgentTraceLevel. 50
ApplicationName 50
BrokerType 50
CharacterEncoding 50
ConcurrentEventTriggeredFlows 51
ContainerManagedEvents. 51
ControllerEventSequencing 52
ControllerStoreAndForwardMode 52
ControllerTraceLevel 52
DeliveryQueue 52
DeliveryTransport 53
DuplicateEventElimination 53
EnableOidForFlowMonitoring 54
FaultQueue 54
jms.FactoryClassName. 54
jms.ListenerConcurrency 54
jms.MessageBrokerName 54

© Copyright IBM Corp. 2004, 2005

jms.NumConcurrentRequests 55
jms.Password 55
jms.TransportOptimized 55
jms.UserName 55
JvmMaxHeapSize 55
JvmMaxNativeStackSize 55
JvmMinHeapSize 56
Locale 56
LogAtInterchangeEnd 56
MaxEventCapacity 57
MessageFileName 57
MonitorQueue 57
OADAutoRestartAgent 57
OADMaxNumRetry 58
OADRetryTimeInterval 58
PollEndTime 58
PollFrequency 58
PollQuantity 59
PollStartTime 59
RepositoryDirectory 59
RequestQueue 60
ResponseQueue 60
RestartRetryCount 60
RestartRetryInterval 60
RHF2MessageDomain 60
SourceQueue 61
SynchronousRequestQueue 61
SynchronousRequestTimeout 61
SynchronousResponseQueue. 61
TivoliMonitorTransactionPerformance 62
WireFormat 62

Appendix B. Connector Specific Properties And Required Business Object Properties 63

Connector Specific Configuration Properties . . . 65
ServerConfiguration 65
Port 65
TransportProtocol 65
MaxRequestProcessors. 65
MaxRequestPoolSize 65
ServerQueueLength 65
ReceiveBufferSize 65
SendBufferSize 65
KeepAlive 66
ServerSocketTimeout 66
SocketTimeOut 66
RetryInterval 66
NumberofRetries 66
ClientConfiguration 66
Clients 66
Client1 66
Host. 67
Port 67
TransportProtocol 67
ReceiveBufferSize 67
SendBufferSize 67
KeepAlive 67
SocketTimeout 67
MaxAttemptsToRead 67
RetryInterval 67
NumberofRetries 67
Client2 68
ConfigurationMetaObject 68
ServiceRegistrationMO 68
DataHandlerMimeType 68
DataHandlerMetaObjectName 68
Supported Business Objects 68

Appendix C. Adapter Processing . . . 69

The TCP/IP protocol 69
The adapter for TCP/IP 69
Connection Management 69
Message Processing Management 70
The PIMO Framework. 70

Notices 73

Programming interface information 74
Trademarks and service marks 75

Index 77

About this document

The products IBM(R) WebSphere(R) Business Integration Server Express and IBM(R) WebSphere(R) Business Integration Server Express are made up of the following components: InterChange Server Express, the associated Toolset Express, CollaborationFoundation, and a set of software integration adapters. The tools in Toolset Express help you to create, modify, and manage business processes. You can choose from among the prepackaged adapters for your business processes that span applications. The standard processes template--CollaborationFoundation--allows you to quickly create customized processes.

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.

This document describes the installation, configuration, troubleshooting, and business object development for the IBM WebSphere Business Integration Server Express and Express Plus Adapter for TCP/IP.

Exccept where noted, all the information in this guide applies to both IBM WebSphere Business Integration Server Express and IBM WebSphere Business Integration Server Express Plus. The term ″WebSphere Business Integration Server Express″ and its variants refer to both products.

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 Server Express installations, and includes reference material on specific components.

You can download, install, and view the documentation at the following site:

- http:/www.ibm.com/websphere/wbiserverexpress/infocenter

Typographic conventions

This document uses the following conventions:

`courier font`	Indicates a literal value, such as a command name, file name, information that you type, or information that the system prints on the screen.
italic	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.

© Copyright IBM Corp. 2004, 2005

{ }	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, `option[,...]` 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 `<server_name><`*`connector_name`*`>tmp.log`.
/ , \	In this document, backslashes (\) are used as the convention for directory paths. For Linux installations, substitute slashes (/) for backslashes. All product path names are relative to the directory where the connector is installed on your machine.
ProductDir	Represents the directory where the IBM WebSphere Business Integration Server Express and Express Plus adapters product is installed.

New in this release

Version 1.0.0 is the first release of the WebSphere Business Integration Server Express adapter for TCP/IP.

© Copyright IBM Corp. 2004, 2005

Chapter 1. Overview

- "Task roadmap"
- "Terminology"
- "Connector architecture" on page 3

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 understand the adapter before installing, configuring, and using it.

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 5	*WebSphere Business Integration Server Express Installation Guide for Windows, for Linux, or for OS/400*.
Configuring business and meta- objects	Chapter 3, "Business Objects and Meta-Objects in the TCP/IP Adapter," on page 9	*Business Object Development Guide*
Configuring the connector	Chapter 4, "Configuring the Connector," on page 19 Appendix A, "Standard Configuration Properties," on page 43 Appendix B, "Connector Specific Properties And Required Business Object Properties," on page 63	*Connector Development Guide*
Running the connector	Chapter 5, "Running the connector," on page 33	*Connector Development Guide*
Maintaining the connector	Chapter 6, "Maintaining the Connector," on page 39	

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 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

© Copyright IBM Corp. 2004, 2005

includes the library in the Object Development Kit (ODK) needed to develop new ODAs. The configuration components include the following tools:

- Business Object Designer Express,
- Connector Configurator Express,
- Log Viewer,
- System Manager,
- System Monitor,
- Test Connector
- and, optionally, any Object Discovery Agents (ODAs) associated with an adapter.

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 InterChangeServer. This subcomponent interacts with collaborations, a feature of InterChange Server Express. A connector controller runs within InterChangeServer 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.

Connector architecture

Figure 1 shows the connector components and their relationships to the WebSphere business integration system and to the TCP/IP network to which they are connected.

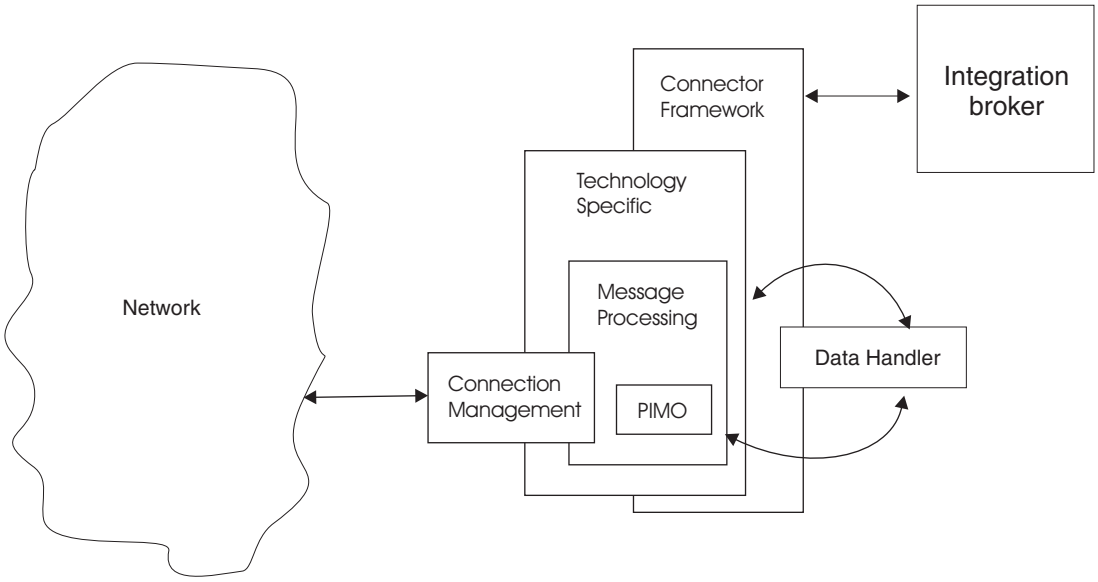

Figure 1. Architecture of the connector

There are industry specific message standards, such as the HL7 protocol in the health care industry, which allow data to be sent out directly over TCP/IP networks. 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.

