Siebel

Transports and Interfaces: Siebel Enterprise Application Integration

April 2021



Siebel
Transports and Interfaces: Siebel Enterprise Application Integration

April 2021

Part Number: F12806-06

Copyright © 2021, Oracle and/or its affiliates.

Authors: Siebel Information Development Team

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display in any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or “commercial computer software documentation” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government’s use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

The business names used in this documentation are fictitious, and are not intended to identify any real companies currently or previously in existence.

Contents

Preface i

1 What's New in This Release 1

What's New in Transports and Interfaces: Siebel Enterprise Application Integration Guide, Siebel CRM 21.4 Update 1

What's New in Transports and Interfaces: Siebel Enterprise Application Integration Guide, Siebel CRM 21.2 Update 1

What's New in Transports and Interfaces: Siebel Enterprise Application Integration Guide, Siebel CRM 20.2 Update 1

What's New in Transports and Interfaces: Siebel Enterprise Application Integration Guide, Siebel CRM 19.10 Update 2

What's New in Transports and Interfaces: Siebel Enterprise Application Integration Guide, Siebel CRM 19.7 Update 2

What's New in Transports and Interfaces: Siebel Enterprise Application Integration Guide, Siebel CRM 19.1 Update 3

2 EAI Transports and Interfaces Overview 5

EAI Transports and Interfaces Overview 5

About EAI Transports 5

About EAI Transport Methods 6

Using Named Subsystems for Transport Parameters 7

About Object Interfaces and EAI 10

Database-Level Interfacing 10

3 EAI MQSeries Server Transport 11

EAI MQSeries Server Transport 11

About the EAI MQSeries Server Transport Business Service 11

Using the SendReceive Method with MQSeries 17

Dispatch Error Handling for the EAI MQSeries Server Transport 17

Increasing the Maximum Message Length on IBM WebSphere MQ 18

Using the EAI MQSeries Server Transport on AIX 18

About EAI MQSeries Transport Re-Entrance 20
About Message ID Tracking for an Inbound Message 20
Invoking a Workflow Using MQSeries Server Receiver 20

4 EAI MSMQ Transport 23
EAI MSMQ Transport 23
About Microsoft Message Queuing (MSMQ) 23
Configuring the EAI MSMQ Transport Servers 25
Configuring EAI MSMQ Transport for Various Send and Receive Scenarios 26

5 EAI Java Business Service 39
EAI Java Business Service 39
About the EAI Java Business Service 39
Requirements for Implementing a Java Business Service 39
Creating a Java Business Service 45
About the Lifecycle of a 32-bit Java Business Service 46
Example of a Java Business Service 47
About the Lifecycle of a 64-bit Java Business Service 47
Restrictions for Implementing a Java Business Service 48
Troubleshooting the Java Business Service 48

6 EAI JMS Transport 49
EAI JMS Transport 49
About the EAI JMS Transport Business Service 49
About Synchronous and Asynchronous Invocation 50
About the JMS Publish-and-Subscribe Model 50
About Operations (Methods) of the EAI JMS Transport 51
Features Not Supported for Use with the Siebel JMS Transport 51
About JMS Message Types 52
About Sending and Receiving XML 52
About Multistep Operations Within a JMS Session 52
About Undeliverable Messages in JMS Transport 53
Detailed Input and Output Specifications for the EAI JMS Transport 53
Configuring the EAI JMS Transport 59
Sending and Receiving JMS Messages 65
Receiving, Dispatching, and Sending JMS Messages 69
Sending and Receiving Custom JMS Properties 72

Enabling Authentication and Authorization for the EAI JMS Transport 74
Troubleshooting for the JMS Transport 78
About Logging for the JMS Transport 79
About Caching for the JMS Transport 79

7 EAI HTTP Transport 81

EAI HTTP Transport 81
About the EAI HTTP Transport 81
Using POST and GET 82
EAI HTTP Transport Named Subsystems 83
EAI HTTP Transport Method Arguments 83
Sending a Message Using the EAI HTTP Transport 87
Using the EAI HTTP Transport for Inbound Integration 89
Process of Using the EAI HTTP Transport for Inbound Messages 94
Handling EAI HTTP Transport Business Service Errors 97
Processing and Sending Outbound XML Documents 98
Sending and Receiving Messages with the EAI HTTP Transport 100
Examples Using HTTP Request 102
Creating Custom Headers for the EAI HTTP Transport Service 105
About Sending and Receiving Messages Through HTTP 105
About Transport Headers and HTTP Response Headers 106

8 Integrating Siebel CRM with Java Applications 107

Integrating Siebel CRM with Java Applications 107
About Siebel CRM and Java Applications 107
About the JDB Business Service API 109
About the Siebel Code Generator 109
About Running the Java Data Bean 116
About the Siebel Resource Adapter 121

9 EAI DLL and EAI File Transports 127

EAI DLL and EAI File Transports 127
About the EAI DLL Transport 127
About the EAI File Transport 130

10 Transcode Service Business Service 135

Transcode Service Business Service 135

About the Transcode Service Business Service 135
Transcode Service Business Service Methods 136
Transcode Service Business Service Examples 138

Preface

This preface introduces information sources that can help you use the application and this guide.

Using Oracle Applications

To find guides for Oracle Applications, go to the Oracle Help Center at *http://docs.oracle.com/*.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the *Oracle Accessibility Program website*.

Contacting Oracle

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit *My Oracle Support* or visit *Accessible Oracle Support* if you are hearing impaired.

Comments and Suggestions

Please give us feedback about Oracle Applications Help and guides! You can send an email to: *oracle_fusion_applications_help_ww_grp@oracle.com*.

1 What's New in This Release

What’s New in Transports and Interfaces: Siebel Enterprise Application Integration Guide, Siebel CRM 21.4 Update

The following information lists the changes described in this version of the documentation to support this release of the software.

Topic	Description
Tuning the EAI JMS Transport	New topic. As of Siebel CRM 20.6 Update, tuning parameters are available for JBS (Java Business Service) and JMS (Java Message Service).

What’s New in Transports and Interfaces: Siebel Enterprise Application Integration Guide, Siebel CRM 21.2 Update

The following information lists the changes described in this version of the documentation to support this release of the software.

Topic	Description
About the JMS Receiver	Modified topic. Changes to repository objects no longer require a restart of the JMS Receiver component.

What’s New in Transports and Interfaces: Siebel Enterprise Application Integration Guide, Siebel CRM 20.2 Update

The following information lists the changes described in this version of the documentation to support this release of the software.

Topic	Description
Creating a 64-bit Java Subsystem by Using the Siebel Dedicated Client	Obsolete topic. This topic has been removed from the guide.

What's New in Transports and Interfaces: Siebel Enterprise Application Integration Guide, Siebel CRM 19.10 Update

The following information lists the changes described in this version of the documentation to support this release of the software.

Topic	Description
Creating a 64bit Java Subsystem by Using the Siebel Dedicated Client	Modified Topic. Revised documentation with a update about the need for a dedicated client subsystem named JAVA64, instead of JAVA. **Note:** As of Siebel CRM 20.1 Update, this topic is now obsolete.

What's New in Transports and Interfaces: Siebel Enterprise Application Integration Guide, Siebel CRM 19.7 Update

The following information lists the changes described in this version of the documentation to support this release of the software.

Topic	Description
About the JMS Publish-and-Subscribe Model *Input Argument Values*	Modified Topics. Revised documentation with input value argument.

What’s New in Transports and Interfaces: Siebel Enterprise Application Integration Guide, Siebel CRM 19.1 Update

No new features have been added to this guide for this release. This guide has been updated to reflect only product name changes.

2 EAI Transports and Interfaces Overview

EAI Transports and Interfaces Overview

Siebel Enterprise Application Integration (EAI) provides mechanisms for exchanging data between Siebel CRM and external systems.

This chapter includes the following topics on these mechanisms:

- *About EAI Transports*
- *About EAI Transport Methods*
- *Using Named Subsystems for Transport Parameters*
- *About Object Interfaces and EAI*
- *Database-Level Interfacing*

About EAI Transports

Transports allow Siebel CRM to exchange data with external applications using standard technologies for both synchronous and asynchronous communication protocols.

Transports handle all data as binary data (bytes) because the IsTextData parameter that was available in previous releases is no longer supported. If you want to use character conversion on the transport, then you use the CharSetConversion parameter. Handling the data as binary defers any character set conversion until needed and avoids conversion at the transport level to prevent data corruption. For example, treating a UTF-8 encoded Extensible Markup Language (XML) document as text when the conversion executes leads to an XML string in the local code page, while its header still describes UTF-8. It is best to treat all self-describing data, including XML, as binary.

Character conversion is available in a number of business services. These business services are:

- EAI Transport business services (MQ Series, MSMQ, JMS, HTTP, DLL, File)
- XML Converter business services
- Transcode Service business service

When business services are invoked from a workflow, the valid set of encodings is controlled by a picklist. If the business services are invoked through scripting or similar mechanisms, then the character set name is supplied textually.

> **Note:** For data validation or conversion from one encoding to another, you can use the Transcode Service business service. For information about the Transcode Service business service, see *Transcode Service Business Service*

Transports provide connectivity to virtually any communication protocol that can represent data as text or binary messages, including MQSeries from IBM, MSMQ from Microsoft, Java Message Service (JMS), and HTTP. EAI Transports allow Siebel CRM to integrate with Web-based applications as well as legacy systems that are encapsulated using middleware. Transports are interchangeable. If you change technologies at any point, then you can reuse existing workflows and logic by switching the transport adapter.

Transports can:

- Support bidirectional exchange of messages.
- Run within the Siebel Application Object Manager.
- Invoke and be invoked by Workflow Process Manager and EAI Dispatch Service.
- Be invoked within an eScript or VBScript.
- Send and receive messages in XML format.
- Pass messages through, or convert messages into, property sets for XML and MIME messages.

Available transports include:

- **EAI MQSeries Server Transport.** For information about these transports, see *EAI MQSeries Server Transport*
- **EAI MSMQ Transport.** For information about this transport, see *EAI MSMQ Transport*
- **EAI JMS Transport.** For information about this transport, see *EAI JMS Transport*
- **EAI HTTP Transport.** For information about this transport, see *EAI HTTP Transport*
- **EAI DLL Transport and EAI File Transport.** For information about these transports, see *EAI DLL and EAI File Transports*

> **Note:** The transport business services are not re-entrant. This applies not only to receivers, but also to nonreceiver mode because users can define scripts in the business service that invoke the same business service. For more information about transport re-entrance, see *About EAI MQSeries Transport Re-Entrance*.

About EAI Transport Methods

The method on a transport adapter's business service controls the action to be performed by the transport. There are two outbound methods and three inbound methods available for EAI Transports. Not every method is available on every transport. These methods are described in the following topics:

- *Outbound Methods for a Transport Business Service*
- *Inbound Methods for a Transport Business Service*

For each method, there are a number of common parameters, as shown in the second table in *Common EAI Transport Parameters*, as well as transport-specific parameters that are discussed in the respective chapter for each transport.

Outbound Methods for a Transport Business Service

Available outbound methods depend on the transport business service in use, such as EAI MSMQ Transport. The business service sends messages from the Siebel application using the appropriate communications protocol, such as MQSeries, MSMQ, HTTP, and so on. There are two outbound methods that you use to send requests from a Siebel application to another application:

- **Send.** Sends a message from a Siebel application when the Siebel application does not require a response. This is an asynchronous request method (except for the EAI HTTP Transport, which expects a correct HTTP response), because the Siebel application does not wait for a response before continuing with the process.
- **Send and Receive (SendReceive).** Sends a message from the Siebel application when the Siebel application requires a response before continuing. This is a synchronous request and response method, because it must receive a response before the Siebel application can continue.

Inbound Methods for a Transport Business Service

Available inbound methods depend on the transport business service in use, such as EAI MSMQ Transport. The inbound methods monitor a specified queue and upon receipt of a message, dispatch it to another service.

There are three inbound methods that can be used to receive requests from another application:

- **Receive.** Receives an inbound request message and returns it to the caller of the transport.
- **Receive and Execute (ReceiveDispatch).** Receives an inbound request message and calls another service with the inbound message as input. This called service is known as the Dispatch Service, and the method that is called is known as the Dispatch Method.
- **Receive, Execute, and Send (ReceiveDispatchSend).** This is a request/response method. It receives an inbound request message, calls another service with the inbound message as input, and then sends the output of the called service as a response. To suppress the response, you can create an output property, on the dispatch service, of type EmptyResponse and set it to True.

  > **Note:** To receive a message and send a reply using the ReceiveDispatchSend method, you must use the `<Value>` process property in dispatched workflows to hold the message.

> **Note:** There are server components (called receivers) first on the inbound methods that run as Siebel Server tasks. When running an EAI receiver such as MQSeries Server or MSMQ Receiver (using the methods ReceiveDispatch or ReceiveDispatchSend), if the dispatch service has an error, then the receiver shuts down. Check the Status column on the Component Tasks for details about the cause of the error.

Using Named Subsystems for Transport Parameters

Named subsystems are groupings of defined enterprise parameters that are stored in the Siebel Gateway. You use named subsystems to specify methods and parameters for EAI Transports. Transport business services take two subsystem names as parameters, which you define using the Siebel Server Manager:

- **Transport Connection Subsystem** (ConnectionSubsystem)
- **Transport Data Handling Subsystem** (DataHandlingSubsystem)

Values for parameters in a named subsystem are common to every user of the subsystem across the enterprise. Subsystem names themselves are parameters for server components. You can logically group parameters into various subsystems.

For the two EAI Transport named subsystem parameters, ConnectionSubsystem and DataHandlingSubsystem, two parameters exist for the EAI receivers: ReceiverConnectionSubsystem and ReceiverDataHandlingSubsystem. The EAI Receiver looks up these parameters from the server component parameters and copies the corresponding properties (ConnectionSubsystem and DataHandlingSubsystem) to the input property set of the transport business service.

> **Note:** You must create named subsystems and specify the parameters for the subsystems. Then, you specify the named subsystems you created, for example, as business service user properties in a workflow or through scripting. You must specify named subsystem parameters by the values of their Alias fields in the Profile Parameters list.

The following subtopics are discussed in this topic:

- *Rules of Precedence for Parameter Specification*
- *Common EAI Transport Parameters*

Rules of Precedence for Parameter Specification

You can specify the two named subsystem parameters, ConnectionSubsystem and DataHandlingSubsystem, as either business service user properties or as run-time arguments. If you specify the parameters in both locations, then the business service user property takes precedence over the run-time arguments.

Note: For additional information about named subsystems, see *Siebel System Administration Guide* .

You specify every other parameter in one of the two named subsystems or as run-time arguments. Siebel EAI looks for the parameter in the ConnectionSubsystem or the DataHandlingSubsystem, depending on which parameter it is. If you specified the appropriate named subsystem, then Siebel EAI always looks for the parameter there.

If you do not specify the parameter in this named subsystem, even if you specified it as a run-time argument, then the run-time specification is ignored. Siebel EAI looks for the parameter in a run-time specification only if no appropriate named subsystem is specified.

Common EAI Transport Parameters

To configure the EAI Transports, you create named subsystems for data handling and connection parameters, as presented in the following table.

When You . . .	Use This Parameter . . .
Call any Business Service	DispatchService. This parameter must be used in conjunction with DispatchMethod.
Call any Business Service	DispatchMethod. This parameter must be used in conjunction with DispatchService.
Call the Dispatch Rule Set Business Service	DispatchRuleSet.
Call any Workflow	DispatchWorkflowProcess.

The data handling parameters are presented in the following table. These parameters are common to every transport method. After you create the named subsystems, you then specify these named subsystems as parameters in the service method argument or the business service user property.

Parameter Name	Description
CharSetConversion	CharSetConversion specifies if and how a character set conversion needs to occur before or after sending or receiving data from the external system. Legal values are None, UTF-8, and UTF-16.