In event, or 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, using a sub-component called the PIMO. After pre-processing is complete, the connector calls a pre-configured data handler to convert the message into the corresponding business object. Then the connector framework publishes that object to the integration broker. One TCP server can be configured per connector instance.

In request, or 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. A data handler converts the object into a message, which may also, if necessary, be sent for post-processing by the PIMO subcomponent. Then the connector, acting as a TCP client, contacts the appropriate server, establishes the connection, and manages sending the data out. If required, the connector also handles any responses that the target server may provide. Multiple TCP clients can be configured per connector instance.

Chapter 2. Installing the Connector

- "The adapter environment"
- "Installing the connector" on page 6
- "Verifying the installation" on page 6

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

The adapter environment

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

- "Broker compatibility"
- "Processing locale-dependent data" on page 6

Broker compatibility

The adapter framework that an adapter uses must be compatible with the versions of the integration broker with which the adapter is communicating. Version 1.0.x of the adapter for TCP/IP is supported on the following adapter framework and with the following integration broker:

- **Adapter framework**: WebSphere Business Integration Adapter Framework version 2.6. This adapter uses the integration broker, InterChange Server Express.

See the *Release Notes* for any exceptions.

Prerequisite software

The following software must be installed before installing and configuring the Adapter for TCP/IP:

-
  - All operating system environments require the Java compiler (IBM JDK 1.4.2 for Windows 2003) for compiling custom adapters
  - **Linux:**
    Red Hat Linux AS 3.0 Update 1, Intel (IA32)
    SuSE Linux 8.1, SP3, Intel (IA32)
    SuSE Linux ES 9.0, Intel (IA32)

    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 XP with Service Pack 1A, for WebSphere Business Integration Adapter Framework (administrative tools only)
    Windows 2003 (Standard Edition or Enterprise Edition)
  - **OS/400 V5R2 and i5/OS V5R3:**
    i5/OS refers to OS/400 and i5/OS unless explictly stated.

© Copyright IBM Corp. 2004, 2005

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 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

This section describes the tasks you must perform to install the connector and its associated business objects. Refer to "The adapter environment" on page 5 for software prerequisites and compatibility.

After you have installed all prerequisite software on the machine, you can install the connector and the business objects.

For complete instructions on installing the adapter (which includes the connector and related files), refer to the *WebSphere Business Integration Server Express Installation Guide for Windows, for Linux, or for OS/400*, located at the following web site:

http://www.ibm.com/websphere/wbiserverexpress/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 Server Express 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
ProductDir\bin\Data\App	BIA_TCPIPAdapterTemplate --The connector's configuration file template
ProductDir\connectors\TCPIP	BIA_TCPIP.jar--The main connector code start_TCPIP.bat--The connector startup batch file
ProductDir\connectors\messages\ BIA_TCPIPadapter.txt	BIA_TCPIPadapter.txt—The connector message file, containing error messages and codes

Table 2. Files installed with the connector - Windows (continued)

Directory	Installed files
ProductDir\repository\ TCPIP\Sample	• Definition files for business and meta objects that the connector uses for internal processing • Sample applications configured with the TCP/IP connector, including samples for use with NCPDP, HL7 over LLP, and text/name-value
*ProductDir**connectors**TCPIP*\ *start_TCPIP_service.bat*	The service startup script for the connector service. It starts the adapter as a service. It is called from the service control panel where you can start and stop all adapters as services.
ProductDir\connectors\TCPIP \start_tcpip.bat	The regular start script. It starts the adapters as a regular process.

Installed Linux File Structure

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

Table 3. Files installed with the connector - Linux

Directory	Installed files
ProductDir/bin/Data/App	BIA_TCPIPAdapterTemplate --The connector's configuration file template
ProductDir/connectors/TCPIP	BIA_TCPIP.jar--The main connector code start_TCPIP.sh--The connector startup shell file
ProductDir/connectors/ messages/ BIA_TCPIPAdapter.txt	BIA_TCPIPAdapter.txt—The connector message file , containing error messages and codes
ProductDir/repository/TCPIP/Sample	• Definition files for business and meta objects that the connector uses for internal processing • Sample applications configured with the TCP/IP connector, including samples for use with NCPDP, HL7 over LLP, and text/name-value

Installed i5/OS File Structure

The table below describes the file structure used by the connector as installed on an i5/OS machine.

Table 4. Files installed with the connector - i5/OS

Directory	Installed files
ProductDir/connectors/TCPIP	BIA_TCPIP.jar--The main connector code start_TCPIP.sh--The connector startup shell file
ProductDir/connectors/messages/ BIA_TCPIPAdapter.txt	BIA_TCPIPAdapter.txt—The connector message file , containing error messages and codes

Table 4. Files installed with the connector - i5/OS (continued)

Directory	Installed files
ProductDir/repository/TCPIP/Sample	• Definition files for business and meta objects that the connector uses for internal processing • Sample applications configured with the TCP/IP connector, including samples for use with NCPDP, HL7 over LLP, and text/name-value

Chapter 3. Business Objects and Meta-Objects in the TCP/IP Adapter

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 WBI business object, and published to the integration broker. Outbound, or service call request, information is received from the integration broker as a WBI business object, transformed into a network message stream, and sent back over the network. The nature of the WBI 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 WBI business object form.

But there are three other types of objects that facilitate data management and direct the flow of processing inside the connector itself:

- "Internal Business Objects"
- "General Meta Objects" on page 11
- "PIMO (Production Instruction Meta Object) Framework Meta Objects" on page 15

The following sections describe these objects, their function in the connector's internal processing, and their structure.

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).

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 off to the message processing component where it is held in a BIA_ContentBO, the basic data wrapper.

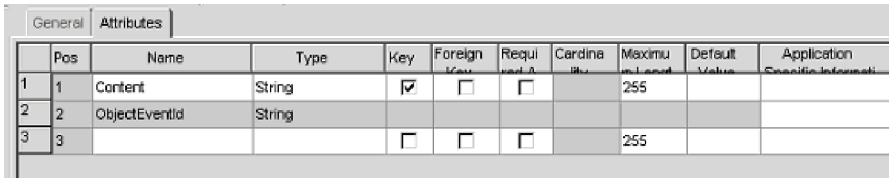

Figure 2. BIA_ContentBO in the Business Object Designer Express

© Copyright IBM Corp. 2004, 2005

The pertinent attributes of the BIA_ContentBO are as follows:

BO Level Attributes	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.

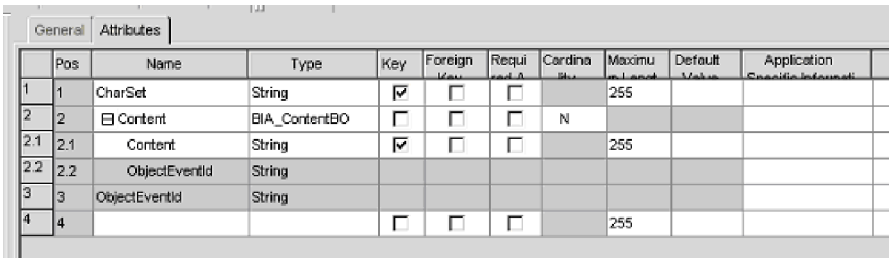

Figure 3. BIA_InputMessage in the Business Object Designer Express

The pertinent attributes of the BIA_InputMessage object are as follows:

BO 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 length of incoming message

BIA_ApplicationMessage

In service call request mode, the WBI 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.

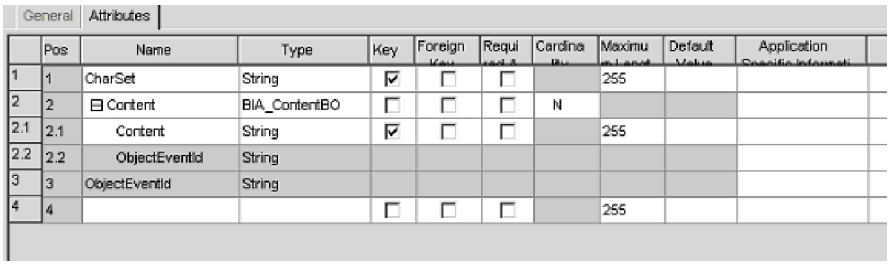

Figure 4. BIA_ApplicationMessage in the Business Object Designer Express

The pertinent attributes of the BIA_ApplicationMessage object are as follows:

BO 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 length of incoming message

There are two other general internal business objects used by the connector: the BIA_ResponseMessage object, which contains any acknowledgement message--wrapped in a content object--from the remote application as a result of a service call request; and BIA_FinalMessage object, which contains information for the connection management subcomponent itself. 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 Linux. They can be viewed either using the Business Object Designer Express or an XML-capable browser.

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, the service registration meta object (this is actually a nested set of objects), and data handler meta object (which is also a nested set).

Configuration Meta Object

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.

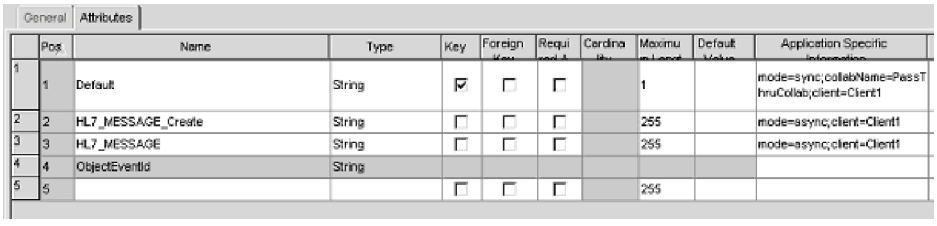

Figure 5. BIA_Static_MO in the Business Object Designer Express

The pertinent attributes of this example object are as follows:

MO Level Attributes	Description
Default	Describes the default values.
HL7_Message_Create	Describes the static meta attribute values for the Create verb associated with HL7_Message objects
HL7_Message	Describes the static meta attribute values associated with HL7_Message objects

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 ″sync″ the connector in event processing contacts the broker in a synchronous manner. When ″async″, the contact is asynchronous
collabName =	Relevant only in synchronous event processing. Names the collaboration that the connector needs to invoke
client = Clientx	Relevant for service call request processing. Names the remote server config stored in the connector configuration attribute Clientx

Service Registration Meta Object

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.

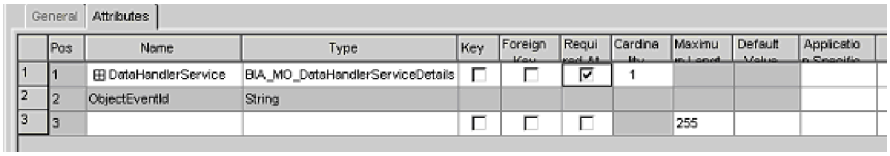

Figure 6. BIA_MO_Service in the Business Object Designer Express

The pertinent attributes of the object are as follows:

MO Level Attributes	Description
DataHandlerService	References a BIA_MO_DataHandlerServiceDetails, which contains further information

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

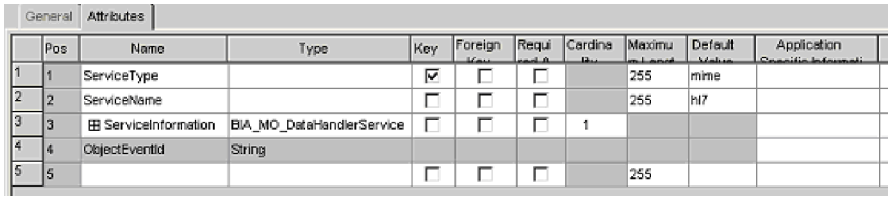

Figure 7. BIA_MO_DataHandlerServiceDetails in the Business Object Designer Express

The pertinent attributes of the object are as follows:

MO Level Attribute	**Description**
Service Type	Contains additional information for specifying the service type. It is not used for connector processing
ServiceName	Specifies the Service Name. In the case of the DataHandlerService, this is the mime type of the message to be processed as it 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.

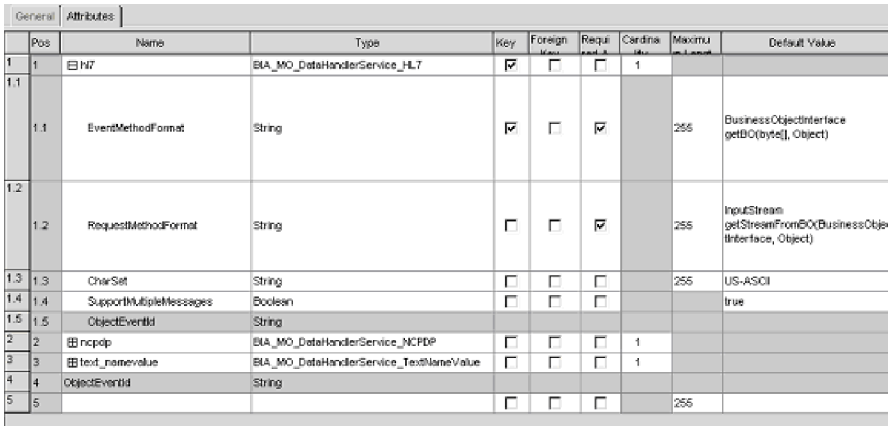

Figure 8. BIA_MO_DataHandlerService in the Business Object Designer Express

The pertinent attributes of the object are as follows:

MO Level Attributes	Description
hl7	Defines the handling for a specific mime type. The object (the example is a BIA_MO_DataHandlerService_HL7) referenced in this attribute contains specific information used by the data handler. The related child attributes below (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
+SupportMultipleMessages	Tells the connector whether the data coming in contains single or multiple messages
ncpdp	Defines the handling for a second mime type through a reference to another type-specific object (the example here is a BIA_MO_DataHandlerService_NCPDP)

Data Handler Meta Object

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.

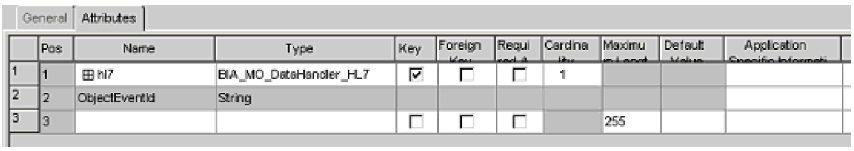

Figure 9. BIA_MO_DataHandlerDefault in the Business Object Designer Express

The pertinent attributes of the object are as follows:

MO Level Attributes	Description
hl7	Defines the handling for a mime type, as designated in the connector configuration file in the DataHandlerMimeType attribute. The referenced object (in this example, a BIA_MO_DataHandler_HL7) contains type specific information used by the data handler

PIMO (Production Instruction Meta Object) Framework Meta Objects

The Production Instruction Meta Object 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. The mechanism itself is discussed in more detail in Appendix C, "Adapter Processing," on page 69, but the main meta objects used by the framework are presented here for completeness.

BIA_MO_Tcpip_MapSubscriptions

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

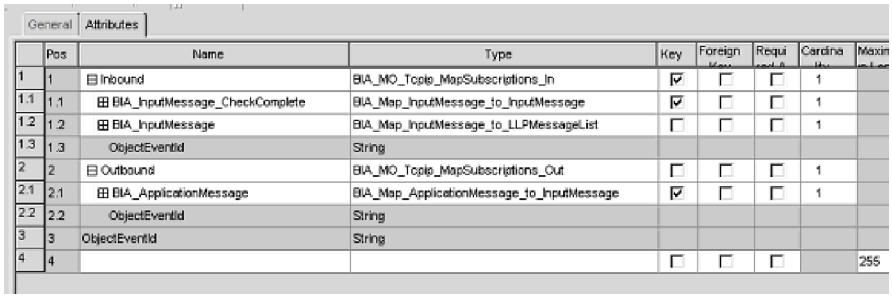

Figure 10. BIA_MO_Tcpip_MapSubscriptions in the Business Object Designer Express

The pertinent attributes of the object are as follows:

MO Level Attributes	Description
Inbound	Indicates the set of maps used by PIMO in event mode processing. In the example, two maps will be used.
+BIA_InputMessage_CheckComplete	Child attribute. Contains a mapping meta object (BIA_Map_InputMessage_to_InputMessage) used to separate multiple objects. Assumes that the SupportMultipleMessages value in the BIA_MO_DataHandlerService object is set to true.
+BIA_InputMessage	Child attribute. Contains a mapping meta object (BIA_Map_InputMessage _to_LLPMessageList) used to remove LLP headers and tailers from the main HL7 data. See below for further details.
Outbound	Indicates the set of maps used by PIMO in service call request mode processing. In the example, one map will be used.
+BIA_ApplicationMessage	Child attribute. Contains a mapping meta object (BIA_Map_ApplicationMessage _to_InputMessage)

BIA_Map_InputMessage_to_LLPMessageList

The BIA_Map_InputMessage_to_LLPMessageList map object contains the instructions PIMO needs to carry out the second stage of the inbound mapping process indicated in the example map subscription meta object above.