Parameter Name	Description
	Default is None. Use the default value for this parameter for selfdescribing content such as XML and MIME. When used with a Receive method, CharSetConversion implies that the external data being read is in whatever charset specified by this setting and must be converted to String. Therefore, the output **<Value>** is a String whenever CharSetConversion is specified. If no CharSetConversion is specified, then the output **<Value>** is in binary and retains its original encoding. When used with a Send method, CharSetConversion defines the character set for the output data. The data in **<Value>** is converted to the character set specified by CharSetConversion. Depending on the value of this parameter, transport business services do implicit character set conversions, if necessary. Note that same CharSetConversion is assumed for requests and responses.
ConverterService	Default is EAI XML Converter. This is the name of the business service to use for serializing property sets to a buffer and unserializing buffers to property sets. This parameter receives arguments through business service user properties if the converter service can accept them. **Note:** You cannot use an arbitrary service as a converter service.
DispatchMethod	DispatchMethod parameter specifies the dispatch method. Specification of DispatchService is mutually exclusive with specification of a DispatchRuleSet or a DispatchWorkflowProcess. This parameter is only applicable for the ReceiveDispatch and ReceiveDispatchSend methods.
DispatchRuleSet	DispatchRuleSet specifies the name of the dispatch rule set for the Dispatcher Service. Specification of DispatchRuleSet is mutually exclusive with specification of DispatchWorkflowProcess or Dispatch Service. This parameter is only applicable for the ReceiveDispatch and ReceiveDispatchSend methods.
DispatchService	DispatchService specifies the dispatch service. Specification of DispatchService is mutually exclusive with specification of a DispatchRuleSet or DispatchWorkflowProcess. This parameter is only applicable for the ReceiveDispatch and ReceiveDispatchSend methods.
DispatchWorkflowProcess	DispatchWorkflowProcess specifies the name of the workflow to dispatch to. Specification of DispatchWorkflowProcess is mutually exclusive with specification of DispatchRuleSet or Dispatch Service. This parameter is only applicable for the ReceiveDispatch and ReceiveDispatchSend methods.
IgnoreCharSetConvErrors	Default is False. This parameter specifies whether character set conversion errors are ignored. If False, with any such errors, then the transport service propagates the error.
Impersonate	Default is False. This parameter indicates whether or not the receiver executes the incoming request using the default credentials of the receiver or those provided in the incoming XML document. If this parameter is set to True, then the receiver analyzes the incoming XML document (**<SiebelMessage>** element) for the eaiusername and eaipassword credential attributes. If these credentials are found, then the receiver attempts to relogin with the credential. If the Impersonate parameter is set to True and the credentials are not found or are not a valid Siebel username or password, then an error message is returned.
RollbackOnDispatchError	Default is True. This parameter indicates whether or not to roll back transport transaction if a Dispatch Method fails. This parameter is only available for the transactional transports MQSeries Server and MSMQ.
SiebelTransactions	Default is True. This parameter indicates whether or not to nest the Siebel transaction within the transport transaction. This parameter is only available for the transactional transports MQSeries Server

Parameter Name	Description
	and MSMQ. If this parameter is set to False, then the transaction support is turned off at the transport level. This setting means that if the transaction fails, then there is no rollback at the Siebel transaction level.

About Object Interfaces and EAI

Object Interfaces allow integration between the Siebel application and external applications. Object Interfaces can be called by eScripts and VB or used within a workflow. The workflow can use other business services and transports as needed.

Available object interface support includes Siebel Java Data Beans for integration with Java EE applications. For information, see *Integrating Siebel CRM with Java Applications*.

Database-Level Interfacing

In addition to transports and object interfaces, Siebel CRM provides Enterprise Integration Manager (EIM) for high-volume data exchange and batch loading. You use the set of interface tables that serve as intermediate tables between your external data source and the Siebel Database.

> **Note:** For more information about Siebel EIM and the interface tables, see *Siebel Enterprise Integration Manager Administration Guide* .

3 EAI MQSeries Server Transport

EAI MQSeries Server Transport

This chapter discusses the EAI MQSeries Server Transport business service. It includes the following topics:

- *About the EAI MQSeries Server Transport Business Service*
- *Using the SendReceive Method with MQSeries*
- *Dispatch Error Handling for the EAI MQSeries Server Transport*
- *Increasing the Maximum Message Length on IBM WebSphere MQ*
- *Using the EAI MQSeries Server Transport on AIX*
- *About EAI MQSeries Transport Re-Entrance*
- *About Message ID Tracking for an Inbound Message*
- *Invoking a Workflow Using MQSeries Server Receiver*

This chapter assumes that you understand the architecture and operation of IBM WebSphere MQ (formerly known as IBM MQSeries). For more information, consult the IBM WebSphere MQ documentation at: http://www.ibm.com/support.

About the EAI MQSeries Server Transport Business Service

The Siebel EAI MQSeries Server Transport provides a messaging solution to help you integrate data between Siebel CRM and external applications that can interface with IBM WebSphere MQ. The EAI MQSeries Server Transport business service transports messages to and from IBM WebSphere MQ queues. It uses the Message queuing API (MQI).

> **Note:** The EAI MQSeries Server Transport can connect only to IBM WebSphere MQ Server software. The IBM WebSphere MQ Server must be running on the same system as your Siebel Server. Before using the EAI MQSeries Server Transport, you must install and configure the IBM WebSphere MQ software. Contact your IBM sales representative for details.

The EAI MQSeries Server Transport supports the inbound and outbound methods described in *Outbound Methods for a Transport Business Service* and *Inbound Methods for a Transport Business Service*. This topic includes the following information:

- *About the MQPMO_SYNCPOINT Option*
- *EAI MQSeries Server Transport Parameters*
- *Exposing MQMD Headers as Properties*
- *EAI MQSeries Server Transport Named Subsystem*

About the MQPMO_SYNCPOINT Option

The EAI MQ Series Server Transport business service uses the MQPMO_SYNCPOINT option for sending messages to IBM WebSphere MQ using the IBM MQ API.

MQPMO_SYNCPOINT sends the message with syncpoint control. A *syncpoint* is a logical point in the execution of a program where changes made by the program can be saved. The message request operates within the unit of work: the message is not visible outside the unit of work until the unit of work is saved. If the unit of work is rolled backed, then the message is deleted. For more information about syncpoint options, consult the IBM WebSphere MQ documentation at:

http://www.ibm.com/support

EAI MQSeries Server Transport Parameters

In addition to supporting the common transport parameters presented in the second table in *Common EAI Transport Parameters*, the EAI MQSeries Server Transport uses the parameters shown in the following table. These can be specified as service method arguments, subsystem parameters, or user properties.

> **Note:** To send to a model queue, the model queue must have a definition type of PERMANENT and the following arguments must be supplied in the workflow: Model Queue, Physical Queue, Queue Manager, and Message Text.

Argument	Display Name	Description
MqAcknowledgements	Receive Acknowledgements	Default is False. This parameter specifies whether or not delivery and arrival acknowledgements are to be received.
MqAckPhysicalQueueName	Acknowledgement Physical Queue Name	If the MqAcknowledgements is set to True, then this parameter contains the name of the physical queue for acknowledgements to responses.
MqAckQueueManagerName	Acknowledgement Queue Manager Name	Defaults to MqQueueManagerName if unspecified. If MqAcknowledgements is set to True, then this parameter contains the name of the queue manager for acknowledgements to responses.
MqModelQueueName	Model Queue Name	Name of the MQSeries model queue.
MqPhysicalQueueName	Physical Queue Name	Name of the MQSeries physical queue. You can also create an alias queue which points to a target queue and use the alias queue name as the input argument physical queue name and send messages to the target queue.

Argument	Display Name	Description
		Note: Using an alias queue works. However, since the alias queue does not have a backout queue defined, the receiver cannot roll back to the backout queue.
MqQueueManagerName	Queue Manager Name	Name of the MQSeries queue manager. If this parameter is not specified, then the default Queue Manager Name, as specified in the MQSeries configuration, is used. The Response Queue Manager is the same as MqQueueManagerName.
MqRespModelQueueName	Response Model Queue Name	Name of model queue for response connection.
MqRespPhysicalQueueName	Response Physical Queue Name	Name of physical queue for response connection.
MqFormat	MQSeries Format	The format of the message from the Siebel application to the outbound queue.
MqSleepTime	Sleep Time	Default is 20000 milliseconds. The timeout interval on receive calls, in milliseconds.

In addition to the EAI MQSeries Server Transport, you can run the MQSeries Server Receiver, which is a server component that periodically checks the MQSeries queues you specify, for inbound messages.

Note: The persistence of the message is the same as the persistence of the queue itself.

Exposing MQMD Headers as Properties

In the inbound direction, that is, when a message is received from a queue, the EAI MQSeries Server Transport feature exposes the MQMD headers as properties of a property set. The supported headers are summarized in the last table, about MQMD Message Headers, in this topic

In the outbound direction, that is, when a message is placed on a queue, the EAI MQ Server Transport supports the headers shown in the following table to be set by the caller.

Header	Value
CodedCharSetId	MQCCSI_Q_MGR, MQCCSI_INHERIT, MQCCSI_EMBEDDED, or any positive Long.
Encoding	MQENC_NATIVE or any positive Long.
Expiry	Any positive Long.
MsgType	Any nonnegative Long.

Header	Value
Persistence	MQPER_PERSISTENT, MQPER_NOT_PERSISTENT, or MQPER_PERSISTENCE_AS_Q_DEF.
Priority	MQPRI_PRIORITY_AS_Q_DEF or any nonnegative Long.
Report	The only settable value is MQRO_NONE.
ReplyToQ	Name of the reply queue, for example, myQueue. ReplyToQ is set in the message header of an incoming MQ message by the sender application. This sets dynamically the queue for the response sent by Siebel CRM. ReplyToQ is valid for the ReceiveDispatchSend method. **Note:** If the Response queue is specified using a static configuration, then the ReplyToQ header of the incoming message is ignored. The static configuration overrides dynamic queuing. ReplyToQ can also be set by the Siebel application, as MQMD_S_In_ReplyToQ while using the Send method, to specify the response parameters.
ReplyToQMgr	Name of the reply queue manager, for example, myQueueManager. ReplyToQMgr is set in the message header of an incoming MQ message by the sender application. This sets dynamically the queue manager for the response sent by Siebel CRM. ReplyToQMgr is valid for the ReceiveDispatchSend method. **Note:** If the Response queue is specified using a static configuration, then the ReplyToQMgr header of the incoming message is ignored. The static configuration overrides dynamic queuing. ReplyToQMgr can also be set by the Siebel application, as MQMD_S_In_ReplyToQMgr while using the Send method, to specify the response parameters.

You can set a MQMD message header for the Siebel application by specifying it as a property in a property set on the outbound side. Whereas on the inbound side, the MQMD message header of the response is exposed to the user as a property on the output property set.

On the inbound side, you can have the supported MQMD message headers as part of the output property set without having to do extra steps to see these MQMD message headers.

On the outbound side, you can set the MQMD message headers using the EAI MQSeries Server Transport. To modify the MQMD message headers on the outbound side, the property value for FullMQMDControl must be set to TRUE.

During the sending business service step (EAI MQSeries Server Transport.Send) within the workflow, input arguments are added that can modify MQMD headers. Once the property FullMQMDControl is set to TRUE, you can modify other MQMD headers as the examples show in the following table.

Note: In workflows and scripts, you set and get MQMD parameters using their full names, for example, MQMD_S_In_Encoding.

Property	Type	Example Value
MQMD_S_In_CodedCharSetId	Literal	1208
MQMD_S_In_Encoding	Literal	MQENC_NATIVE
MQMD_S_In_Expiry	Literal	MQEI_UNLIMITED
MQMD_S_In_MsgType	Literal	TestMsgHeader
MQMD_S_In_Persistence	Literal	MQPER_PERSISTENT
MQMD_S_In_Priority	Literal	MQPRI_PRIORITY_AS_Q_DEF
MQMD_S_In_ReplyToQ	Literal	MyQueue
MQMD_S_In_ReplyToQMgr	Literal	MyQueueManager

Note: When using the Message Type header (MQMD_S_In_MsgType), make sure that the message type set makes sense in context. For example, if the Send method is used to send a message to MQSeries, then do not set the MsgType to MQMT_REQUEST. If the SendReceive method is used to send and request a response from MQSeries, then the MsgType of MQMT_REQUEST is applicable (this is automatically set by the Siebel application). In the previous table, MsgType is set to TestMsgHeader.

The following table summarizes the MQMD message headers that are exposed as properties in a property set.

Field	Data Type	Description	Input or Output Property?
AccountingToken	MQBYTE32	Accounting token	Output
ApplIdentityData	MQCHAR32	Application data relating to identity	Output
ApplOriginData	MQCHAR4	Application data relating to origin	Output
BackCount	MQLONG	Backout counter	Output
CodedCharSetId	MQLONG	Character set identifier of message	Input and Output
CorrelId	MQBYTE24	Correlation identifier	Output
Encoding	MQLONG	Numeric encoding of message data	Input and Output
Expiry	MQLONG	Message lifetime	Input and Output

Field	Data Type	Description	Input or Output Property?
Feedback	MQLONG	Feedback or reason code	Output
Format	MQCHAR8	Format name of message data	Input and Output
GroupId	MQBYTE24	Group Identifier	Output
MsgFlags	MQLONG	Flags that specify attributes of the message or control its processing	Output
MsgSeqNumber	MQLONG	Sequence number of logical message within group	Output
MsgType	MQLONG	Message Type	Input and Output
Offset	MQLONG	Offset of data in physical message from start of logical message	Output
OriginalLength	MQLONG	Length of original message	Output
Persistence	MQLONG	Message persistence	Input and Output
Priority	MQLONG	Message priority	Input and Output
PutApplName	MQCHAR28	Name of application that sent the message	Output
PutApplType	MQLONG	Type of application that sent the	Output
PutDate	MQCHAR8	Date when message was sent	Output
PutTime	MQCHAR8	Time when message was sent	Output
ReplyToQ	MQCHAR48	Name of reply queue	Input and Output
ReplyToQMgr	MQCHAR48	Name of reply queue manager	Input
Report	MQLONG	Options for report messages	Output
UserIdentifier	MQLONG	User identifier	Output
Version	MQLONG	Structure version number	Output

EAI MQSeries Server Transport Named Subsystem

The EAI MQSeries Transport can read parameters from a named subsystem. For the EAI MQSeries Server Transport, the named subsystem type is MqSeriesServerSubsys.

The following is an example of the EAI MQSeries Server Transport and the commands to create a named subsystem and start a receiver:

```
create named subsystem MyMqSrvrSubsys for subsystem MQSeriesServerSubsys with
MqPhysicalQueueName=Receiver, MqRespPhysicalQueueName=Sender,
MqQueueManagerName=myQueueMgr

create named subsystem SiebelEcho for subsystem EAITransportDataHandlingSubsys with
DispatchService="Workflow Utilities", DispatchMethod=ECHO

start task for comp MqSeriesSrvRcvr with
ReceiverConnectionSubsystem=MyMqSrvrSubsys,
ReceiverDataHandlingSubsystem=SiebelEcho, ReceiverMethodName=ReceiveDispatchSend
```

For a discussion of named subsystems for Siebel EAI, see . For more information about named subsystems, see *Siebel System Administration Guide* .

Using the SendReceive Method with MQSeries

The SendReceive method on the EAI MQSeries Server Transport sends a message and waits for a response from the target application on a response queue. This response message corresponds to the original message using the correlation ID in MQSeries.

> **Note:** It is the responsibility of the external application to set the correlation ID of the response to the Siebel Business Application to the message ID of the original message.

> **Note:** It is recommended that, when using the EAI MQSeries Server Transport business service with the SendReceive method, you check the TimedOut process property. If you send a message and the MQ transport times out waiting for a response, then the business service does not raise an error but the TimedOut value is true.

Dispatch Error Handling for the EAI MQSeries Server Transport

When using the ReceiveDispatch and ReceiveDispatchSend methods, certain MQSeries behavior might affect your messages.

> **Note:** The transaction does not end when the message is received from the queue because it waits for the entire dispatch process to either complete successfully for commit or fail for rollback.

If all of the following conditions are met, then the message is sent to the Backout Requeue Queue of the current queue manager:

- A dispatch error has occurred.
- The RollbackOnDispatchError property is set to TRUE.
- The message has been rolled back by a count exceeding the Backout Threshold of the queue.

Note: If the Backout Requeue Queue has not been specified for the Queue Manager, then the message is sent to the Dead Letter Queue of the current queue manager. If there is no specified Dead Letter Queue for the current queue manager, then the queue defaults to the SYSTEM.DEAD.LETTER.QUEUE.

Increasing the Maximum Message Length on IBM WebSphere MQ

The MaxMsgLength queue manager attribute in the IBM WebSphere MQ software defines the maximum length of a message that can be handled by a queue manager. The MaxMsgLength queue attribute is the maximum length of a message that can be handled by a queue.