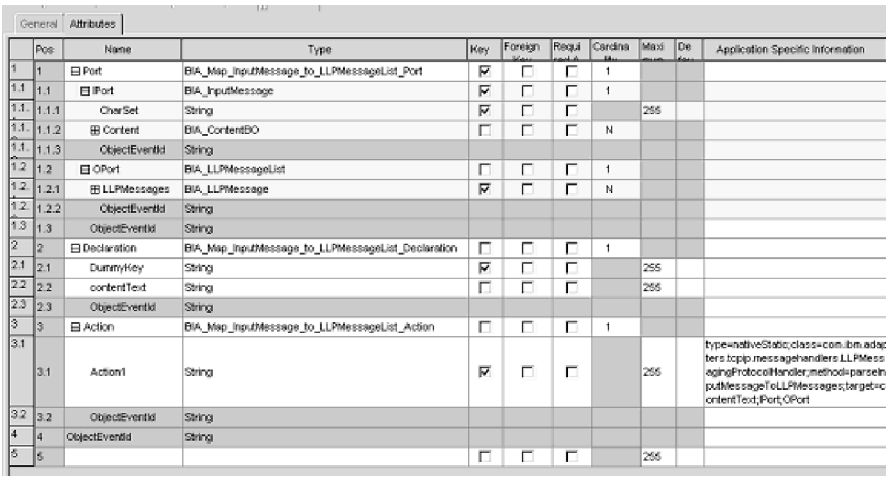

Figure 11. BIA_Map_InputMessage_to_LLPMessageList in the Business Object Designer Express

The pertinent attributes of the object are as follows:

Map BO Level Attributes	Description
Port	Each PIMO map object has two ports: the IPort and the OPort, defined in the example by the BIA_InputMessage_to _LLPMessageList_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_LLPMessageList).
Declaration	A PIMO map object 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 _LLPMessageList_Declaration) contains one such name, "contentText"
Action	Each PIMO map object has at least one action object (here a BIA_Map_InputMessage_to _LLPMessageList) that stores information pointing to the actual class and method that does the processing

The ASI values for *each* action defined by action objects contained by PIMO map objects are as follows:

Application Specific Information for each Defined Action	Description
type =″nativeStatic″	Indicates a static method of a Java class that is being invoked. At present PIMO supports only static methods.
class =	The fully qualified name of the Java class. In the case of the example, the class is the `LLPMessagingProtocolHandler` class in the `com.ibm.adapters.tcpip.messagehandlers` package.
method =	The method being called. In the example the method is `parseInputMessageToLLPMessages` method.
target =	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 and var2 is the OPort from the Port attribute

Chapter 4. Configuring the Connector

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

Overview of Connector Configurator Express

Connector Configurator Express allows you to configure the connector component of your adapter for use with the integration broker, InterChange Server Express.

You use Connector Configurator Express 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 InterChange Server Express, 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 Express, and the configuration file type you use, may differ according to which integration broker you are running. Because InterChange Server Express is your broker, you run Connector Configurator Express from within System Manager (see "Running Configurator in stand-alone mode" on page 20).

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 Express 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 Express.

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 Express 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 21 to set up a new one.

Note: Connector Configurator Express runs only in a Windows environment. If you are running the connector in a Linux environment, use Connector Configurator Express in Windows to modify the configuration file and then copy the file to your Linux environment.

© Copyright IBM Corp. 2004, 2005

Starting Connector Configurator Express

You can start and run Connector Configurator Express in either of two modes:

- Independently, in stand-alone mode
- From System Manager

Running Configurator in stand-alone mode

You can run Connector Configurator Express independently and work with connector configuration files, irrespective of your broker.

To do so:

- From **Start>All Programs**, click **IBM WebSphere Business Integration Express>Toolset Express>Development>Connector Configurator Express**
- Select **File>New>Connector Configuration**.
- When you click the pull-down menu next to **System Connectivity Integration Broker**, you can select InterChange Server Express.

You may choose to run Connector Configurator Express 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 25.)

Running Configurator from System Manager

You can run Connector Configurator Express from System Manager.

To run Connector Configurator Express:

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 Express**. The Connector Configurator Express 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 InterChange Server Express.

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 Express opens and displays the configuration file with the integration broker type and file name at the top.
- From Connector Configurator Express, 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 21.
- 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 Express:

1. Click **File>New>Connector-Specific Property Template**.
2. The **Connector-Specific Property Template** dialog box appears.
   - Enter a name for the new template in the **Name** field below **Input a New Template Name.** You will see this name again when you open the dialog box for creating a new configuration file from a template.
   - To see the connector-specific property definitions in any template, select that template's name in the **Template Name** display. A list of the property definitions contained in that template appears in the **Template Preview** display.
3. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template. If you do not see any template that displays the connector-specific properties used by your connector, you will need to create one.
   - If you are planning to modify an existing template, select the name of the template from the list in the **Template Name** table below **Select the Existing Template to Modify: Find Template.**
   - This table displays the names of all currently available templates. You can also search for a template.

Specifying general characteristics

When you click **Next** to select a template, the **Properties - Connector-Specific Property Template** dialog box appears. The dialog box has tabs for General characteristics of the defined properties and for Value restrictions. The General display has the following fields:

- **General:**
  Property Type
  Updated Method
  Description
- **Flags**
  Standard flags
- **Custom Flag**
  Flag

After you have made selections for the general characteristics of the property, click the **Value** tab.

Specifying values

The **Value** tab enables you to set the maximum length, the maximum multiple values, a default value, or a value range for the property. It also allows editable values. To do so:

1. Click the **Value** tab. The display panel for Value replaces the display panel for General.
2. Select the name of the property in the **Edit properties** display.
3. In the fields for **Max Length** and **Max Multiple Values**, enter your values.

To create a new property value:

1. Select the property in the **Edit properties** list and right-click on it.
2. From the dialog box, select **Add**.
3. Enter the name of the new property value and click OK. The value appears in the **Value** panel on the right.

The **Value** panel displays a table with three columns:

The **Value** column shows the value that you entered in the **Property Value** dialog box, and any previous values that you created.

The **Default Value** column allows you to designate any of the values as the default.

The **Value Range** shows the range that you entered in the **Property Value** dialog box.

After a value has been created and appears in the grid, it can be edited from within the table display.

To make a change in an existing value in the table, select an entire row by clicking on the row number. Then right-click in the **Value** field and click **Edit Value**.

Setting dependencies

When you have made your changes to the **General** and **Value** tabs, click **Next**. The **Dependencies - Connector-Specific Property Template** dialog box appears.

A dependent property is a property that is included in the template and used in the configuration file *only if* the value of another property meets a specific condition. For example, `PollQuantity` appears in the template only if JMS is the transport mechanism and `DuplicateEventElimination` is set to `True`.
To designate a property as dependent and to set the condition upon which it depends, do this:

1. In the **Available Properties** display, select the property that will be made dependent.
2. In the **Select Property** field, use the drop-down menu to select the property that will hold the conditional value.
3. In the **Condition Operator** field, select one of the following:

   == (equal to)

   != (not equal to)

   > (greater than)

   < (less than)

   >= (greater than or equal to)

<=(less than or equal to)

4. In the **Conditional Value** field, enter the value that is required in order for the dependent property to be included in the template.
5. With the dependent property highlighted in the **Available Properties** display, click an arrow to move it to the **Dependent Property** display.
6. Click **Finish**. Connector Configurator Express stores the information you have entered as an XML document, under `\data\app` in the`\bin` directory where you have installed Connector Configurator Express.

Creating a new configuration file

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

- In the System Manager window, right-click on the **Connectors** folder and select **Create New Connector**. Connector Configurator Express opens and displays the **New Connector** dialog box.
- In stand-alone mode: from Connector Configurator Express, select **File>New>Connector Configuration**. In the New Connector window, enter the name of the new connector.

You also need to select the integration broker. The broker determines the properties that appear in the configuration file. To select the broker:

- In the **Integration Broker** field, select InterChange Server Express.
- Complete the the remaining fields in the **New Connector** window, as described later in this chapter.

Creating a configuration file from a connector-specific template

Once a connector-specific template has been created, you can use it to create a configuration file:

1. Click **File>New>Connector Configuration**.
2. The **New Connector** dialog box appears, with the following fields:
   - **Name**

     Enter the name of the connector. Names are case-sensitive. The name you enter must be unique, and must be consistent with the file name for a connector that is installed on the system.

     Important: Connector Configurator Express does not check the spelling of the name that you enter. You must ensure that the name is correct.
   - **System Connectivity**

     Click InterChange Server Express.
   - **Select Connector-Specific Property Template**

     Type the name of the template that has been designed for your connector. The available templates are shown in the **Template Name** display. When you select a name in the Template Name display, the **Property Template Preview** display shows the connector-specific properties that have been defined in that template.

     Select the template you want to use and click **OK**.

3. A configuration screen appears for the connector that you are configuring. The title bar shows the integration broker and connector name. You can fill in all the field values to drop-downte the definition now, or you can save the file and complete the fields later.
4. To save the file, click **File>Save>To File** or **File>Save>To Project**. To save to a project, System Manager must be running.
   If you save as a file, the **Save File Connector** dialog box appears. Choose `*.cfg` as the file type, verify in the File Name field that the name is spelled correctly and has the correct case, navigate to the directory where you want to locate the file, and click **Save**. The status display in the message panel of Connector Configurator Express indicates that the configuration file was successfully created.

   Important: The directory path and name that you establish here must match the connector configuration file path and name that you supply in the startup file for the connector.
5. To complete the connector definition, enter values in the fields for each of the tabs of the Connector Configurator Express window, as described later in this chapter.

Using an existing file

You may have an existing file available in one or more of the following formats:

- A connector definition file.
  This is a text file that lists properties and applicable default values for a specific connector. Some connectors include such a file in a `\repository` directory in their delivery package (the file typically has the extension `.txt`; for example, CN_XML.txt for the XML connector).
- An InterChange Server Express repository file.
  Definitions used in a previous InterChange Server Express 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 Express, 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 Express, 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`)
   - InterChange Server Express Repository (`*.in`, `*.out`)

     Choose this option if a repository file was used to configure the connector in an InterChange Server Express 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 Express.
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 Express 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 InterChange Server Express is the 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 `InterChange Server Express`.
2. The Standard Properties tab will display the properties associated with the broker. You can save the file now or complete the remaining configuration fields, as described in "Specifying supported business object definitions" on page 28..
3. When you have finished your configuration, click **File>Save>To Project** or **File>Save>To File**.

   If you are saving to file, select `*.cfg` as the extension, select the correct location for the file and click **Save**.

   If multiple connector configurations are open, click **Save All to File** to save all of the configurations to file, or click **Save All to Project** to save all connector configurations to a System Manager project.

   Before it saves the file, Connector Configurator Express checks that values have been set for all required standard properties. If a required standard property is missing a value, Connector Configurator Express displays a message that the validation failed. You must supply a value for the property in order to save the configuration file.

Setting the configuration file properties

When you create and name a new connector configuration file, or when you open an existing connector configuration file, Connector Configurator Express displays a configuration screen with tabs for the categories of required configuration values.

Connector Configurator Express requires values for properties in these categories for connectors running on InterChange Server Express:

- 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 **InterChange Server Express**, values for these properties are also required:

- Associated Maps
- Resources
- Messaging (where applicable)

Important: Connector Configurator Express accepts property values in either English or non-English character sets. However, the names of both standard and connector-specific properties, and the names of supported business objects, must use the English character set only.

Standard properties differ from connector-specific properties as follows:

- Standard properties of a connector are shared by both the application-specific component of a connector and its broker component. All connectors have the same set of standard properties. These properties are described in Appendix A of each adapter guide. You can change some but not all of these values.
- Application-specific properties apply only to the application-specific component of a connector, that is, the component that interacts directly with the application. Each connector has application-specific properties that are unique to its application. Some of these properties provide default values and some do not; you can modify some of the default values. The installation and configuration chapters of each adapter guide describe the application-specific properties and the recommended values.

The fields for **Standard Properties** and **Connector-Specific Properties** are color-coded to show which are configurable:

- A field with a grey background indicates a standard property. You can change the value but cannot change the name or remove the property.
- A field with a white background indicates an application-specific property. These properties vary according to the specific needs of the application or connector. You can change the value and delete these properties.
- Value fields are configurable.
- The **Update Method** field is displayed for each property. It indicates whether a component or agent restart is necessary to activate changed values. You cannot configure this setting.

Setting standard connector properties

To change the value of a standard property:

1. Click in the field whose value you want to set.
2. Either enter a value, or select one from the drop-down menu if it appears.
3. After entering all the values for the standard properties, you can do one of the following:

- To discard the changes, preserve the original values, and exit Connector Configurator Express, click **File>Exit** (or close the window), and click **No** when prompted to save changes.
- To enter values for other categories in Connector Configurator Express, select the tab for the category. The values you enter for **Standard Properties** (or any other category) are retained when you move to the next category. When you close the window, you are prompted to either save or discard the values that you entered in all the categories as a whole.
- To save the revised values, click **File>Exit** (or close the window) and click **Yes** when prompted to save changes. Alternatively, click **Save>To File** from either the File menu or the toolbar.

Setting application-specific configuration properties

For application-specific configuration properties, you can add or change property names, configure values, delete a property, and encrypt a property. The default property length is 255 characters.

1. Right-click in the top left portion of the grid. A pop-up menu bar will appear. Click **Add** to add a property. To add a child property, right-click on the parent row number and click **Add child**.
2. Enter a value for the property or child property.
3. To encrypt a property, select the **Encrypt** box.
4. Choose to save or discard changes, as described for "Setting standard connector properties" on page 26.

The Update Method displayed for each property indicates whether a component or agent restart is necessary to activate changed values.

Important: Changing a preset application-specific connector property name may cause a connector to fail. Certain property names may be needed by the connector to connect to an application or to run properly.

Encryption for connector properties

Application-specific properties can be encrypted by selecting the **Encrypt** check box in the Connector-specific Properties window. To decrypt a value, click to clear the **Encrypt** check box, enter the correct value in the **Verification** dialog box, and click **OK**. If the entered value is correct, the value is decrypted and displays.

The adapter user guide for each connector contains a list and description of each property and its default value.

If a property has multiple values, the **Encrypt** check box will appear for the first value of the property. When you select **Encrypt**, all values of the property will be encrypted. To decrypt multiple values of a property, click to clear the **Encrypt** check box for the first value of the property, and then enter the new value in the **Verification** dialog box. If the input value is a match, all multiple values will decrypt.

Update method

Refer to the descriptions of update methods found in the *Standard configuration properties for connectors* appendix, under "Configuration property values overview" on page 43.

Specifying supported business object definitions

Use the **Supported Business Objects** tab in Connector Configurator Express 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.

Because InterChange Server Express is your broker

To specify that a business object definition is supported by the connector, or to change the support settings for an existing business object definition, click the **Supported Business Objects** tab and use the following fields.

Business object name: To designate that a business object definition is supported by the connector, with System Manager running:

1. Click an empty field in the **Business Object Name** list. A drop-down list displays, showing all the business object definitions that exist in the System Manager project.
2. Click on a business object to add it.
3. Set the **Agent Support** (described below) for the business object.
4. In the File menu of the Connector Configurator Express 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 Express 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 Express 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.