The default maximum message length on IBM WebSphere MQ is 4 MB. If the message is too large for the queue, then MQRC_MSG_TOO_BIG_FOR_Q is returned. Similarly, if the message is too large for the queue manager, then MQRC_MSG_TOO_BIG_FOR_Q_MGR is returned.

If you are handling large messages, then you can change the MaxMsgLength queue manager and queue attributes independently. You can set the queue manager attribute value between 32768 bytes and 100 MB; you can set the queue attribute value between 0 and 100 MB.

After changing one or both of the MaxMsgLength attributes, restart your applications and channels to ensure that the changes take effect. For more information, consult the IBM WebSphere MQ documentation at:

http://www.ibm.com/support

Using the EAI MQSeries Server Transport on AIX

When you use the EAI MQSeries Server Transport on AIX, the shared memory segment required by the EAI MQSeries Server process can collide with the shared memory segment required by the queue manager. By default, the EAI MQSeries queue manager attempts to use shared memory segment number 8. The EAI MQSeries Server Transport does not rely on any specific number and uses whatever segment is given to the process by the AIX operating system.

However, if you are using the default configuration, then there is a possibility that the EAI MQSeries Server process gets segment number 8 from the operating system first, and as a result the queue manager cannot get its segment. In this case, the EAI MQSeries Server Transport service fails with an error code of 2059 because it cannot connect to the queue manager.

Fixing a Shared Memory Segment Conflict on AIX

You edit the mqs.ini file, found in the `/var/mqm` directory, to fix a shared memory segment conflict with the EAI MQSeries Server Transport on AIX.

To fix a shared memory segment conflict with the EAI MQSeries Server Transport on AIX

1. Shut down any queue manager connected to the EAI MQSeries Transport.
2. Edit the `/var/mqm/mqs.ini` file.

   In the QueueManager section for each queue manager of interest, add an additional line explicitly specifying the shared memory segment to use. For example:

   ```
   QueueManager:
   Name=myQueueManager
   Prefix=/var/mqm
   Directory=myQueueManager
   IPCCBaseAddress=12
   ```

3. Restart each queue manager.

Note: This example shows shared number 12 as the memory segment number. Valid values for the IPCCBaseAddress are 4, 5, 8, 9, 10, 11, and 12, although 8 has been found to be problematic. It is possible to get a shared memory segment conflict even with the number set to 12, if the operating system has allocated segment 12 to the EAI MQSeries Server process ahead of the queue manager. If this is the case, then a different segment number must be specified.

Configuring AIX to Run the Siebel Server with Less Memory

If the EAI MQSeries Server Transport business service on AIX continues to fail even after you have followed the previous procedures, then you can configure the AIX environment to run Siebel Server with less memory using the environment variable LDR_CNTRL. After you have finished, follow the procedures in the preceding topic. For more information about setting parameters for AIX, see *Siebel Performance Tuning Guide* .

To configure the AIX environment to run the Siebel Server with less memory

1. Shut down the Siebel Server.
2. In the shell that you use to bring up the Siebel Server, set the environment variable LDR_CNTRL. Using csh:

   ```
   setenv LDR_CNTRL MAXDATA=0x30000000
   ```

   Note: You can save the setting in the siebenv.sh or siebenv.csh.

3. Restart the Siebel Server with this environment variable.

About EAI MQSeries Transport Re-Entrance

The EAI MQSeries Server Receiver uses the EAI MQSeries Server Transport business service but cannot dispatch to a workflow that either uses this business service as one of its steps or dispatches directly to this business service.

While in-process re-entrance is not supported, you can indirectly invoke the EAI MQSeries Server Transport as one of the steps out of process by calling the Synchronous Server Requests business service.

About Message ID Tracking for an Inbound Message

You can keep track of Message IDs of inbound messages by creating a process property, MsgId, of type String, and then adding an output argument with the following configuration to the Send step of your process as shown in the following table.

Type	Output Argument
Output Argument	MQSeries Message Identifier

This captures the Message IDs for the Queue Manager assigned to the messages in the MsgId process property.

Invoking a Workflow Using MQSeries Server Receiver

Following are examples of commands to create named subsystems and start a MQSeries Server Receiver to invoke a workflow.

> **Note:** If there is either an exception step or an error process in your workflow, then the workflow assumes that the error step or the error process handles the error and the workflow does not send the error out. To capture the error, insert a stop step into your workflow. Note that by adding a stop step, the caller gets the generic workflow stop error and not the original error, but the original error is stored in the Error Code and Error Message process properties.

Command to Create an EAI Transport Data Handling Subsystem

The following command creates an EAI Transport Data Handling Subsystem:

```
create named subsystem MYDataSubSys for subsystem EAITransportDataHandlingSubsys
with DispatchWorkflowProcess="MQ Inbound Workflow"
```

Command to Create an EAI Transport Connection Subsystem

The following command creates an EAI Transport Connection Subsystem:

```
create named subsystem MYSubSys for subsystem mqseriesserversubsys with
MQQueueManagerName=QueueMgr, MQPhysicalQueueName=LocalQueue
```

Command to Start an MQSeries Server Receiver

The following command starts an MQSeries Server Receiver:

```
start task for component MqSeriesSrvRcvr with ReceiverConnectionSubsystem=MYSubSys,
ReceiverDataHandlingSubsystem=MYDataSubSys, ReceiverMethodName=ReceiveDispatch
```

When calling your workflow by the MQSeries Server Receiver, it is not necessary to include a step to pull the messages off the queue and pass them to the next step. The MQSeries Server Receiver automatically pulls the messages off the queue and passes them on if:

- You have created a new process property of data type String and a default string of <Value>. This process property stores the inbound message text picked up by the MqSeriesSrvRcvr.
- In your workflow step, where you handle the inbound messages from IBM WebSphere MQ, you insert an input argument of <Value> with type Process Property. The Property Name is the name of the process property that you created in the previous step.

Note: When you type in `<Value>`, the display name might change to Message Text or XML Document.

4 EAI MSMQ Transport

EAI MSMQ Transport

This chapter discusses Oracle's implementation of Microsoft Message Queuing (MSMQ) support with the EAI MSMQ Transport business service. It includes the following topics:

- *About Microsoft Message Queuing (MSMQ)*
- *Configuring the EAI MSMQ Transport Servers*
- *Configuring EAI MSMQ Transport for Various Send and Receive Scenarios*

About Microsoft Message Queuing (MSMQ)

Many large organizations are integrating various enterprise business applications into application networks. These networks allow applications to communicate with each other and share data, either automatically or by request. Technologies such as Microsoft Message Queuing (MSMQ) provide a messaging infrastructure for transporting data from one application to another, without the need for programming.

MSMQ allows applications running at different times to communicate across heterogeneous networks and systems, even when one or many of those systems are temporarily offline. Because applications send messages to queues and read messages from queues, the messages are always available and remain in the queue for as long as required. For example, the messages are still there when a system that was offline comes back online to retrieve them. Optionally, messages can be sent to a *dead letter* queue after a predetermined amount of time has passed to help make sure that only timely, relevant messages are received.

The following subtopics are described in this topic:

- *About the EAI MSMQ Transport*
- *Methods for Sending and Receiving Messages*
- *EAI MSMQ Transport Named Subsystems*

About the EAI MSMQ Transport

EAI MSMQ Transport is a Siebel business service that can be customized using Siebel Tools. With Siebel Tools, you define integration objects to be transported across the EAI MSMQ Transport business service. EAI MSMQ Transport is responsible for sending and receiving messages between a Siebel application and MSMQ queues. EAI MSMQ Transport allows you to:

- Send a message to an external system
- Send and receive synchronous messages between a Siebel application and an external system
- Receive a message and perform an action based on that message within a Siebel application
- Receive a message, perform an action within a Siebel application, and then send a synchronous response to the external system

Methods for Sending and Receiving Messages

EAI MSMQ Transport supports two transport modes: sending messages and receiving messages. The following methods are supported:

- Send
- Send and Receive Response (SendReceive)
- Receive
- Receive and Execute Service (ReceiveDispatch)
- Receive, Execute, Send Response (ReceiveDispatchSend)

Messages from a Siebel Application to an External System

You configure EAI MSMQ Transport using the Siebel Business Process Designer, where you specify various parameters, such as the queue where Siebel outbound messages are sent. You configure the message itself using the integration object feature within Siebel Tools. The message can be in any text or binary format, including XML. The default format is XML, where the integration object defines the XML Schema Definition (XSD) or the Document Type Definition (DTD) associated with the XML document.

You configure the EAI MSMQ Transport at design time to specify the MSMQ queue computer name and the queue name. You use the EAI MSMQ Transport along with the Siebel Business Process Designer Manager to model business processes for sending messages to the external system.

You can configure the EAI MSMQ Transport to send messages to external systems when an event occurs in a Siebel application. For example, suppose that one of your sales representatives enters a new opportunity for an account into a Siebel application. This information needs to be sent to other business units that might or might not be using a Siebel application. The message can be sent using EAI MSMQ Transport as the transport mechanism to inform these external systems.

EAI MSMQ Transport can also be used synchronously to send a message and receive a response back from an external system in a single session. For example, suppose that one of your customers calls your Call Center requesting information on an account. The sales agent initiates a process to send a request with the account name from a Siebel application to an external mainframe system using the EAI MSMQ Transport. In response, the sales agent then receives a list of transaction details for that customer displayed within a Siebel application form.

Messages to a Siebel Application from an External System

External applications can send messages to a Siebel application using EAI MSMQ Transport. These messages are received and routed by the EAI MSMQ Receiver in conjunction with the MSMQ system.

The EAI MSMQ Receiver is a Siebel Server component that waits for messages in a specified queue. If you select the Receive, Execute, Send Response method, then the EAI MSMQ Receiver waits for a response from a Siebel application and places the output into a response queue.

EAI MSMQ Transport Named Subsystems

The EAI MSMQ Transport can read parameters from a named subsystem. For this transport, the named subsystem type is MSMQSubsys.

For a discussion of named subsystems for Siebel EAI, see *EAI Transports and Interfaces Overview*. For more information about named subsystems, see *Siebel System Administration Guide* .

Configuring the EAI MSMQ Transport Servers

The instructions in this topic are for configuring the EAI MSMQ Transport servers. Use a two-server setup, configured as listed in the following topic. However, you can implement a single server or multiple servers.

MSMQ Primary Enterprise Controller

You configure the MSMQ Primary Enterprise Controller with the following components:

- Windows Server (for supported versions, see the Certifications tab on My Oracle Support)

  Note: For information about the Certifications application, see 1492194.1 (Article ID) on My Oracle Support.

- MSMQ Server
- As many MSMQ queues as needed
- Relevant ODBC driver
- Siebel Server
- Siebel Gateway
- Siebel Web Client
- Siebel Tools

Regional Enterprise Server and MSMQ Client

You configure the Regional Enterprise Server and MSMQ Client with the following components:

- Windows Server (for supported versions, see the Certifications tab on My Oracle Support)

  Note: For information about the Certifications application, see 1492194.1 (Article ID) on My Oracle Support.

- MSMQ Client
- As many MSMQ queues as needed
- The relevant ODBC driver
- Siebel Server
- Siebel Gateway
- Siebel Web Client

Note: The MSMQ Server can reside on either the MSMQ Primary Enterprise Controller or the Regional Enterprise Server. This functionality is independent of the underlying database. You can use any supported database, including IBM DB2, DB2 for z/OS, Oracle Database, and Microsoft SQL Server.

Configuring EAI MSMQ Transport for Various Send and Receive Scenarios

The EAI MSMQ Transport and the Siebel Business Process Designer Manager work in tandem to transfer data using MSMQ from one Siebel application to another Siebel application or to an external application. You can set up a workflow and choose attributes and values to define the transport for a particular send or receive scenario.

The following topics are described:

- *EAI MSMQ Transport Prerequisites*
- *EAI MSMQ Transport Parameters*
- *About Defining Integration Objects*
- *Sending Outbound Messages with EAI MSMQ Transport*
- *Receiving Inbound Messages with MSMQ Receiver*

EAI MSMQ Transport Prerequisites

You must set up both Microsoft SQL Server and MSMQ before configuring the EAI MSMQ Transport. In addition, the Siebel Business Process Designer Manager functionality must be available within Siebel Tools and Siebel Web Client.

EAI MSMQ Transport Parameters

The following information presents the parameters used for configuring the EAI MSMQ Transport.

Parameter	Description
EndOfData	Set to True to indicate end of data.
MsmqPhysicalQueueName	Name of the MSMQ Queue. Can be used for both sending and receiving messages.
MsmqQueueMachineName	Computer that owns the queue specified by the physical queue name.
MsmqRespQueueMachineName	Computer that owns the queue specified by MsmqRespQueueName.
MsmqRespQueueName	Name of the response queue.
MsmqSleepTime	Default is **20000** milliseconds. The amount of time that the EAI MSMQ Transport business service waits to receive a message.
TimedOut	If no message is received in seconds specified in SleepTime, then the TimedOut argument in the Output Property set is set to True.

Parameter	Description
IgnoreCorrelationId	Default is False. Set to ignore Correlation Id value on the inbound messages. If this flag is True, then the message is picked up from the queue regardless of the correlation Id on the message. This parameter is ignored for the SendReceive Method because Correlation Id is required to match the response with the original message.
LargeMessageSupport	Default is True. Set to enable or disable large-message (messages over 4 MB) support. Set IgnoreCorrelationId to False for Large Message Support.

About Defining Integration Objects

Before you use the EAI MSMQ transport, you must define integration objects for use with the transport. The various methods explained in the following pages assume that this integration object has already been defined. You define your Siebel messages as integration objects using Siebel Tools. These messages correspond to the information that you want to exchange between the Siebel application and an external application. An example of an integration object would be an order, an account, a quote, or a contact.

After you have created an integration object, you can then send the message corresponding to this integration object through the EAI MSMQ Transport, either as part of a workflow or as a custom business service.

For information about creating integration objects, see *Integration Platform Technologies: Siebel Enterprise Application Integration* .

Sending Outbound Messages with EAI MSMQ Transport

With the Siebel application as the sender (outbound messaging), you design a workflow that queries for a record (such as a contact) and then converts that record to an XML document. The XML document is then sent to an MSMQ queue.

Because MSMQ imposes a limit of four megabytes on the size of the messages it can handle, the EAI MSMQ Transport separates outbound Siebel messages larger than four megabytes into smaller messages acceptable to MSMQ. The message is then reassembled after it has left MSMQ and arrived at your partner's system.

There are two methods for sending messages from a Siebel application to MSMQ:

- Send
- Send and Receive Response (SendReceive)

Sending Messages with EAI MSMQ Transport

The following procedure describes how to set up your system to send a message to an external system using the EAI MSMQ Transport.

To send messages from a Siebel application to MSMQ

1. Access the Windows Computer Management tool by choosing the Start menu, Programs, Administrative Tools, and then Computer Management.

2. Set up an MSMQ queue to receive messages from the Siebel application. Give the queue an easily identified name, such as fromsiebel. The MSMQ queue you create (for example, fromsiebel) will appear in the list of queues.
3. Set the queue to be Transactional.

> **Note:** This flag allows Siebel CRM to group a number of Send or Receive messages. This is critical when large data sets are being used because it allows a commit or a rollback to be executed without failure.

4. Set up a workflow in Siebel Tools to send a message to MSMQ. The workflow should contain the following steps, as show in the following image:

   a. Siebel Adapter (Query)
   b. XML Converter (PropSet ToXML)
   c. EAI MSMQ Transport (Send)



> **Note:** For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide* .