Associated maps (InterChange Server Express only)

Each connector supports a list of business object definitions and their associated maps that are currently active in InterChange Server Express. 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 Express window.
- **Associated Maps**

  The display shows all the maps that have been installed to the system for use with the supported business objects of the connector. The source business object for each map is shown to the left of the map name, in the **Business Object Name** display.
- **Explicit**

  In some cases, you may need to explicitly bind an associated map.

  Explicit binding is required only when more than one map exists for a particular supported business object. When InterChange Server Express 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 Express window, click **Save to Project**.
  4. Deploy the project to InterChange Server Express.
  5. Reboot the server for the changes to take effect.

Setting trace/log file values

When you open a connector configuration file or a connector definition file, Connector Configurator Express uses the logging and tracing values of that file as default values. You can change those values in Connector Configurator Express.

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.
     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.
   - To File: (i5/OS platform uses only this option.)

     Note: For the i5/OS platform, if the agent is running on iSeries, you must specify the iSeries file system path manually; do not use the **Browse** button to find the iSeries mapped drive. For example, manually specify the path as: /QIBM/UserData/WBIServer44/servername/log/adapter.log

Data handlers

The data handlers section is available for configuration only if you have designated a value of JMS for DeliveryTransport and a value of JMS for ContainerManagedEvents. Not all adapters make use of data handlers.

See the descriptions under `ContainerManagedEvents` in Appendix A, Standard Properties, for values to use for these properties.

Saving your configuration file

When you have finished configuring your connector, save the connector configuration file. Connector Configurator Express saves the file in the broker mode that you selected during configuration. The title bar of Connector Configurator Express always displays the broker mode (InterChange Server Express) 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 `\WebSphereServer\bin\Data\App`.

For details about using projects in System Manager, and for further information about deployment, see the following guide:

- For InterChange Server Express: *System Implementation Guide*

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 Express in a globalized environment

Connector Configurator Express is globalized and can handle character conversion between the configuration file and the integration broker. Connector Configurator Express uses native encoding. When it writes to the configuration file, it uses UTF-8 encoding.

Connector Configurator Express 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 34
- "Creating multiple instances of connectors on one server" on page 35

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*tcpip*.

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

On i5/OS systems the startup script should reside in the `/QIBM/UserData/WBIServer44/<instance>/connectors/<ConnInstance>/` with which the connector runs.

The name of the startup script depends on the operating-system platform, as Table 5 shows.

Table 5. Startup scripts for a connector

Operating system	Startup script
Linux	`connector_manager`
i5/OS	`start_tcpip.sh`
Windows	`start_tcpip.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.

- **Starting the connector on a Windows system:**
  - From the **Start** menu, select **Programs>IBM WebSphere Business Integration Express>Adapters>Connectors**. By default, the program name is "IBM WebSphere Business Integration Server Express." However, it can be customized. Alternatively, you can create a desktop shortcut to your connector.
  - From the Windows command line: start_connName connName brokerName {-cconfigFile}.
  - 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).
- **Starting the connector on a Linux system**:
  - From the command line type:

© Copyright IBM Corp. 2004, 2005

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

where *connName* is the name of the connector and *brokerName* identifies the integration broker.

- For InterChange Server Express, specify for *brokerName* the name of the InterChange Server Express instance.

- **Starting the connector on an i5/OS system**:
  - From the Windows system where the WebSphere Business Integrations Server Express Console is installed, select **IBM WebSphere Business Integration Server Express>Toolset Express> Administrative>Console**. Then specify the i5/OS system name or IP address and a user profile and password that has *JOBCTL special authority. Select the connector from the list of connectors, and click Start
  - To automatically start the adapter using the Console, use the submit_adapter.sh script. You can also set the adapter agent from the Console to start automatically. To set the option, right-click on the adapter agent and click to enable the **Autostart** check box.
  - In Batch mode, from the i5/OS command line, you need to run the CL command QSH and from the QSHELL environment. Run /QIBM/ProdData/WBIServer44/bin/submit_adapter.sh *connName WebSphereICSName pathToConnNameStartScript jobDescriptionName*, where *connName* is the connector name, *WebSphereICSName* is the Interchange Server Express server name (default is QWBIDFT44), *pathToConnNameStartScript* is the full path to the connector start script, *jobDescriptionName* is the name of the job description to use in the QWBISVR44 library.
  - In interactive mode, specify: connector_manager -start connName WebSphereICSName [-cconfigFile]. For InterChange Server Express, for WebSphereICSName specify the name of the InterChange Server Express instance.

  For more information on how to start a connector, including the command-line startup options, refer to the *System Administration Guide*.

Tasks performed during connector startup

When the connector is started, it performs the following tasks:

- Retrieves configuration information
- Retrieves the supported internal business object definitions
- Returns the connector version
- Returns a pointer to the internal business object handler
- Retrieves a pointer to the data handler

Stopping the connector

The way to stop a connector depends on the way the connector was started.

- **Windows**:
  - You can invoke the startup script which creates a separate “console” window for the connector. In this window, type “q” and press Enter to stop the connector.
  - You can configure the connector to start as a Windows service. In this case, the connector stops when the Windows system shuts down.
- **i5/OS**:

– If you started the connector using the Console, or using the ″submit_adapter.sh″ script in QSHELL, then you can use one of two methods to stop the connector:
– From the Windows system where the WebSphere Business Integration Server Express Console is installed, select IBM **WebSphere Business Integration Express> Toolset Express>Administrative>Console**. Then specify the i5/OS system name or IP address and a user profile and password that has *JOBCTL special authority. Select the iSeries adapter from the list and select the Stop button. Use the CL Command WRKACTJOB SBS (QWBISVR44) to display the jobs to the Server Express product.Scroll the list to find the job with the jobname that matches the job description for the connector. Use the CL Command WRKSMBJOB to display the jobs; the job name is QWBIISRSC. Select Option 4 on this job, and press F4 to get the prompt for the ENDJOB command. Then specify *IMMED for the Option parameter and press enter.

  Note: The connector will end when the QWBISVR44 subsystem has ended.
– If you used the start_connName.sh script to start the adapter from QSHELL, press F3 to
  end the connector. You can also stop the agent, by using a script named *stop_adapter.sh*
  located in the /QIBM/ProdData/WBIServer44/bin directory.

- **Linux**:

  Connectors run in the background so they have no separate window. Instead, run the following command to stop the connector:

  ```
  connector_manager -stop connName
  ```

  where *connName* is the name of the connector.

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

- **For Windows Platforms**:

  ProductDir\connectors\connectorInstance

  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

  where *connectorInstance* uniquely identifies the connector instance.

  You can specify the InterChange Server Express server name as a parameter of startup.bat; an example is: start_iseries.bat connName serverName.

- **For i5/OS Platforms:**

  ```
  /QIBM/UserData/WBIServer44/WebShereICSName/connectors/connectorInstance
  ```

where connectorInstance uniquely identifies the connector instance and where WebSphereICSName is the name of the Interchange Server Express instance with which the connector runs.

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:

/QIBM/UserData/WBIServer44/WebSphereICSName/repository /connectorInstance where WebSphereICSName is the name of the Interchange Server Express instance with which the connector runs.

- **For Linux Platforms**:
  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.You can specify the InterChange Server Express servername as a parameter of connector_manager; an example is connector_manager -start connName WebSphereICSName [-cConfigFile].

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 Express 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

Use the following steps to create a configuration file (connector definition) for the connector instance in Connector Configurator Express.

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 business object definitions."
3. (For Windows only.) Create a startup script shortcut.
4. (For Windows only.) 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.

5. (For i5/OS only.) Create a job description for the connector using the information below:
   CRTDUPOBJ(QWBIISRSC) FROMLIB(QWBISVR44)OBJTYPE(*JOBD)TOLIB (QWBISVR44) NEWOBJ(newemailname) where newemailname is a 10-character name that you use for the job description for your new connector.
6. (For i5/OS only.) Add the new connector to the WebSphere Business Integration Server Express Console. For information about the WebSphere Business Integration Server Express console, refer to the online help provided with the Console.