5. Create the following process properties:

Name	Data Type	In/Out	Value
Employee Message	Hierarchy	In/Out	Not applicable
Employee XML	Binary	In/Out	Not applicable
Error Code	String	In/Out	Not applicable
Error Message	String	In/Out	Not applicable
Object Id	String	In/Out	Row Id of an Employee record
Siebel Operation Object Id	String	In/Out	Not applicable

6. Set up the first step of the workflow, after Start, to use the EAI Siebel Adapter business service with the Query method to query the information from the Siebel database using the following input and output arguments:

Input Argument	Type	Value	Property Name
OutputIntObjectName	Literal	Sample Employee	Not applicable
PrimaryRowId	Process Property	Not applicable	Object Id

Input Argument	Type	Value	Property Name

Property Name	Type	Output Argument
Employee Message	Output Argument	SiebelMessage

7. Set up the second step to use the EAI XML Converter business service with the PropSetToXML method to convert the data extracted from the Siebel Database to XML format using the following input and output arguments:

Input Argument	Type	Property Name
SiebelMessage	Process Property	Employee Message

Property Name	Type	Output Argument
Employee XML	Output Argument	<Value>

8. Set up the third step to use EAI MSMQ Transport with the Send method to send the information to the external system, using the following input arguments:

Input Argument	Type	Value	Property Name
<Value>	Process Property	Not applicable	Employee XML
MsmqPhysicalQueueName	Literal	private$\FromSiebel	Not applicable
MsmqQueueMachineName	Literal	SiebelServer Computer name where the Siebel MSMQ Transport is running.	Not applicable

9. Save the workflow and run it from the Workflow Simulator.

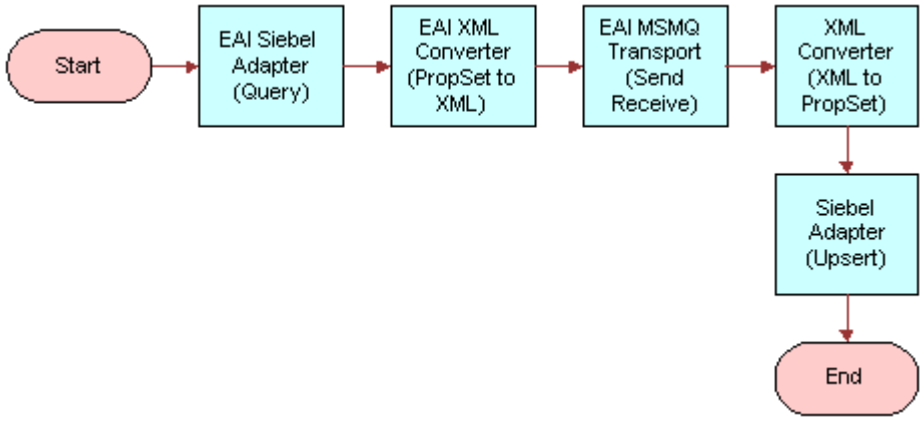
   Confirm that a message was sent to the queue using the MSMQ Explorer. In this example, if the simulation is successful, then a message is in the fromSiebel queue and contains an XML file with employee information.

Sending and Receiving Messages with EAI MSMQ Transport

The following procedure describes how to set up your system to send a message to an external system using the EAI MSMQ Transport and receive a synchronous message back from the external system by the EAI MSMQ Transport.

To send a literal to MSMQ and receive a response

1. Access the Windows Computer Management tool by choosing the Start menu, Programs, Administrative Tools, and then Computer Management.
2. Set up an MSMQ queue to receive messages from the Siebel application, and give the queue an easily identified name, such as fromsiebel.
3. Set up another queue to send messages to the Siebel application, and give the queue an easily identified name, such as tosiebel.
4. Set up a workflow in Siebel Tools to send a message out and receive a message back in response using EAI MSMQ Transport. The workflow should contain the following steps, as shown in the following image:
   a. EAI Siebel Adapter (Query)
   b. EAI XML Converter (PropSet to XML)
   c. EAI MSMQ Transport (Send Receive)
   d. XML Converter (XML to PropSet)
   e. Siebel Adapter (Upsert)



> **Note:** For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide* .

5. Create the following process properties:

Name	Data Type	In/Out
Test Message	Hierarchy	In/Out
Test XML	Binary	In/Out
Error Code	String	In/Out
Error Message	String	In/Out

Name	Data Type	In/Out
Object Id	String	In/Out
Siebel Operation Object Id	String	In/Out

6. Set up the first step of the workflow after Start to use EAI Siebel Adapter with the Query method to query the information from the Siebel Database using the following input and output arguments:

Input Argument	Type	Value	Property Name	Property Data Type
OutputIntObjectName	Literal	Sample Employee	Not applicable	Not applicable
PrimaryRowId	Process Property	Not applicable	Object Id	String

Property Name	Type	Output Argument
Test Message	Output Argument	SiebelMessage

7. Set up the second step to use the EAI XML Converter business service with the IntObjHierToXMLDoc method to convert the data extracted from the Siebel Database to XML format, using the following input and output arguments:

Input Argument	Type	Property Name	Property Data Type
SiebelMessage	Process Property	Test Message	Hierarchy

Property Name	Type	Output Argument
Test XML	Output Argument	<Value>

8. Set up the third step of the workflow, after Start, to use the EAI MSMQ Transport business service with the SendReceive method to receive the incoming XML message, using the following input and output arguments:

Input Argument	Type	Value	Property Name	Property Data Type
<Value>	Process Property	Not applicable	Test XML	Binary

Input Argument	Type	Value	Property Name	Property Data Type
MsmqPhysicalQueueName	Literal	fromsiebel	Not applicable	Not applicable
MsmqQueueMachineName	Literal	SiebelServer1 Computer name where the Siebel MSMQ Transport is running.	Not applicable	Not applicable
MsmqRespQueueMachineName	Literal	SiebelServer2	Not applicable	Not applicable
MsmqRespQueueName	Literal	tosiebel	Not applicable	Not applicable

Property Name	Type	Output Argument
Test XML	Output Argument	<Value>

9. Set up the fourth step to use the EAI XML Converter business service with the XMLDocToIntObjHier method to convert the XML message to a Siebel Message using the following input and output arguments:

Input Argument	Type	Property Name	Property Data Type
<Value>	Process Property	Test XML	Binary

Property Name	Type	Output Argument
Test Message	Output Argument	SiebelMessage

10. Set up the last step to use the EAI Siebel Adapter with the Upsert method to update the Siebel Database, using the following input argument:

Input Argument	Type	Property Name	Property Data Type
SiebelMessage	Process Property	Test Message	Hierarchy

11. Save the workflow and run a test using the Workflow Simulator.

The output property set must have a message in the Value field. Additionally, the EndOfData argument in the property set must be set to True.

> **Note:** To test this scenario adequately, you must have a partner application that can accept the message and return a response. The correlation ID of the response message must be set to the message ID of the message originally sent by the Siebel application.

Receiving Inbound Messages with MSMQ Receiver

With the Siebel application as the receiver (inbound messaging), you design a workflow that reads from the queue and converts the XML messages found there into Siebel message format. Then, the EAI Siebel Adapter updates the appropriate tables within the Siebel Database.

> **Note:** MSMQ Receiver must run on the same computer where you have defined the receiving queue.

There are two methods for receiving messages for a Siebel application:

- Receive and Execute Service (ReceiveDispatch)
- Receive, Execute, Send Response (ReceiveDispatchSend)

Receiving and Dispatching MSMQ Messages with MSMQ Receiver

The following procedure describes how to set up your system to receive an inbound message from MSMQ by MSMQ Receiver, then perform an action based on that message within the Siebel application.

To receive and dispatch messages using the EAI MSMQ Transport (MSMQ Receiver)

1. Access the Windows Computer Management tool by choosing the Start menu, Programs, Administrative Tools, and then Computer Management.
2. Set up a queue to send messages to the Siebel application:
   a. Name the queue an easily identified name, such as toSiebel.
   b. Create a message in the queue.

   > **Note:** To test this procedure adequately, you must have a partner application that can send a valid message for the Siebel application to the queue.

3. Create a named subsystem for the MSMQ Receiver using the following lines:

```
create named subsystem MyMSMQSubsys for subsystem MSMQSubsys with
MsmqQueueMachineName=SiebelServer1, MsmqPhysicalQueueName=fromSiebel,
MsmqRespQueueMachineName=SiebelServer2, MsmqRespQueueName=toSiebel

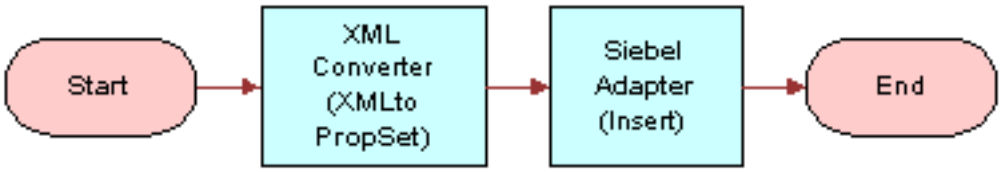
create named subsystem SiebelEcho for subsystem EAITransportDataHandlingSubsys
with DispatchService="Workflow Process Manager", DispatchMethod=RunProcess,
DispatchWorkflowProcess="MyMSMQWorkflow"

start task for comp MSMQRcvr with ReceiverConnectionSubsystem=MyMSMQSubsys,
ReceiverDataHandlingSubsystem=SiebelEcho, ReceiverMethodName=ReceiveDispatch
```

   > **Note:** The DispatchService and DispatchMethod parameters are optional.

4. Set up a workflow in Siebel Tools to receive and dispatch a message from MSMQ. The workflow should contain the following steps, as shown in the following image:

   a. XML Converter (XMLto PropSet)
   b. Siebel Adapter (Insert)



> **Note:** For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide* .

5. Create the following process properties:

Name	Data Type	In/Out
Test Message	Hierarchy	In/Out
Test XML	Binary	In/Out
Error Code	String	In/Out
Error Message	String	In/Out
Object Id	String	In/Out
Siebel Operation Object Id	String	In/Out

6. Set up the first step of the workflow after Start to use the EAI XML Converter business service with the XMLDocToIntObjHier method to convert the XML message to a Siebel Message using the following input and output arguments:

Input Argument	Type	Property Name	Property Data Type
<Value>	Process Property	Test XML	Binary

Property Name	Type	Output Argument
Test Message	Output Argument	SiebelMessage

7. Set up the second step to use the EAI Siebel Adapter with the Upsert method to update the Siebel Database, using the following input arguments:

Input Argument	Type	Property Name	Property Data Type
SiebelMessage	Process Property	Test Message	Hierarchy

> **Note:** In order to test this scenario adequately, you must have a partner application that can send a valid message for the Siebel application to the queue.

8. Save the workflow.

Receiving, Dispatching, and Sending MSMQ Messages with MSMQ Receiver

The following procedure shows you how to set up your system to receive an inbound message from MSMQ by MSMQ Receiver, perform an action within a Siebel application based on that message, and then send a synchronous response back to the external system.

To receive, dispatch, and send messages using the EAI MSMQ Transport (MSMQ Receiver)

1. Access the Windows Computer Management tool by choosing the Start menu, Programs, Administrative Tools, and then Computer Management.
2. Set up an MSMQ queue to receive messages from the Siebel application.

   Give the queue an easily identified name, such as fromSiebel.
3. Set up another queue to send messages to the Siebel application.
   a. Name the queue an easily identified name, such as toSiebel.
   b. Create a message in the queue.

> **Note:** To test this procedure adequately, you must have a partner application that can send a valid message for the Siebel application to the queue.

4. Create a named subsystem for the MSMQ Receiver using the following lines:

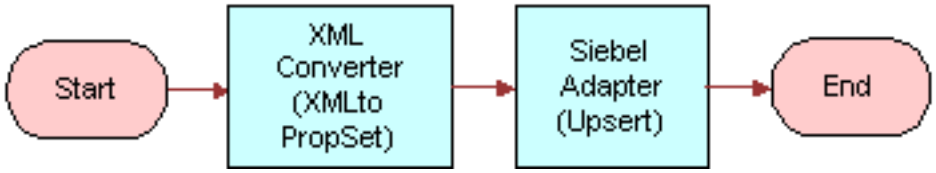
```
create named subsystem MyMSMQSubsys for subsystem MSMQSubsys with
MsmqQueueMachineName=SiebelServer1, MsmqPhysicalQueueName=fromSiebel,
MsmqRespQueueMachineName=SiebelServer2, MsmqRespQueueName=toSiebel

create named subsystem SiebelEcho for subsystem EAITransportDataHandlingSubsys
with DispatchService="Workflow Process Manager", DispatchMethod=RunProcess,
DispatchWorkflowProcess="MyMSMQWorkflow"

start task for comp MSMQRcvr with ReceiverConnectionSubsystem=MyMSMQSubsys,
ReceiverDataHandlingSubsystem=SiebelEcho,
ReceiverMethodName=ReceiveDispatchSend
```

> **Note:** The DispatchService and DispatchMethod parameters are optional.

5. Set up a workflow in Siebel Tools to receive and dispatch a message from MSMQ. The workflow should contain the following steps, as shown in the following image:
   a. XML Converter (XMLto PropSet)
   b. Siebel Adapter (Upsert)



Note: For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide* .

6. Create the following process properties:

Name	Data Type	In/Out	Value
Test Message	Hierarchy	In/Out	Not applicable
Test XML	Binary	In/Out	Test Message from Siebel Server
Error Code	String	In/Out	Not applicable
Error Message	String	In/Out	Not applicable
Object Id	String	In/Out	Not applicable
Siebel Operation Object Id	String	In/Out	Not applicable

7. Set up the first step of the workflow after Start to use the EAI XML Converter business service with the XMLDocToIntObjHier method to convert the XML message to a Siebel Message using the following input and output arguments:

Input Argument	Type	Property Name	Property Data Type
<Value>	Process Property	Test XML	Binary

Property Name	Type	Output Argument
Test Message	Output Argument	SiebelMessage

8. Set up the second step to use the EAI Siebel Adapter with the Upsert method to update the Siebel Database, using the following input arguments:

Input Argument	Type	Property Name	Property Data Type
SiebelMessage	Process Property	Test Message	Hierarchy

> **Note:** To test this scenario adequately, you must have a partner application that can send a valid message for the Siebel application to the queue.

9. Save the workflow.

   After running the workflow, confirm that the message is removed from the queue using the MSMQ Explorer. In this example, the Siebel Database is updated with the message in the fromSiebel queue. Also, a response message is in the queue specified by the MSMQRespQueueName and MSMQRespQueueMachineName arguments.

5 EAI Java Business Service

EAI Java Business Service

This chapter discusses the EAI Java Business Service. It includes the following topics:

- *About the EAI Java Business Service*
- *Requirements for Implementing a Java Business Service*
- *Creating a Java Business Service*
- *About the Lifecycle of a 32-bit Java Business Service*
- *Example of a Java Business Service*
- *Restrictions for Implementing a Java Business Service*
- *Troubleshooting the Java Business Service*

About the EAI Java Business Service

The EAI Java Business Service (JBS) is a service framework that allows custom business services to be implemented in Java and run from a Siebel application. The framework consists of the following:

- A template business service, EAI Java Business Service, which is defined in the repository.
- An abstract Java class, com.siebel.eai.SiebelBusinessService, that defines the interface of the Java class that implements the business service.

The EAI Java Business Service works in two different ways:

- 32-bit JVM. The EAI Java Business Service works by creating a 32-bit Java Virtual Machine (JVM) in-process with the Siebel application and invoking Java implementations using Java Native Interface (JNI). Each Siebel process (component) has at most one JVM. JVMs are not shared across components.
- 64-bit JVM. The EAI Java Business Service works by creating a 64-bit Java Virtual Machine (JVM) that runs in a separate process from Siebel application and invokes Java implementations using HTTP.

Requirements for Implementing a Java Business Service

To implement a Java business service, the following software must be installed and properly configured on each Siebel Server or Siebel Mobile and Developer Web Clients:

- A Java Runtime Environment (JRE)
- All necessary Java code
- A configured named subsystem of type:
  - JVMSubSys for a 32-bit JRE or

- JavaContainerSubSys for a 64-bit JRE

The named 32-bit subsystem supplies the following parameters to the JBS: DLL, CLASSPATH, and VMOPTIONS. These parameters are described as follows.

- **DLL.** The complete path of the JRE library, as shown in the following table.
- **CLASSPATH.** The classpath used by the JVM.

  The classpath must include the following Siebel JAR files as well as all Java code implementing the desired business service.

  The required Siebel JAR files are:

  - Siebel.jar
  - SiebelJI_lang.jar (lang corresponds to the default language for your installation).
- **VMOPTIONS.** Java Virtual Machine options. On all platforms, except AIX, it is recommended that the option -Xusealtsigs be used to make sure that the signal handlers used by the Siebel Server do not conflict with those of the JVM.

  > **Note:** The -Xusealtsigs option is mandatory for use on the Oracle Solaris platform. The JVM options do not load successfully into the Application Object Manager without this option.

Operating System	JRE Library	Typical Location on Server and Environment Variable Setting
AIX	libjvm.so	**`/usr/java/jre/lib/ppc/j9vm`** You must include both **`/usr/java/jre/lib/ppc/`** and **`/usr/java/jre/lib/ppc/j9vm`** in the LIBPATH variable. For example: siebenv.csh **`setenv LIBPATH=/siebel/siebsrvr/lib:/siebel/siebsrvr/mw/lib:/siebel/siebsrvr/SYBSsa90/lib:/usr/lib:/siebel/siebsrvr/lib:/oracle_client/app/oracle/OraHome_1/lib32:/oracle_client/app/oracle/OraHome_1/lib:/usr/java/jre/lib/ppc/:/usr/java/jre/lib/ppc/j9vm`** For example: siebenv.sh **`LIBPATH=/siebel/siebsrvr/lib:/siebel/siebsrvr/mw/lib:/siebel/siebsrvr/SYBSsa90/lib:/usr/lib:/siebel/siebsrvr/lib:/oracle_client/app/oracle/OraHome_1/lib32:/oracle_client/app/oracle/OraHome_1/lib:/usr/java/jre/lib/ppc/:/usr/java/jre/lib/ppc/j9vm`**
HP-UX	libjvm.sl	**`/opt/java/jre/lib/PA_RISC2.0/server`** Set the environment variable SHLIB_PATH to include the JVM's **`jre`** and **`server`** directories. Set LD_PRELOAD in the siebmtshw file located in **`/siebsrvr/bin`**. For example: **`setenv SHLIB_PATH=${SHLIB_PATH}:/opt/java/jre/lib/PA_RISC2.0; export SHLIB_PATH`** In siebmtshw:

Operating System	JRE Library	Typical Location on Server and Environment Variable Setting
		`LD_PRELOAD=/opt/java/jre/lib/PA_RISC2.0/server/libjvm.sl` `export LD_PRELOAD`

Linux	libjvm.so	`/usr/java/jdk/jre/lib/i386/server`
		Set the environment variable LD_LIBRARY_PATH to include the JVM's **server** directory. For example:
		`setenv LD_LIBRARY_PATH=/usr/java/jdk/jre/lib/i386/server:`
		`/usr/java/jdk/jre/lib/i386`

Windows	jvm.dll	JDK installation directory. If using Java 7, then set the environment variable Path to include the JRE library. This is not necessary for Java 5 or 6. For example: `<JRE_HOME>\jre7\bin\client\jvm.dll`

Oracle Solaris	libjvm.so	`/usr/jdk/instances/jdk/jre/lib/sparc/server`
		Set the environment variable LD_LIBRARY_PATH to include the JVM's **server** directory. Add `/platform/SUNW,Sun-Fire-V440/lib` to LD_LIBRARY_PATH.
		For example: siebenv.csh:
		`setenv LD_LIBRARY_PATH=/usr/jdk/instances/jdk/jre/lib/`
		`sparc/server:/platform/SUNW,Sun-Fire-V440/`
		`lib:{LD_LIBRARY_PATH}`
		For example: siebenv.sh
		`LD_LIBRARY_PATH=/usr/jdk/instances/jdk/jre/lib/sparc/`
		`server:/platform/SUNW,Sun-Fire-V440/`
		`lib:{LD_LIBRARY_PATH};export LD_LIBRARY_PATH`

The named 64-bit subsystem supplies the following parameters to the JBS: CONTAINERURL, CLASSPATH, and OPTIONS. These parameters are described as follows.

- **CONTAINERURL:** The URL to the Java Web Container server for all the JBS requests.
- **CLASSPATH:** The classpath used by the JVM.

  The classpath must include the location of the jndi.properties file.

  Ensure that the file contains the file name of the jndi.properties file.

  With Java 8, the required JAR files for the execution of JMS must reside on the Java Web Container server. The Siebel.jar and SiebelJI_enu,jar are packaged within the war file. All other JMS Provider Jars depending on the usage of customer must be placed in the lib directory of the Apache Tomcat server.
- **OPTIONS:** In the Java 64-bit subsystem, OPTIONS is not used. Therefore, OPTIONS must be set using the CATALINA_OPTS option in the setenv.bat or setenv.sh file of the javacontainer based on the operating system. An example is as follows:

  Windows

  `javacontainerX\bin\setenv.bat`

```
set CATALINA_OPTS=-Djava.compiler=NONE
```

non-Windows

```
File: javacontainerX/bin/setenv.sh
CATALINA_OPTS="-Djava.compiler=NONE"
```

The following topics are also discussed here:

- *Creating a 32-bit Java Subsystem by Using the Siebel Server Manager*
- *Creating a 64-bit Java Subsystem by Using the Siebel Server Manager*
- *Creating a 32-bit Java Subsystem by Using the Siebel Web Client*
- *Creating a 64-bit Java Subsystem by Using the Siebel Web Client*
- *Creating a 32-bit Java Subsystem by Using the Siebel Dedicated Client*
- *About Platform-Specific Configurations for the JVM*

Creating a 32-bit Java Subsystem by Using the Siebel Server Manager

The following example shows how to create a named 32-bit Java subsystem using the Siebel Server Manager:

```
create named subsystem JAVA for subsystem JVMSubSys with
DLL="D:\jdk\jre\bin\server\jvm.dll",
CLASSPATH="c:\cp\Siebel.jar;c:\cp\SiebelJI_enu.jar;c:\cp\myJARs.jar;.",
VMOPTIONS="-Xrs -Djava.compiler=NONE"
```

Note: On Oracle Solaris, the create statement might be truncated. To avoid this, you can set CLASSPATH in the create statement and DLL and VMOPTIONS in the Siebel application.

Alternatively, the parameters to the Java Business Service can be specified in the application configuration (CFG) file instead of a named subsystem. This applies only to the Siebel Mobile and Developer Web Clients, and not the Siebel Server.

```
[JAVA]
DLL = D:\jdk\jre\bin\server\jvm.dll
CLASSPATH = c:\cp\Siebel.jar;c:\cp\SiebelJI_enu.jar;c:\cp\myJARs.jar;.
VMOPTIONS = -Xrs -Djava.compiler=NONE
```

Creating a 64-bit Java Subsystem by Using the Siebel Server Manager

The following example shows how to create a named 64-bit Java subsystem using the Siebel Server Manager:

```
create named subsystem JAVA64 for subsystem JavaContainerSubSys
change param CONTAINERURL=http://localhost:<Config Agent HTTP Port>/siebel/jbs for
named subsystem JAVA64
change param CLASSPATH=<JNDI file path> for named subsystem JAVA64
change param JVMSubsys= JAVA64 for comp sccobjmgr_enu
```

Based on your usage, place all other JMS Provider jars in the lib directory of the Apache Tomcat server.

Note: In Innovation Pack 2017 Siebel Server, the JAVA64 subsystem is preconfigured with the required CONTAINERURL, which eliminates the need for you to create the JAVA64 subsystem manually. However, you can modify parameters such as CLASSPATH-based JNDI file path.

Creating a 32-bit Java Subsystem by Using the Siebel Web Client

The following is an alternative procedure for creating a Java subsystem by using the Siebel Web Client.

To create a Java subsystem by using the Siebel Web Client

1. In the Siebel client, navigate to the Administration - Server Configuration screen, Enterprises view.
2. In the first list applet, select the Enterprise Server that you want to configure.
3. In the middle applet, click the Profile Configuration tab.
4. Click New to create a new component profile and set the following parameters:

Name	Value
Profile	JAVA
Alias	JAVA
Subsystem Type	JVMSubsys

5. In the Profile Parameters list applet (the last applet), set the following values:
   a. Set the Value of the JVM Classpath parameter to one of the following:
      - The location of the jndi.properties file (if using the JMS Transport).
      - The JMS provider JAR files (if using the JMS Transport).
      - The Siebel.jar and SiebelJI_enu.jar files. These files can be installed by using either Siebel Tools or the Siebel Server. An example of these files for Microsoft Windows follows:

        ```
        c:\Oracle\Middleware\wlserver_10.3\server\lib\weblogic.jar;c:\siebel\jndi;
        c:\siebel\siebsrvr\CLASSES\Siebel.jar;
        c:\siebel\siebsrvr\classes\SiebelJI_enu.jar
        ```

   b. Set the Value of the JVM DLL Name parameter to the path where you have the jvm.dll file installed. For example:

      ```
      D:\jdk\jre\bin\server\jvm.dll
      ```

   c. Set the Value of the JVM Options record to any JVM-specific options that you would like to enable, for example:

      ```
      -Djava.compiler=NONE
      ```

Creating a 64-bit Java Subsystem by Using the Siebel Web Client

The following is an alternative procedure for creating a 64-bit Java subsystem by using the Siebel Web Client.

To create a 64-bit Java subsystem by using the Siebel Web Client

1. In the Siebel client, navigate to the Administration - Server Configuration screen, Enterprises view.
2. In the first list applet, select the Enterprise Server that you want to configure.
3. In the middle applet, click the Profile Configuration tab.
4. Click New to create a new component profile and set the following parameters:

Name	Value
Profile	JAVA64
Alias	JAVA64
Subsystem Type	JavaContainerSubSys

5. In the Profile Parameters list applet (the last applet), set the following values:
   a. Set the Value of the JVM Classpath parameter to the following:

      The location of the jndi.properties file (if using the JMS Transport).
   b. Set CONTAINERURL to point to the java container. For example:

      ```
      CONTAINERURL=http://localhost:<Config Agent HTTP Port>/siebel/jbs for named
      subsystem JAVA64
      ```

   c. Based on your usage, place all other JMS Provider jars in the lib directory of the Apache Tomcat server.

Note: In Innovation Pack 2017 Siebel Server, the JAVA64 subsystem is preconfigured with the required CONTAINERURL, which eliminates the need for you to create the JAVA64 subsystem manually. However, you can modify parameters such as CLASSPATH-based JNDI file path.

Creating a 32-bit Java Subsystem by Using the Siebel Dedicated Client

For dedicated client, define subsystem in the .cfg file with name JAVA.

Define the 32-bit subsystem as follows:

```
[JAVA]

DLL = "<jre Install Dir>\bin\server\jvm.dll#?
```

```
CLASSPATH = "c:\cp\Siebel.jar;c:\cp\SiebelJI_enu.jar;c:\cp\myJARs.jar#?;

VMOPTIONS = "-Xrs -Djava.compiler=NONE#?
```

About Platform-Specific Configurations for the JVM

Depending on the platform, it is necessary to set certain environment variables to load the JVM properly:

- **AIX.** Make sure that you have the environment variable LIBPATH set to include the JVM's shared libraries, `/usr/java/jre/lib/ppc/` and `/usr/java/jre/lib/ppc/j9vm`. For example:

  ```
  setenv LIBPATH=/siebel/siebsrvr/lib:/siebel/siebsrvr/mw/lib:/siebel/siebsrvr/
  SYBSsa90/lib:/usr/lib:/siebel/siebsrvr/lib:/oracle_client/app/oracle/OraHome_1/
  lib32:/oracle_client/app/oracle/OraHome_1/lib:/usr/java/jre/lib/ppc/:/usr/java/
  jre/lib/ppc/j9vm
  ```

  For more information about setting the LIBPATH environment variable, see the documentation for IBM SDK and Java Runtime Environments (JREs) at:

  http://www.ibm.com/support

- **HP-UX.** Make sure that you have the environment variable SHLIB_PATH set to include the JVM's `jre` and `server` directories. For example:

  ```
  setenv SHLIB_PATH /opt/java/jre/lib/PA_RISC2.0:/opt/java/jre/lib/PA_RISC2.0/
  server:${SHLIB_PATH}
  ```

  Set the variable LD_PRELOAD to the full path of the Java library.

- **Oracle Solaris, Windows.** No additional settings are needed.

When a Java business service is invoked on UNIX from a server component (for example, the JMS Receiver; see *EAI JMS Transport* for more information), the necessary settings must be done in the script that creates the component.

For the receiver, the script is `siebshw`; for the Application Object Managers, it is `siebmtshw`. These scripts are present in the `bin` directory where the Siebel Server is installed.

Creating a Java Business Service

The following topics describe how to create a Java business service:

- *Defining a Business Service in Java*
- *About Implementing a Business Service in Java*
- *About Exception Handling for the Java Business Service*

Defining a Business Service in Java

You define a business service in Java by:

- Defining a new business service in the repository using Siebel Tools.

- Specifying the necessary Java classes.

To define and specify a new Java business service in Siebel Tools

1. Copy the EAI Java Business Service (using the Copy Record command in Siebel Tools) and rename the copy.

   Note: Checking the Cache column when you are creating the new Java business service causes the same Java object to be reused by subsequent invocations within the same session. See *About the Lifecycle of a 32-bit Java Business Service*.

2. Add a business service user property named @class, whose value is the fully qualified name of the Java class (for example, com.example.siebelBusinessService.ImportCustomer).

About Implementing a Business Service in Java

Once the Java business service has been defined in Siebel Tools, the Java class must be implemented. The Java class implementing the business service must extend com.siebel.eai.SiebelBusinessService.

SiebelBusinessService is an abstract Java class found in Siebel.jar. It declares three methods:

- **destroy.** This method is called when the Java object is released by the Siebel application. It has a default empty implementation and can be overridden for the purpose of performing any cleanup.
- **invokeMethod.** This method contains a default implementation that calls doInvokeMethod and catches any exceptions that are thrown by it. It does not declare any exceptions. It is invoked by means of JNI in the Siebel application's native process. This method is not intended to be overridden.
- **doInvokeMethod.** This method must be implemented by the subclass that implements the business service. It takes as arguments the methodName, input property set, and output property set. The property sets are instances of com.siebel.data.SiebelPropertySet. This method throws SiebelBusinessServiceException.

About Exception Handling for the Java Business Service

Errors are handled by throwing a com.siebel.eai.SiebelBusinessServiceException class. The constructor for this class takes two String arguments, an error code and an error message. The error code can be used for programmatic handling in Siebel eScript when the business service is called. Both the error code and the error message are displayed as an ordinary Siebel error message.

It is strongly recommended that proper error handling be employed when implementing the Java Business Service class. By invoking a SiebelBusinessServiceException, the standard Siebel error handling facilities are employed.

If any other exception is received from doInvokeMethod, then an error is produced with the details of the exception.

About the Lifecycle of a 32-bit Java Business Service

A JVM is created in-process with the Siebel process the first time a Java business service is invoked. Thereafter, the same JVM is used for all invocations of any Java business services.

An instance of the Java class implementing a business service is created the first time that business service is invoked. This instance is released through JNI when the native business service is destroyed. For business services that are not cached, this occurs whenever the caller (workflow, script) releases the native business service. For business services that are cached, this occurs when the session is destroyed (for example the user logs out). For a business service marked as cached in the repository, repeated invocations by a user during a single session invoke methods on the same Java object.

Example of a Java Business Service

Following is an example of a Java class implementing a business service:

```
package com.example.jbs;
import com.siebel.data.SiebelPropertySet;
import com.siebel.eai.SiebelBusinessServiceException;
public class AddBusinessService extends com.siebel.eai.SiebelBusinessService {
public void doInvokeMethod(String methodName, SiebelPropertySet input,
 SiebelPropertySet output) throws SiebelBusinessServiceException {
 String X = input.getProperty("X");
 String Y = input.getProperty("Y");
 if (X == null || X.equals("") || (Y == null) || Y.equals(""))
 throw new SiebelBusinessServiceException("NO_PAR", "Missing param");
 if (!methodName.equals ("Add"))
 throw new SiebelBusinessServiceException("NO_SUCH_METHOD#?, "No such method");
 else {
 int x = 0;
 int y = 0;
 try {
 x = Integer.parseInt(X);
 y = Integer.parseInt(Y);
 }
 catch (NumberFormatException e) {
 throw new SiebelBusinessServiceException("NOT_INT", "Noninteger passed");
 }
 int z = x + y;
 output.setProperty("Z", new Integer(z).toString());
 }
 }
}
```

About the Lifecycle of a 64-bit Java Business Service

An instance of a Java class implementing a business service is created when it is invoked for the first time. This Java instance is stored in the Object Pool that is maintained inside the Java Web Container server. The Java instance is released from the Object Pool when the `destroy` method is called on the business service, or if the business service has been idle for more than 1800 seconds. During a single session, all repeated invocations by a user are invoked on the same Java Object. The Java Object is borrowed from the Object Pool and is returned to the pool after execution of a request.