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 40

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; the name and location of this file are set by the LogFileName connector configuration property.

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_TCPIPAdapter.txt message file installed in the *ProductDir*\connectors\messages directory. Table 6 describes some of more common errors and how the connector handles those errors.

Table 6. 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, e.g. 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) application server is running and b) remote host is available from 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

© Copyright IBM Corp. 2004, 2005

Table 6. 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. 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 7 describes the types of tracing messages that the connector outputs at each trace level. All the trace messages appear in the file specified by the connector property TraceFileName. These messages are in addition to any tracing messages output by the IBM WebSphere Business Integration Server Express adapter architecture.

Table 7. Tracing messages for the connector

Tracing level	Tracing messages
Level 0	Trace adapter version
Level 1	• 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	• Trace each time doVerbFor () is called. Trace which Protocol Handler is processing this request. • Trace each time executeCollab() or gotApplEvent() is called

Table 7. Tracing messages for the connector (continued)

Tracing level	Tracing messages
Level 3	• 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	• Trace spawning of threads • Trace all ASI processed • Trace entry and exit of important functions
Level 5	• Trace entry and exit for each important method • Trace all the Adapter properties • Trace dumps of BO sent to the broker • Trace dumps of BO sent by the broker

Appendix A. Standard Configuration Properties

This appendix describes the standard configuration properties for the connector component of WebSphere Business Integration Server Express adapters. The information covers InterChange Server Express.

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

New properties

These standard properties have been added in this release:

- AdapterHelpName
- ControllerEventSequencing
- jms.ListenerConcurrency
- jms.TransportOptimized
- TivoliTransactionMonitorPerformance

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 Express and describes characteristics common to all properties. For information on configuration properties specific to a connector, see its adapter user guide.

Starting Connector Configurator Express

You configure connector properties from Connector Configurator Express, which you access from System Manager. For more information on using Connector Configurator Express, refer to the sections on Connector Configurator Express in this guide.

Connector Configurator Express and System Manager run only on the Windows system. If you are running the connector on a Linux system, you must have a Windows machine with these tools installed.

To set connector properties for a connector that runs on Linux, you must start up System Manager on the Windows machine, connect to the Linux integration broker, and bring up Connector Configurator Express for the connector.

Configuration property values overview

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

1. Default

© Copyright IBM Corp. 2004, 2005

2. Repository for InterChange Server Express 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).
- **Agent restart (InterChange Server Express 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 Express window, or see the Update Method column in Table 8 on page 45.

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

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

Note: In the Notes column in Table 8, the phrase "RepositoryDirectory is set to <REMOTE>" indicates that the broker is InterChange Server Express.

Table 8. 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 has a valid <Regional Setting> 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 `IDL`, The value of Repository Directory is set to <REMOTE> and the value of BrokerType is InterChange Server Express.
AgentTraceLevel	`0` through `5`	`0`	Dynamic for Inter-Change Server Express; otherwise Component restart	
ApplicationName	Application name	The value specified for the connector application name	Component restart	
BrokerType	`InterChange Server Express`	`InterChange Server Express`	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:iiop:host:2809`.	No default value.	Component restart	This property is valid only if the value of CommonEvent Infrastructure is `true`.

Table 8. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
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 InterChange Server Express.
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 InterChange Server Express.
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 InterChange Server Express.
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 InterChange Server Express.
DeliveryQueue	Any valid JMS queue name	*<CONNECTORNAME>*/DELIVERYQUEUE	Component restart	This property is valid only when the value of Delivery Transport is JMS.
DeliveryTransport	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 InterChange Server Express.
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.

Table 8. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
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.NumConcurrent Requests	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 InterChange Server Express.
jms.UserName	Any valid name		Component restart	This property is valid only if the value of Delivery Transport is JMS.
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 InterChange Server Express.
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 InterChange Server Express.
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 InterChange Server Express.
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 InterChange Server Express.
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 InterChange Server Express.

Table 8. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
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 InterChange Server Express.
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 InterChange Server Express.
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 InterChange Server Express.
PollEndTime	HH = 0 through 23 MM = 0 through 59	HH:MM	Component restart	
PollFrequency	A positive integer (in milliseconds)	10000	Dynamic if broker is InterChange Server Express; 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 InterChange Server Express; otherwise any valid local directory.	For InterChange Server Express, the value is set to <REMOTE>	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	3	Dynamic if InterChange Server Express; otherwise Component restart	

Table 8. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
RestartRetryInterval	A value in minutes from 1 through 2147483647	1	Dynamic if InterChange Server Express; otherwise Component restart	
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 CwBO	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 CwBO if the value of RepositoryDirectory is set to <REMOTE>.
WsifSynchronousRequest Timeout	0 to any number (milliseconds)	0	Component restart	This property is not valid if the value of BrokerType is InterChange Server Express.
XMLNameSpaceFormat	short or long	short	Agent restart	This property is not valid if the value of BrokerType is InterChange Server Express.

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.

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.

BrokerType

The BrokerType property identifies the integration broker type that you are using. The value is InterChange Server Express, , or .

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 Express appendix in this guide.

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 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.

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 Express.

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 set to <REMOTE> (BrokerType is InterChange Server Express).

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 InterChange Server Express (ICS), the connector controller blocks the request to the application-specific component. When the application-specific component becomes operational, the controller forwards the request to it.

However, if the destination application's application-specific component becomes unavailable after the connector controller forwards a service call request to it, the connector controller fails the request.

If this property is set to false, the connector controller begins failing all service call requests as soon as it detects that the destination application-specific component is unavailable.

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

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. For Java Messaging Service, the value is JMS.

- If the value of the RepositoryDirectory property is set to <REMOTE>, the value of the DeliveryTransport property can be IDL or JMS, and the default is IDL.

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

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 Express. 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 when InterChange Server Express (ICS) is the integration broker.

In this environment, you may experience difficulty starting both the connector controller (on the server side) and the connector (on the client side) due to memory use within the WebSphere MQ client. If your installation uses less than 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 *WebSphere Business Integration Server Express Installation Guide for Windows, for Linux, or for i5/OS*.

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 InterChange Server Express.

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 `InterChange Server Express`.

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.

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 Express 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 RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is InterChange Server Express).

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 InterChange Server Express).

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 *WebSphere Business Integration Server Express Installation Guide for Windows, for Linux* or *for i5/OS*.

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

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 InterChange Server Express).

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 InterChange Server Express).

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 InterChange Server Express, this value must be set to set to <REMOTE> because the connector obtains this information from the InterChange Server Express 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 InterChange Server Express (InterChange Server Express), 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 3.

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.

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 not valid if the value of BrokerType is InterChange Server Express. 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 51.

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 `CwBO`.

Appendix B. Connector Specific Properties And Required Business Object Properties

Table 9 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 9. 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
+MaxRequest Processors	Integer greater than 0	2	agent restart	No
+MaxRequest PoolSize	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
+ServerSocket Timeout	Integer greater than 0		connector restart	No
+SocketTimeout	Integer greater than 0		agent restart	No
+RetryInterval	Integer 0 or greater	0	agent restart	No
+NumberofRetries	Integer 0 or greater	0	agent restart	No

© Copyright IBM Corp. 2004, 2005

Table 9. Connector Specific Properties (continued)

Properties	Possible values	Default values	Update Method	Required?
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
+++ReceiveBuffer Size	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		agent restart	No
+++MaxAttempts ToRead	Integer greater than 0	1	agent restart	No
+++RetryInterval	Integer 0 or greater	0	connector restart	No
+++Numberof Retries	Integer 0 or greater	0	agent restart	No
++Client2	As above		agent restart	No
Configuration MetaObject	Static Meta Object	BIA _Static _MO	agent restart	Yes
Service RegistrationMO	Service Meta Object	BIA _MO _Service	agent restart	Yes
DataHandler MimeType	Mime types	text/ name value	connector restart	Yes

Table 9. Connector Specific Properties (continued)

Properties	Possible values	Default values	Update Method	Required?
DataHandler MetaObjectName	DataHandler Configuration Meta Object	BIA _MO _Data Handler _Default	agent restart	Yes

Connector Specific Configuration Properties

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/0 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/0 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, acknowledgement and window numbers.