You can configure the Java Object idle time in the `siebsrvr.properties` file. For the 64-bit Java Business Service, the following parameters have been added to the `siebsrvr.properties` file:

- **Lang.** This parameter defines the language for logging messages. The default value is enu.

- **JBSLogLevel.** This parameter defines the log level for the Java Business Service. The value of this parameter is an integer between 0-5. The default value is 2.
- **JBSSessKeepAlive.** This parameter defines the idle time for a Java Business Service class object in the Object Pool. The default values is 1800 seconds.

Restrictions for Implementing a Java Business Service

When implementing a Java business service, keep in mind the following recommendations and restrictions:

- Each business service method invocation is atomic and stateless.
- The explicit creation of threads is discouraged. It is not recommended that customers invoke a threaded component from a Java business service.
- All data and context required to perform the necessary business functions must be provided as input to the class. The external Java class cannot call back into the Siebel application to obtain additional context.

Troubleshooting the Java Business Service

A common source of errors is the Java CLASSPATH. Remember the following conventions of the Java CLASSPATH:

- On UNIX, CLASSPATH entries are separated by a colon (:); on Windows, by a semicolon (;).
- If .class files are to be used instead of .jar files, then the root directory (for example, the one containing the `com` folder) must be listed in the CLASSPATH.

If the Java business service states that the com.siebel.data.SiebelPropertySet class is not found, then the Siebel.jar files are not correctly specified in the CLASSPATH.

If the Java business service implementation cannot be found, then the .class or .jar file containing its code is not properly specified in the CLASSPATH.

To help troubleshoot CLASSPATH errors, you can use one of the following utilities to see where the Application Object Manager or Web client is looking for the .jar files:

- Windows: filemon. For more information about filemon, see:

  http://www.microsoft.com
- UNIX: truss/strace

6 EAI JMS Transport

EAI JMS Transport

This chapter discusses the EAI JMS Transport business service. It includes the following topics:

- *About the EAI JMS Transport Business Service*
- *About Synchronous and Asynchronous Invocation*
- *About the JMS Publish-and-Subscribe Model*
- *About Operations (Methods) of the EAI JMS Transport*
- *Features Not Supported for Use with the Siebel JMS Transport*
- *About JMS Message Types*
- *About Sending and Receiving XML*
- *About Multistep Operations Within a JMS Session*
- *About Undeliverable Messages in JMS Transport*
- *Detailed Input and Output Specifications for the EAI JMS Transport*
- *Configuring the EAI JMS Transport*
- *Sending and Receiving JMS Messages*
- *Receiving, Dispatching, and Sending JMS Messages*
- *Sending and Receiving Custom JMS Properties*
- *Enabling Authentication and Authorization for the EAI JMS Transport*
- *Troubleshooting for the JMS Transport*
- *About Logging for the JMS Transport*
- *About Caching for the JMS Transport*

About the EAI JMS Transport Business Service

The EAI JMS Transport business service is an API for accessing enterprise messaging systems. It supports the ability to send and receive messages by way of Java Message Service (JMS) servers. JMS defines two messaging models: point-to-point (by way of JMS queues) and publish-and-subscribe (by way of JMS topics). Both are supported by the Siebel EAI JMS Transport.

JMS queues and topics are identified by their Java Naming and Directory Interface (JNDI) names. A JNDI naming service is required to use the EAI JMS Transport. It contains entries for the queues and topics used.

Invoked Business Service methods read the JNDI properties in the Siebel application framework using the classpath defined for a named subsystem and then pass the information using HTTP to the Java layer that resides in the Java Web Container server.

The API of the EAI JMS Transport is very similar to other Siebel messaging APIs such as the EAI MQSeries Server Transport and EAI MSMQ Transport.

The EAI JMS Transport is built using the Java Business Service and therefore inherits all the requirements of that business service. This includes the independent installation of a Java Virtual Machine (JVM) and the configuration of the Siebel application to identify and create the JVM.

Oracle supports integration, using the EAI JMS Transport, with any JMS provider that conforms to the JMS 1.0.2b standard. The EAI JMS Transport provides support for basic integration with both queues and topics, with message types that are specified in *About JMS Message Types*.

Oracle does not support any vendor extensions to the JMS standard except where specified enhanced functionality is released and documented as part of a Siebel CRM release. The EAI JMS Transport provides basic JMS 1.0.2b functionality as described in this chapter. This transport does not provide access to advanced capabilities, such as any capabilities of the JMS 1.1 standard that are not backward-compatible with JMS 1.0.2b, or provide access to any other functionality that is not described in the *Siebel Bookshelf*. For information about JMS standards, see:

http://www.oracle.com/technetwork/java/jms/index.html

About Synchronous and Asynchronous Invocation

Like the EAI MQSeries Server Transport, the EAI JMS Transport has two modes of execution: synchronous and asynchronous. Synchronous execution involves invoking individual methods of the JMS Transport directly, just like any other business service. Because the caller waits for the method to return, such invocation is synchronous. Asynchronous execution means listening for messages arriving on a particular queue and taking action whenever one arrives. This involves the creation of a separate Siebel component, called a JMS Receiver. Like the MQ Receiver, whenever a message arrives on the queue, the JMS Receiver *dispatches* to a business service (or workflow) and optionally sends a reply message.

> **Note:** The JMS Receiver uses the EAI JMS Transport business service but cannot dispatch to a workflow that either uses this business service as one of its steps or dispatches directly to this business service. While in-process re-entrance is not supported, you can indirectly invoke the EAI JMS Transport as one of the steps out of process by calling the Synchronous Server Requests business service.

About the JMS Publish-and-Subscribe Model

The traditional message model, where a message is sent to a queue and later removed by a single receiver, is called point-to-point messaging. In addition to this familiar model, JMS also supports the publish-and-subscribe messaging model. Here, messages are *published* to *topics*, rather than sent to queues. Interested receivers *subscribe* to individual topics and receive a copy of each message published to the topic. To subscribe, a subscriber registers with the topic, providing a unique identifier.

For more information about the JMS publish-and-subscribe model, see:

http://www.oracle.com/technetwork/java/jms/index.html

JMS queues and topics are identified by their JNDI names. A JNDI naming service is required to use the JMS Transport. The JNDI naming service contains entries for the JMS queues (implementers of javax.jms.Queue) and topics (implementers of javax.jms.Topic) used, as well as the necessary JMS connection factories (implementers of either javax.jms.QueueFactory or javax.jms.Topic).

All methods that receive messages automatically time out if no message is available. The timeout length is three seconds by default and can be specified by the ReceiveTimeout argument.

CAUTION: The ReceiveTimeout must not be set to value less than or equal to 0.

Whether a call to Receive or Subscribe timed out is provided as the TimedOut property of the output property set.

Subscriptions to JMS topics are always *durable* subscriptions.

The term *dispatch* is used to refer to the operation of calling a business service or workflow, passing as input the content of a newly received message.

About Operations (Methods) of the EAI JMS Transport

The following is a summary of supported operations for use with the EAI JMS Transport:

- **Receive.** Receive a message from a JMS queue.
- **ReceiveDispatch.** Receive a message from a JMS queue, then dispatch.
- **ReceiveDispatchSend.** Receive a message from a JMS queue, dispatch, and then send the result to a (possibly different) JMS queue.
- **Send.** Send a message to a JMS queue.
- **SendReceive.** Send a message to a JMS queue then receive a message from a (possibly different) JMS queue. The JMSCorrelationID header of the reply message must be equal to the JMSCorrelationID of the message sent, unless it is null (if none was provided as an input to SendReceive), in which case it must be the JMSMessageID of the message sent.
- **Subscribe.** Receive a message from a JMS topic. The subscriber identifier must be supplied as an input to this method.
- **SubscribeDispatch.** Receive a message from a JMS topic, then dispatch. The subscriber identifier must be supplied as an input to this method.
- **Publish.** Publish a message to a JMS topic.

The arguments to these methods and their exact semantics (along with valid values, default values, and so on) are described in the topic *Detailed Input and Output Specifications for the EAI JMS Transport*. All methods require the JNDI name of JMS ConnectionFactory and the JNDI name of the queue or topic.

Features Not Supported for Use with the Siebel JMS Transport

The following features are not supported for use with the Siebel JMS Transport:

- **Message Selection.** JMS has a feature called Message Selection, by which a receiver or subscriber can filter the messages it receives by specifying certain criteria. This feature is not supported by the Siebel JMS Transport.
- **Concurrency with non-JMS messaging.** It is not recommended that JMS messaging be used concurrently (for a single queue) with non-JMS messaging. For example, it is not recommended that a message be sent by way

of JMS and later read using native tools. JMS vendors do not typically support such usage; it can result in the appearance of additional headers or additional obscure data in the body of the message.

- **Transport Layer Security (TLS).** The Siebel JMS Transport is primarily designed to support message exchange with external messaging systems (providers) using the JMS 1.0.2b standard. The JMS standard is not bound to transport layers, such as TCP/IP, and does not address transport layer-specific features, such as securing TCP/IP socket connections using TLS. For information about enabling and using TLS with the Siebel JMS Transport, contact the vendor of your JMS system. For information about the JMS 1.0.2b standard, see:

  http://www.oracle.com/technetwork/java/jms/index.html

About JMS Message Types

JMS defines five types of messages: TextMessage, BytesMessage, ObjectMessage, MapMessage, and StreamMessage. The Siebel JMS Transport supports only the types TextMessage and BytesMessage. If the JMS Transport receives an ObjectMessage, MapMessage, or StreamMessage from the JMS server, then the error *Unsupported Message Type* is produced.

Like all Siebel business services, the output of any method is a property set. If a BytesMessage is received, then the value of the property set has Binary type. If a TextMessage is received, then the value has String type.

Conversely, the input to any method is also a property set. For methods that involve sending or publishing a message, the type of message sent or published depends on the type of the value of the input property set. If the type is Binary, then a BytesMessage is sent and published. If the type is String, then a TextMessage is sent and published.

Note: The Siebel Business Service Simulator in Siebel Call Center always creates the input with a value type of String.

About Sending and Receiving XML

Messages whose content is XML are generally best treated as binary data and sent as BytesMessages. For example, the output of the Siebel business service EAI XML Converter is binary; therefore, if this is passed as the input to Send, then a BytesMessage is sent.

If XML is sent as a TextMessage, then the characters are encoded as UTF-16. Therefore, the XML document declares its encoding to be UTF-16.

Typically, when a message containing an XML document is received by the Siebel application, it is desirable to convert the document to a property set representation before processing it. This is accomplished automatically during the Dispatch step by specifying the ConverterService argument to be either XML Converter or EAI XML Converter. For more details about these converter services see *XML Reference: Siebel Enterprise Application Integration* .

About Multistep Operations Within a JMS Session

All JMS operations are performed in the context of a transactional JMS QueueSession. If a send or receive operation throws an exception, then the session is immediately rolled back. If the operation is successful, then the session

is committed, unless the operation is part of a larger multistep operation. In the case of multistep operations, the transaction is handled as follows:

- **SendReceive.** If the send operation succeeds, then the JMS session is committed and a receive operation is performed. This is necessary because the receive operation might depend on a response to the first message.
- **ReceiveDispatch.** If the receive operation fails, then the JMS session is rolled back, and the dispatch operation is not attempted. If the receive operation succeeds, then the dispatch operation is attempted. If the dispatch succeeds, then the JMS session is committed; otherwise, both the Siebel transaction and the JMS session are rolled back.
- **SubscribeDispatch.** Same as ReceiveDispatch.
- **ReceiveDispatchSend.** If the receive operation fails, then the JMS session is rolled back, and further operations are not attempted. If the receive operation succeeds, then the dispatch operation is attempted. If the dispatch operation fails, then the JMS session and the Siebel transaction are rolled back; otherwise, the send operation is attempted. If the send operation fails, then the JMS session and the Siebel transaction are rolled back; otherwise, both are committed.

Each Dispatch operation is performed within a Siebel transaction.

> **Note:** Do not attempt ReceiveDispatch and ReceiveDispatchSend operations from within an existing Siebel transaction, as nested transactions are not supported.

Also, as with all Siebel EAI receivers, if an operation fails during the execution of the JMS Receiver, then the JMS Receiver component terminates. (A timeout is not a failure.)

About Undeliverable Messages in JMS Transport

If a message is undeliverable, in the sense that repeated attempts by the Siebel JMS Transport to receive the message fail, then the message must be removed from the queue. Most JMS vendors provide some mechanism for dealing with such “poison messages” Oracle WebLogic, for example, can be configured to limit the number of times it attempts to deliver a message before redirecting the message to an error queue or deleting the message altogether.

Detailed Input and Output Specifications for the EAI JMS Transport

This topic provides detailed information about the exact semantics of all input arguments and output values for each method of the EAI JMS Transport.

The following topics are discussed:

- *JMS Headers and Properties*
- *Input Arguments Used by the Dispatch Step*
- *About the Output of the JMS Transport*

JMS Headers and Properties

Every JMS message has a set of standard *headers*. Some of these headers can be specified as arguments to the methods of the JMS Transport that involve sending or publishing, and some are available as properties of the output property set of methods that involve receiving or subscribing. These are detailed in the tables in *Input Arguments Used by the Dispatch Step*, *Input Argument Values*, and *About the Output of the JMS Transport*.

A JMS message can also be assigned *properties*. These might be user-defined properties specific to a particular application, or JMS-defined properties (for example JMSXProducerTXID) that are optionally supported by the JMS vendor. A property can be an instance of any Java class or any of the primitive Java types. All properties of a message received by the Siebel JMS Transport are available as properties of the output property set.

The name of the property is the original name with the eleven characters SIEBEL_JMS: prefixed; the value is the string obtained by converting the original value to a Java String. Conversely, when a message is sent, any property of the input property set whose name begins with SIEBEL_JMS: is added to the message as a JMS Message string property with the prefix SIEBEL_JMS: removed. For example, the property SIEBEL_JMS:foo is added to the message as the string property foo.

Input Arguments Used by the Dispatch Step

The following table shows the options for each input argument of the JMS Transport methods, except the user-defined properties and arguments used by the Dispatch step. R denotes a required argument; NR denotes an optional argument (not required); and I denotes an argument that is ignored. Notes following the table provide further explanation for particular values.

Input Argument	Send	Publish	Send Receive	Receive	Subscribe	Receive Dispatch	Receive Dispatch Send	Subscribe Dispatch
ConnectionFactory	R	R	R	R	R	R	R	R
ReceiveQueue	I	I	R	R	I	R	R	I
ReceiveTimeout	I	I	NR	NR	NR	NR	NR	NR
SendQueue	R	I	R	I	I	I	R	I
Topic	I	R	I	I	R	I	I	R
ConnectionUsername	NR	NR	NR	NR	NR	NR	NR	NR
ConnectionPassword	NR	NR	NR	NR	NR	NR	NR	NR
SendUsername	NR	I	NR	I	I	NR	I	I
SendPassword	NR	I	NR	I	I	NR	I	I

Input Argument	Send	Publish	Send Receive	Receive	Subscribe	Receive Dispatch	Receive Dispatch Send	Subscribe Dispatch
ReceiveUsername	I	I	NR	I	I	I	NR	I
ReceivePassword	I	I	NR	I	I	I	NR	I
TopicUsername	I	NR	I	I	NR	I	I	NR
TopicPassword	I	NR	I	I	NR	I	I	NR
SubscriberIdentifier	I	I	I	I	R	I	I	R
JMS Headers								
JMSPriority	NR	NR	NR	I	I	I	NR 2	I
JMSDeliveryMode	NR	NR	NR	I	I	I	NR 2	I
JMSExpiration	NR	NR	NR	I	I	I	NR 2	I
JMSReplyTo	NR	NR	I	I	I	I	NR 2	I
JMSType	NR	NR	NR	I	I	I	NR 2	I
JMSCorrelationID	NR	NR	NR	I	I	I	I	I
Dispatch								
Connection Subsystem	NR	NR	NR	NR	NR	NR	NR	NR
DataHandling Subsystem	I	I	I	I	I	NR	NR	NR
DispatchService	I	I	I	I	I	R	R	R
DispatchMethod	I	I	I	I	I	R	R	R
DispatchWorkflow Process	I	I	I	I	I	R	R	R
DispatchRuleSet	I	I	I	I	I	R	R	R

Input Argument	Send	Publish	Send Receive	Receive	Subscribe	Receive Dispatch	Receive Dispatch Send	Subscribe Dispatch
ConverterService	I	I	I	I	I	NR	NR	NR

Some special notes regarding particular values in the previous table:

- When the JMSReplyTo header is used, the SendQueue value is ignored.
- When the JMSPriority, JMSDeliveryMode, JMSExpiration, JMSReplyTo, or JMSType header is used, these values are assigned to the reply message during the Send step.
- The JMSReplyTo header of the sent message is set to the value of the ReceiveQueue argument.
- The JMSCorrelationID header of the reply message cannot be set directly. The JMSCorrelationID of the reply message is set to the JMSCorrelationID of the received message, unless empty, in which case it is set to its JMSMessageID.
- For DispatchService, DispatchMethod, DispatchWorkflowProcess, and DispatchRuleSet method arguments, one of the following three combinations is required:
  - (DispatchService && DispatchMethod)
  - DispatchWorkflowProcess
  - DispatchRuleSet
- The ConnectionUsername and ConnectionPassword input parameters apply to IBM WebSphere MQ only.

  > **Note:** When sending messages to IBM WebSphere MQ, ConnectionUsername and ConnectionPassword are required for the Windows 2012 Server platform and recommended for all other Windows platforms.

- The SendUsername, SendPassword, ReceiveUsername, ReceivePassword, TopicUsername, and TopicPassword input parameters apply to Oracle WebLogic only.
- The JMSType and JMSCorrelationID input arguments can also be used as output arguments.
- For the ConnectionSubsystem input argument, a subsystem can be provided instead of the connection parameters. However, it must contain the same required method arguments as used for the connection parameters.
- For the DataHandlingSubsystem input argument, a subsystem can be provided instead of the dispatch parameters. However, it must contain the same required method arguments as used for the dispatch parameters.
- The ConverterService input argument is used to process the output of the received message before dispatching.

In place of providing the arguments individually, the single argument ConnectionSubsystem can be provided. Its value must be the name of a valid named subsystem of type JMSSubsys, and it must include all of the arguments that are required by the method to which it is passed. See *About the JMS Receiver* for more information about that named subsystem.

JMS message properties are also supported as input arguments (properties) as described in *JMS Headers and Properties*.

Input Argument Values

The following table provides details for each input argument about the allowable values, default values, and special values, as well as the behavior if an invalid value is passed.

Input	Default	Allowable Values	Special Values	If Value Invalid
ConnectionFactory	NONE	JNDI connection factory name	Not applicable	ERROR
ReceiveQueue	NONE	JNDI queue name	Not applicable	ERROR
ReceiveTimeout	3000	Any integer greater than to 0 **CAUTION:** Be careful NOT to set the value less than or equal to 0.	Not applicable	Noninteger defaults to 3000
ConnectionUsername	NONE	Valid username	Not applicable	Not applicable
ConnectionPassword	NONE	Valid password	Not applicable	Not applicable
SendQueue	NONE	JNDI queue name	Not applicable	ERROR
SendUsername	NONE	Valid username	Not applicable	Not applicable
SendPassword	NONE	Valid password	Not applicable	Not applicable
ReceiveUsername	NONE	Valid username	Not applicable	Not applicable
ReceivePassword	NONE	Valid password	Not applicable	Not applicable
TopicUsername	NONE	Valid username	Not applicable	Not applicable
TopicPassword	NONE	Valid password	Not applicable	Not applicable
Topic	NONE	JNDI topic name	Not applicable	ERROR
SubscriberIdentifier	NONE	ANY STRING	Not applicable	Not applicable
JMS Headers				
JMSCorrelationID	NOT SET	ANY STRING	Not applicable	Not applicable

Input	Default	Allowable Values	Special Values	If Value Invalid
JMSPriority	javax.jms.Message. DEFAULT_PRIORITY (4)	Any integer from 0 to 9	(0 lowest; 9 highest)	DEFAULT
JMSDeliveryMode	javax.jms.Delivery Mode.PERSISTENT	PERSISTENT, NON_ PERSISTENT	Not applicable	DEFAULT
JMSExpiration	javax.jms.Message. DEFAULT_TIME_TO_ LIVE (0)	Any nonnegative integer	0: Message never expires	DEFAULT
JMSReplyTo	NOT SET	JNDI queue name	Not applicable	ERROR
JMSType	SiebelJMSMessage	ANY STRING	Not applicable	Not applicable
Dispatch				
ConnectionSubsystem	NONE	A JMSSubsys named subsystem	Not applicable	ERROR
DataHandlingSub system	NONE	An EAITransportData HandlingSubsys named subsystem	Not applicable	ERROR
DispatchService	NONE	Business service name	Not applicable	ERROR
DispatchMethod	NONE	Business service method	Not applicable	ERROR
DispatchWorkflow Process	NONE	Workflow name	Not applicable	ERROR
DispatchRuleSet	NONE	Rule set name	Not applicable	ERROR
ConverterService	NONE	Business service name	Not applicable	ERROR

About the Output of the JMS Transport

The output of the JMS Transport methods includes the following parts:

- The content of the received message (if the method involves receiving a message). See the previous topic, *Input Arguments Used by the Dispatch Step*, for details about typing.
- JMS properties of the received message (if the method involves receiving a message), as described in the topic *JMS Headers and Properties*.
- Certain JMS headers of the message sent or received, as described in the following table.

- The special properties TimedOut (if the method involves receiving a message) and DispatchError (if the method involves dispatching), as described in the following table. Each property is either True or False.

The following table enumerates for each method of JMS Transport the JMS headers and other distinguished properties that appear as properties of the output property set of the method. Yes means the argument is present; No means the argument is absent.

Output	Send	Publish	Send Receive	Receive	Subscribe	Receive Dispatch	Receive Dispatch Send	Subscribe Dispatch
TimedOut	No	No	Yes	Yes	Yes	Yes	Yes	Yes
JMSType +	No	No	Yes	Yes	Yes	Yes	No	Yes
JMSCorrelation ID+	No	No	Yes	Yes	Yes	Yes	No	Yes
JMSRedelivered	No	No	Yes	Yes	Yes	Yes	No	Yes
JMSTimestamp	No	No	Yes	Yes	Yes	Yes	No	Yes
JMSMessageID	Yes [1]	Yes [1]	Yes [2]	Yes [2]	Yes [2]	Yes	Yes [1]	Yes
DispatchError	No	No	No	No	No	Yes	Yes	Yes

Some special notes regarding the information in this table:

- **Yes[1]:** JMSMessageID, the value assigned by the JMS server of the sent (or published) message.
- **Yes[2]:** JMSMessageID, the value assigned by the JMS server of the received (or subscribed) message.
- **+:** An output argument that can also be used as an input argument.

All other message properties (user-defined; not JMS headers) are provided as output properties with SIEBEL_JMS: prefixed to the original property name, and the value is converted to a String.

For the multistep methods ReceiveDispatch, ReceiveDispatchSend, and SubscribeDispatch, properties are passed between the individual steps according to the following rules:

- All outputs of the Receive (or Subscribe) step are passed as inputs to the subsequent Dispatch step.
- In the case of an error during the Dispatch step, its output is returned.
- The input to the Dispatch step includes all properties in the original input as well as properties returned by the Receive (or Subscribe) step.

Configuring the EAI JMS Transport

The EAI JMS Transport is built using the Java Business Service and therefore inherits all the requirements of that business service. This includes the independent installation of a Java Virtual Machine (JVM) and the configuration of the Siebel application to identify and create the VM. Configuration of the Siebel application requires creating a named

subsystem of type JVMSubSys with the necessary properties. Refer to the Java Business Service documentation for instructions on how to configure the JVM named subsystem.

The EAI JMS Transport requires that the CLASSPATH property of the JVM subsystem include the following packages or classes:

- Siebel.jar
- SiebelJI_lang.jar (where lang corresponds to the default language for your installation)
- A directory containing the location of the jndi.properties file

  The jndi.properties file contains the necessary name value pairs required to perform a JNDI lookup and bind to the remote queue.
- Necessary classes and JAR files as required by the JMS provider.

Note: You can have only one JVM loaded in a process, and therefore only one JVM subsystem in a process. If you try to load more than one JVM subsystem into a process, then an error occurs.

If you want multiple JVM subsystems, then you must configure additional components. For example, you can have EAIObjMgr_WL pointing to a JVM subsystem called JAVA_WL and EAIObjMgr_ORACLE pointing to a JVM subsystem called JAVA_ORACLE.

To verify that the CLASSPATH and jndi.properties are properly configured, see *Troubleshooting for the JMS Transport*.

The following JMS Transport configuration topics are also discussed here:

- *About the JMSSubsys Named Subsystem*
- *About the JavaContainerSubsys Named Subsystem*
- *About the JMS Receiver*
- *About Reconnecting to the External JMS Queue*
- *Creating a JMS Subsystem by Using the Siebel Web Client*
- *Tuning the EAI JMS Transport*

About the JMSSubsys Named Subsystem

The arguments to any method of JMS Transport can be supplied individually as properties of the input property set or as part of a named 32-bit subsystem of type JMSSubsys. When invoking the JMS Transport asynchronously by starting a JMS Receiver component, the arguments must be supplied by way of a named subsystem.

This subsystem supplies all of the necessary parameters for any one of these three methods: ReceiveDispatch, ReceiveDispatchSend, or SubscribeDispatch. The parameters for the three methods are ConnectionFactory, ReceiveQueue, SendQueue, Topic, SubscriberIdentifier, ReceiveTimeout, JMSType, JMSPriority, JMSExpiration, and JMSDeliveryMode.

In addition, this subsystem has a property JVMSubsys, which can be given the name of the JVM subsystem instance to use. The default value is JAVA. Therefore, if the property JVMSubsys is not explicitly given a value, then there must be a properly configured instance of the type JVMSubSys named JAVA.

About the JavaContainerSubsys Named Subsystem

The arguments to any method of JMS Transport can be supplied individually as properties of the input property set or as part of a named 64-bit subsystem of type JavaContainerSubsys. When invoking the JMS Transport asynchronously by starting a JMS Receiver component, the arguments must be supplied by way of a named subsystem.

This subsystem supplies all of the necessary parameters for any one of these three methods: ReceiveDispatch, ReceiveDispatchSend, or SubscribeDispatch. The parameters for the three methods are CONTAINERURL and CLASSPATH.

For more information about configuring the JavaContainerSubsys subsystem, see *Creating a 64-bit Java Subsystem by Using the Siebel Server Manager* and *Creating a 64-bit Java Subsystem by Using the Siebel Web Client*.

About the JMS Receiver

The JMS Receiver (alias EAIJMSRcvr) is a Siebel Server component that makes it possible for the JMS Transport to be invoked asynchronously.

> **Note:** In Siebel CRM 21.2 update and later, any changes to repository objects, such as a business service, business component or an integration object, used by JMSReceiver don't require a restart of the JMS Receiver component. These changes are available immediately once the changed workspace is delivered or migrated.

The JMS Receiver:

- Listens for messages arriving on a JMS queue or topic and takes action whenever a message arrives.
- Repeatedly invokes a single method of the JMS Transport: ReceiveDispatch, ReceiveDispatchSend, or SubscribeDispatch.
- Takes a message from the queue and dispatches it to the corresponding workflow or business service for execution. If execution is successful, then the message is committed to the queue. If there is an error, then the message is rolled back to the queue.
- Uses AUTO_ACKNOWLEDGE mode. In AUTO_ACKNOWLEDGE mode, the session automatically acknowledges the receipt of a message when it has either successfully returned from a call to receive or the message listener it has called to process the message successfully returns.

About Multithreading in the JMS Receiver Component

The JMS Receiver is multithreaded and operates in batch mode. (In some previous versions, this component was single-threaded and ran in the background.)

A task for the JMS Receiver component starts automatically when the Siebel Server is started, where Default Tasks (alias DfltTasks) is set to 1. Otherwise, you must start tasks manually. For more information about DfltTasks, see *Siebel System Administration Guide* .

When a JMS Receiver task is started, a main task and several worker threads are created, whose number depends on the MinWorkQThreads and MaxWorkQThreads parameters listed in the following table.

After the worker threads are created, the main task thread starts calling the specified method on the EAI JMS Transport business service in an infinite loop, until a shutdown component is signaled. At the same time, the worker threads start

their own infinite loop and perform the same duties as the main thread: they call the specified method on the EAI JMS Transport business service.

In effect, the ReceiveDispatchSend operation (or any method specified) is now called by multiple threads in the same task. Previously, a single sequential call to the ReceiveDispatchSend method was made in a single process. Now, the same operation happens in parallel. The thread parallelism effectively increases scalability while limiting CPU load and memory utilization.

Parameter	Description
MaxTasks	Total number of tasks that can run concurrently on a Siebel Application Object Manager. For more information about MaxTasks, see *Siebel Performance Tuning Guide* .
MaxMTServers	Maximum number of multithreaded processes that can run concurrently on a Siebel Application Object Manager. For more information about MaxMTServers, see *Siebel Performance Tuning Guide* .
MinMTServers	Minimum number of multithreaded processes that can run concurrently on a Siebel Application Object Manager. For more information about MinMTServers, see *Siebel Performance Tuning Guide* .
MaxWorkQThreads	Maximum number of worker threads in a process. The default is 4.
MinWorkQThreads	Minimum number of worker threads in a process. The default is 4.
MaxWorkQLength	Maximum number of work items handled by a worker thread. The default is 20.

Considerations When Using Multithreading

Multithreading works best when messages are atomic; that is, their processing does not depend on the processing of other messages. The messages can be processed in parallel without conflicts.

If one message is dependent on another, then the messages must be processed in the correct order. For example, you must create an account before creating a service request for it, and create an order before adding an order line item.

You can use validation scripting to make sure that parents are created before children. However, creating a data architecture that does not require additional scripting will deliver a performance benefit to the end-to-end solution.

About Configuring the JMS Receiver

An instance of the JMS Receiver is configured with the parameters of a JMSSubsys named subsystem, which specify the queue or topic to listen to, and the action to be taken.

The JMSReceiver task has the following three parts:

- ReceiverConnectionSubsystem is the named subsystem.
- ReceiverDataHandlingSubsystem dispatches the message from the ReceiveQueue to the workflow previously defined.
- ReceiverMethodName is the EAI JMS Transport business service method invoked.

The following is an example of how an instance of the JMS Receiver can be configured and run by using the Siebel Server Manager command-line interface:

```
create named subsystem MyJMSConnSubsys_SR for subsystem JMSSubsys with
```

```
ConnectionFactory="weblogic.examples.jms.QueueConnectionFactory",
ReceiveQueue="weblogic.examples.jms.exampleQueueReceive",
SendQueue="weblogic.examples.jms.exampleQueueSend",
ReceiveTimeout=3000

create named subsystem SiebelEcho for subsystem EAITransportDataHandlingSubsys with
DispatchService="Workflow Utilities",
DispatchMethod="ECHO"

start task for comp JMSReceiver with
ReceiverConnectionSubsystem=MyJMSConnSubsys_SR,
ReceiverDataHandlingSubsystem=SiebelEcho,
ReceiverMethodName=ReceiveDispatchSend
```

For a detailed workflow example using a JMS Receiver, see *Receiving, Dispatching, and Sending JMS Messages*. For a discussion of named subsystems for Siebel EAI, see *EAI Transports and Interfaces Overview* For more information about administering named subsystems, see *Siebel System Administration Guide* .

About Reconnecting to the External JMS Queue

If the external system is not ready to receive messages, then the JMS Receiver component fails when it attempts to connect. To avoid this failure, two new parameters have been added to the JMS Receiver: CompMaxRetries and CompRetryInterval. You can set these parameters as needed.

The reconnection parameters for the JMS Receiver are described in the following table.

Parameter	Default Value	Description
CompMaxRetries	10	Specifies the number of times the reconnection is attempted. Valid values are positive integers.
CompRetryInterval	60	Specifies the interval in seconds between each retry. Valid values are positive integers.

These parameters are used with the AutoRestart and NumRestart parameters. AutoRestart enables restart attempts when set to TRUE. NumRestart determines how many attempts are made to restart the JMS Receiver task if it fails.

CompMaxRetries and CompRetryInterval control reconnection attempts to the JMS queue to avoid JMS Receiver failure. AutoRestart and NumRestart are used when the JMS Receiver task ends with an error unrelated to losing the queue connectivity. For more information about NumRestart and AutoRestart, see *Siebel System Administration Guide* .

Creating a JMS Subsystem by Using the Siebel Web Client

The following is an alternative procedure for creating a JMS Subsystem by using the Siebel Web Client and then configuring the JMS Transport.