ServerSocketTimeout

Sets timeout blocking in milli-seconds for this ServerSocket. A timeout of zero is interpreted as an infinite timeout. With this option set to a non-zero timeout, a call to accept() for this ServerSocket will block for only this amount of time. If the timeout expires, a `java.io.InterruptedIOException` is raised, though the ServerSocket is still valid. The option must be enabled prior to entering the blocking operation to have effect.
If the listener thread does not receive a request in this interval, it will check to see if the Connector shutdown flag is set. If Connector shutdown flag is set, it will terminate. This value is applicable only when the Connector is acting as a TCP server accepting requests.

SocketTimeOut

Sets base timeout blocking in milli-seconds 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, etc.

NumberofRetries

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

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 remote host port 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/0 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/0 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, acknowledgement and window numbers.

SocketTimeout

Sets timeout blocking in milli-seconds 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 time the connector will read from the socket once it starts receiving data. This property allows for the reception of separate acknowledgement data. This approach differs from Event processing because it is assumed that the amount of data received in an acknowledgement 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, etc.

NumberofRetries

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

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 11.

ServiceRegistrationMO

The top level meta object that holds service information. See the description of `BIA_MO_Service` in "General Meta Objects" on page 11.

DataHandlerMimeType

Sets the expected mime type of the incoming data. Used to specify the appropriate DataHandler.

DataHandlerMetaObjectName

The top level meta object that holds DataHandler configuration information other than that contained in the service object. See the description of the `BIA_MO_DataHandler_Default` in"General Meta Objects" on page 11

Supported Business Objects

The Connector Configurator should also be used to set up the Supported Business Objects tab. Table 10 shows the values that should be present in the case of HL7 processing. In other situations, other values may obtain.

Table 10. Supported Business Objects Properties

Business Object Name	Message Set ID
BIA_ApplicationMessage	1
BIA_FinalMessage	2
BIA_MO_DataHandler_Default	3
BIA_MO_Service	4
BIA_MO_Tcpip_MapSubscriptions	5
BIA_ResponseMessage	6
BIA_STATIC_MO	7
BIA_InputMessage	8
HL7_SGMSH	9
HL7_MESSAGE	10

For more information on these objects, please see "Internal Business Objects" on page 9.

Appendix C. Adapter Processing

This appendix presents a more detailed description of the processing flow inside the TCP/IP adapter, including a description of the PIMO infrastructure.

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.

The adapter for TCP/IP

The Websphere Business Integration Server Express adapter for TCP/IP provides a way of routing data sent over raw TCP/IP connections into and out of the system. Certain industry standard protocols for the transmission of domain specific messages, like the HL7 protocol in the health care industry, are designed to be sent directly over TCP/IP connections. The adapter for TCP/IP provides a way to capture such data and integrate it into a larger Integration flow.

Connection Management

The connector, the runtime component of the TCP/IP Adapter, 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. 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 63 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 configuration property, `MaxRequestPoolSize`. 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.

© Copyright IBM Corp. 2004, 2005

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.

Message Processing Management

The Message Processing Framework manages the transformation of incoming event data into usable business objects and of outgoing service request business objects into various supported message structures sendable over TCP/IP. It is made up of three parts: the PIMO framework, PIMO maps or Message Handlers, and Data Handlers. 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. The data is then handed over to a Data Handler, an independent plugin used by many different adapters, to be built into a WBI-usable business object. The choice of data handlers is based on the `DataHandlerMimeType` and related properties in the CFG. The business object is passed on to the broker. The structure of the business object is completely determined by the data handler, so the TCP/IP adapter is capable of handling any sort of message data for which there is a related data handler.

Service call request processing follows the same stages in reverse: the data handler transforms the business object into appropriate message structures, the PIMO Framework performs post-processing using Message Handlers, and the messages are passed to the Protocol Handler Framework in the Connection Manager, which sends the service request out to the remote host.

The PIMO Framework

Message structures that come in over a TCP/IP connection may require a certain amount of pre-processing before they can be sent on to a data handler to be turned into WebSphere Business Integration Server Express business objects.

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 these. However many mini and mainframe computer systems, which may 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 infrastructure in the TCP/IP connector is designed to do exactly this sort of pre-processing. The Production Instruction Meta Object Framework is a highly flexible, general purpose abstraction for effecting business logic processing at the connector level. The PIMO infrastructure is capable of a wide-range of operations, and is used, in differing forms, in other adapters. In the TCP/IP connector, it has been customized specifically to take care of these pre- and post-processing issues.

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. The graphic "BIA_MO_Tcpip_MapSubscriptions" on page 15 shows what this object looks like in the Business Object Designer Express. 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, the BIA_MO_Tcpip_MapSubscriptions_In and the equivalent Out object, each of which contains a reference to an appropriate PIMO map object. In the case of HL7 object above, the appropriate In PIMO map object might be the BIA_Map_InputMessage_to_LLPMessageList. You can look at "BIA_Map_InputMessage_to_LLPMessageList" on page 16 to see what this object looks like in the Business Object Designer Express.

This PIMO map object serves as the core PIMO object. PIMO objects consist of three basic attributes: Port, Declaration, and Action.

Each Port is made up in turn of two attributes: the IPort and the OPort. These indicate the expected type of the source object (for example, an BIA_InputMessage object, which is the internal wrapper object for event processing) and the destination object (a BIA_LLPMessage List object). The example on page 15 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. By separating the process into multiple steps, chaining maps can produce more complex types of processing. If chained maps are used, it is essential that the OPort type of step 1 be exactly the same as the IPort type of step 2, and so forth.

The declaration attribute is optional. It contains names for temporary variables to be used during processing. In the example the declaration object contains one such name, "contentText."

Finally, PIMO maps contain Action attributes. Each Action attribute consists of one or more defined Actions. The Application Specific Information or 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 example ASI from page 16 serves to illustrate the necessary information. It is as follows:

```
type=nativeStatic;
class=com.im.adapters.tcpip.messagehandlers.LLPMessagingProtocoHandler;
method=parseInputMessageToLLPMessages; target=contentText;IPort;Oport
```

The ASI contains the following information:

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.

The method at the heart of this set, `parseInputMessageToLLPMessages` knows how to separate out the LLP specific wrapper data, and then return a list which stores the individual parts of the LLPMessage (the header, the message itself, and the tailer) 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.

Once again, service call request processing would follow the same stages except in reverse: the PIMO Framework would be used to wrap messages in their protocol specific data before sending them to be forwarded to the remote host.

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:

© Copyright IBM Corp. 2004, 2005

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
CICS
CrossWorlds
DB2
DB2 Universal Database
Domino
IMS
Informix
iSeries
Lotus
Lotus Notes
MQIntegrator
MQSeries
MVS
OS/400
Passport Advantage
SupportPac
WebSphere
z/OS
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both. MMX, Pentium, and ProShare are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both. Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. 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.

WebSphere Business Integration Server Express and Express Plus include software developed by the Eclipse Project (http://www.eclipse.org/).

WebSphere Business Integration Server Express, Version 4.4, and WebSphere Business Integration Server Express Plus, Version 4.4

WebSphere Business Integration Server Express and Express Plus include software developed by the Eclipse Project (http://www.eclipse.org/).

Index

A
adapters, multiple instances 35
architecture overview 3

B
broker compatibility 5
business objects, internal 9

C
Connection Manager 69
connector specific properties 65
connector startup 33
connector stopping 34
conventions, typographic v

E
error handling 39
event processing 3

F
file structure - UNIX 7
file structure - Windows 6

G
glossary 1

I
installation task roadmap 1
installing the connector 6

L
locale-dependent processing 6

M
Message Processing Framework 70
messages, tracing 40
meta objects, general 11
meta objects, PIMO 15
multiple instances, of adapters 35

P
PIMO 70
pre-installation requirements 5

R
related documents v
request processing 3

S
starting more than one adapter 35
stopping the connector 34

T
tcp/ip 69
tracing messages 40
typographic conventions v

U
URLs v

W
web sites v

© Copyright IBM Corp. 2004, 2005

Printed in USA