To configure the JMS Transport by using the Siebel Web Client

1. In the Siebel client, navigate to the Administration - Server Configuration screen, Enterprises view.
2. In the first list applet, select the Enterprise Server that you want to configure.

3. In the middle applet, click the Profile Configuration tab.
4. Click New to create a new component profile and set the following parameters:

Name	Value
Profile	JMS_Q1ReceiveDispatchSend
Alias	JMS_Q1ReceiveDispatchSend
Subsystem Type	JMSSubsys

5. In the Profile Parameters list applet (the last applet), specify the parameters required for the type of operations the subsystem will support (for example, Receive or ReceiveDispatchSend).

   For example, if this subsystem needed to support the ReceiveDispatchSend operation, then at least the following values must be set:

Name	Value
ConnectionFactory	examples.jms.QueueConnectionFactory
JVM Subsystem	JAVA
ReceiveQueue	examples.jms.fromSiebel
SendQueue	examples.jms.toSiebel
Receive Timeout	1000

Tuning the EAI JMS Transport

As of Siebel CRM 20.6 Update, two tuning parameters have been introduced for JBS (Java Business Service) and JMS (Java Message Service). You can set these parameters in `applicationcontainer_internal\webapps\configagent.properties` in the Siebel CRM installation.

- **JBSSessKeepAlive.** Specifies the time interval (in seconds) at which the connection clean-up thread runs. The default value is 1800 seconds (30 minutes). The suggested value is the maximum time that the system takes to process or dispatch a single message, plus some HTTP communication time between the Siebel process and the Apache Tomcat process for the application container. This ensures that connections are not removed while a message is being processed or dispatched, which could lead to queue rollback and duplicate messages.

- **JBSSessEvictPerRun.** Specifies the number of idle connections objects that the clean-up thread must try to evict in each clean-up run. The value must be a positive number. The default value is 20. Ideally, the value is equal to the maximim number of JMS tasks that run on that server, so that all connections are properly cleaned up.

To set these parameters, do the following (values are examples only):

1. For each application container, edit `applicationcontainer_internal\webapps\configagent.properties` and add these two lines to the end of the file:

```
JBSSessKeepAlive = 30 (seconds)
JBSSessEvictPerRun = 100 (number of idle connections)
```

2. Restart the application container.

After you configure these parameters (with the example values):

- Every 30 seconds, Apache Tomcat runs an eviction thread to close idle connections to MQ Receiver or JMS Receiver.
- Tomcat closes idle connections that have been idle for more than 30 seconds, to a limit of 100 connections.
  - At 30 seconds, Tomcat does nothing, because no connections have been idle for 31 secs. However, at 60 seconds, Tomcat will likely find connections that have been idle for 31 or more seconds. This process repeats itself every 30 seconds.
  - If there are fewer than 100 idle connections, then Tomcat closes all of them. If there are more than 100 idle connections, then Tomcat stops at 100 and leaves the rest for the next run.

These settings are just an example. Tune these parameters according to the loads experienced in your environment, striving for an appropriate balance:

- For JBSSessKeepAlive, values that are too high yield more idle connections, leaving less room for new connections. Values that are too low lead to early closing of connections (even though connections could have been reused), leading to increased cycles of opening and closing, and consuming more CPU due to increased clean-up cycles.
- For JBSSessEvictPerRun, values can be as high as the Maximum Tasks (MaxTasks) value for that server (cumulative of the JMS Receiver plus any other process that connects to the JMS server). If you set this parameter on the higher end (such as equal to MaxTasks), then all connections are considered for clearing if they are idle.

Sending and Receiving JMS Messages

The following procedure describes how to set up the Siebel application to send a message to an external system using the EAI JMS Transport and receive a corresponding reply from the external system.

To send and receive messages with the JMS Transport

1. Set up a JMS queue to receive messages from the Siebel application and give the queue an easy-to-identify name, such as fromSiebel.

   Refer to your JMS provider documentation on how to administer, monitor, and define new persistent queues.
2. Set up a JMS queue to send messages to the Siebel application.

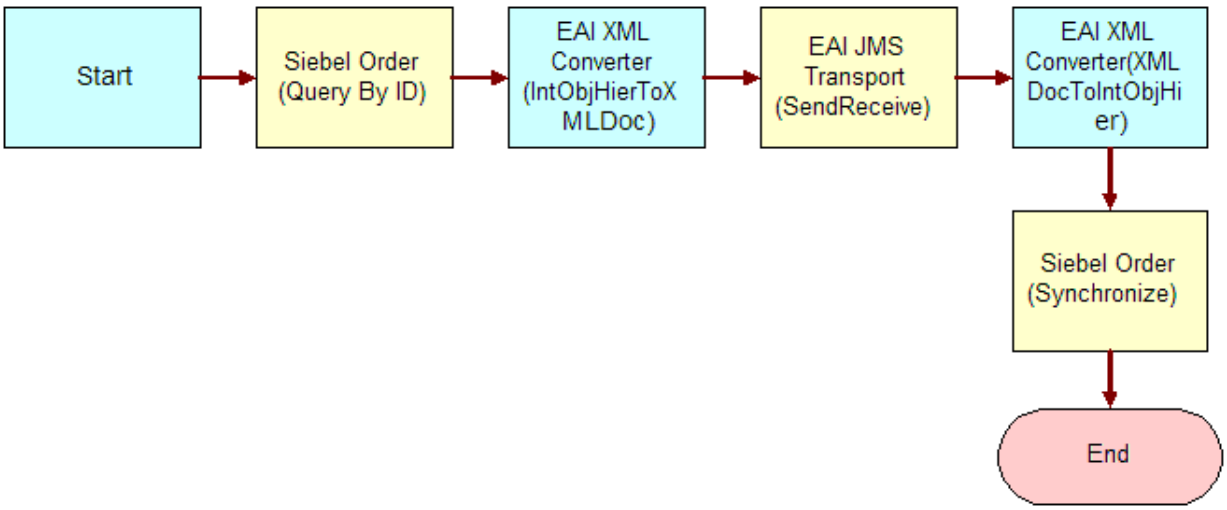
   Refer to your JMS provider documentation on how to administer, monitor and define new persistent queues.

a. Give the queue an easy-to-identify name, such as toSiebel.
b. Create a message in the queue.

> **Note:** To test this scenario adequately, you must have a partner application that can place a valid message for the Siebel application into the queue.

3. Set up a workflow in Siebel Tools to send a message out and receive a message back in response using the EAI JMS Transport. The workflow should contain the following steps, as shown in the following image:

   a. Siebel Order (Query by ID)
   b. EAI XML Converter (IntObjHierToXMLDoc)
   c. EAI JMS Transport (SendReceive)
   d. EAI XML Converter (XMLDocToIntObjHier)
   e. Siebel Order (Synchronize)



> **Note:** For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide* .

4. Create the following process properties:

Name	Data Type	In/Out	Default String	Comments
OrderXML	Binary	In	Not applicable	Not applicable
JMSConnectionFactory	String	In	examples.jms. ConnectionFactory	JNDI name of the JMS connection factory
JMSReceiveQueue	String	In	examples.jms.toSiebel	JNDI name of the queue
JMSSendQueue	String	In	examples.jms.fromSiebel	JNDI name of the queue

Name	Data Type	In/Out	Default String	Comments
JMSReceiveTimeout	String	In	180000	Not applicable
Order Message	Integration Object	In	Not applicable	Not applicable

5. Set up the first step of the workflow to use the Siebel Order ASI with the QueryById method to query the information from the Siebel database using the following input and output arguments:

Input Argument	Type	Property Name
PrimaryRowId	Process Property	Object Id

Property Name	Type	Output Argument
Order Message	Output Argument	SiebelMessage

6. Set up the second step of the workflow to use the EAI XML Converter with the IntObjHierToXMLDoc method to convert the data extracted from the Siebel database to XML using the following input and output arguments:

Input Argument	Type	Value	Property Name
GenerateProcessingInstructions	Literal	False	Not applicable
SiebelMessage	Process Property	Not applicable	Order Message

Property Name	Type	Output Argument
OrderXML	Output Argument	<Value>

7. Set up the third step of the workflow, after Start, to use the EAI JMS Transport with the SendReceive method using the following input and output arguments:

Input Argument	Type	Property Name
<Value>	Process Property	OrderXML

Input Argument	Type	Property Name
ConnectionFactory	Process Property	JMSConnectionFactory
ReceiveQueue	Process Property	JMSReceiveQueue
ReceiveTimeout	Process Property	JMSReceiveTimeout
SendQueue	Process Property	JMSSendQueue

Property Name	Type	Output Argument
OrderXML	Output Argument	<Value>

8. Set up the fourth step to use the EAI XML Converter with the XMLDocToIntObjHier method to convert the XML message to an Integration Object using the following input and output arguments:

Input Argument	Type	Property Name
<Value>	Process Property	OrderXML

Property Name	Type	Output Argument
Order Message	Output Argument	SiebelMessage

9. Set up the last step to use the Siebel Order ASI with the Synchronize message to update the Siebel database using the following input and output arguments:

Input Argument	Type	Property Name
SiebelMessage	Process Property	Order Message

Property Name	Type	Output Argument
Order Message	Output Argument	SiebelMessage

10. Save and deploy the workflow.

    It is recommended that the Workflow Simulator be used for testing purposes.

    > **Note:** To test this scenario adequately, you must have a partner application that can accept the message and return a response. The correlation ID of the response message must be set to the message ID of the message originally sent by the Siebel application.

Receiving, Dispatching, and Sending JMS Messages

The procedure in this section describes how to set up your system to receive inbound messages from JMS, perform an action within the Siebel application based upon the message, and send a synchronous response back to the external system.

To receive, dispatch, and send messages using EAI JMS Transport

1. Set up a JMS queue to receive messages from the Siebel application and give the queue an easy to identify name, such as fromSiebel.

   Refer to your JMS provider documentation on how to administer, monitor, and define new persistent queues.
2. Set up a JMS queue to send messages to the Siebel application.

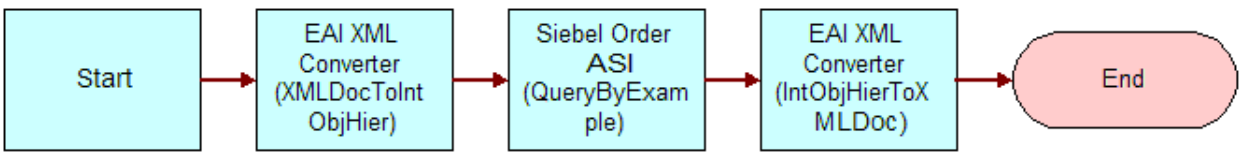
   Refer to your JMS provider documentation on how to administer, monitor and define new persistent queues.

   a. Give the queue an easy-to-identify name such as toSiebel.
   b. Create a message in the queue.

   > **Note:** To test this scenario adequately, you must have a partner application that can place a valid message for the Siebel application into the queue.

3. Set up a workflow in Siebel Tools to process the incoming XML request. The workflow should contain the following steps, as shown in the following image:

   a. EAI XML Convert (XMLDocToIntObjHier): Receives an incoming XML document and converts it to an integration object.
   b. Siebel Order ASI (QueryByExample): Executes a query using Siebel Order application service.
   c. EAI XML Converter (IntObjHierToXMLDoc): Converts the response to an XML document.



   > **Note:** For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide* .

4. Create the following process properties:

Name	Data Type	In/Out	Default String	Comments
<Value>	Binary	In/Out	<Value>	Order Integration Object
Order Message	Hierarchy	In/Out	Not applicable	XML representation of the integration object

5. Set up the first step of the workflow, after Start, to use the EAI XML Converter with the XMLDocToIntObjHier method.

   This step converts the incoming XML document to an integration object representation using the following input and output arguments:

Input Argument	Type	Property Name
<Value>	Process Property	<Value>

Property Name	Type	Output Argument
Order Message	Output Argument	SiebelMessage

6. Set up the second step to use the Siebel Order ASI with the QueryByExample method.

   This step queries the Order business object based upon the provided XML document using the following input and output arguments:

Input Argument	Type	Property Name
SiebelMessage	Process Property	Order Message

Property Name	Type	Output Argument
Order Message	Output Argument	SiebelMessage

7. Set up the third step to use the EAI XML Converter with the IntObjHierToXMLDoc method.

   This step converts the integration object to a well-formed XML document using the following input and output arguments:

Input Argument	Type	Property Name
SiebelMessage	Process Property	Order Message

Property Name	Type	Output Argument
<Value>	Output Argument	<Value>

8. Save and deploy the workflow.
   For details on deploying workflows, see *Siebel Business Process Framework: Workflow Guide* .
9. Define a JMS Connection subsystem using SrvrMgr (command line utility) or the Server Administration screen.

   Note: Restart the Siebel Server to make the new subsystem available.

   Following is an example using SrvrMgr:

   Note: ConnectionFactory, ReceiveQueue and SendQueue require JNDI names, which varies depending upon the JMS provider and your implementation.

```
create named subsystem JMSToFromSiebel for subsystem JMSSubsys with
ConnectionFactory="jndiName.ConnectionFactory",
ReceiveQueue="jndiName.toSiebel ",
SendQueue="jndiName.fromSiebel",
ReceiveTimeout=3000
```

10. Define a data handling subsystem to dispatch the message from the toSiebel queue to the workflow as previously defined (JMS Query Order):

```
create named subsystem QueryOrder for subsystem EAITransportDataHandlingSubsys
with DispatchWorkflowProcess="JMS Query Order"
```

    Note: The Siebel Server must be restarted in order for the data handling subsystem to be available.

11. After restarting the Siebel Server, start a new JMS Receiver from the SrvrMgr command line.
    The following is an example that instructs the receiver to use the JMSToFromSiebel connection subsystem defined in Step 9, the QueryOrder data handling subsystem defined in Step 10, and instructs the receiver to use the ReceiveDispatchSend method of the EAI JMS Transport:

```
start task for comp JMSReceiver with
ReceiverConnectionSubsystem= JMSToFromSiebel,
ReceiverDataHandlingSubsystem=QueryOrder,
ReceiverMethodName=ReceiveDispatchSend
```

12. Place a message resembling the following on the toSiebel queue:

    Note: A third-party product such as Hermes (available from Sourceforge.net) is required to place a message on a queue. In the following sample document, the Siebel Order ASI queries for all orders associated with the Hibbings Manufacturing account.

```
<?xml version="1.0" encoding="UTF-16"?>
<SiebelMessage IntObjectName="Order Interface">
 <ListOfOrderInterface>
 <Orders>
 <Account>Hibbings Manufacturing</Account>
 </Orders>
 </ListOfOrderInterface>
</SiebelMessage>
```

Sending and Receiving Custom JMS Properties

Properties can be assigned to a JMS message. A property can be an instance of any Java class or any of the primitive Java types. All properties of a message received by the Siebel JMS Transport are available as properties of the output property set. The Siebel EAI infrastructure can send and receive custom JMS properties without having to write custom code.

The name of a custom property is the original name with the eleven characters `SIEBEL_JMS:` prefixed; the value is the string obtained by converting the original value to a Java String object. When sending a message, any property of the input property set whose name begins with `SIEBEL_JMS:` is added to the message being sent as a JMS Message string property with the prefix `SIEBEL_JMS:` removed. For example, the property `SIEBEL_JMS:foo` is added to the message as the string property `foo`.

Receiving Custom Properties in Inbound Messages

Inbound messages are received through the JMS Receiver component (ReceiveDispatchSend or ReceiveDispatch method). This component is usually configured to dispatch the message to a workflow process.

To receive a custom JMS property in a workflow process

1. Create a workflow process property as follows:

Name	Data Type	In/Out
SIEBEL_JMS:*name*	String	In/Out

   Note: There is no space between the colon and the custom property name.

2. Repeat the previous step for every custom JMS property that is expected to be received and processed.

   At run time, the Siebel EAI infrastructure automatically copies the value of the correct JMS property from the received message to the appropriate Workflow process property.

   For example, to have two JMS properties called TLFXUserId and TLFXGroupId available to a workflow process, you must define two process properties called SIEBEL_JMS:TLFXUserId and SIEBEL_JMS:TLFXGroupId.

   The workflow process can also set the values of the JMS properties using a step that calls the Workflow Utilities business service (Echo method) as shown in the following example:

Input Argument	Type	Value
SOV_Group	Expression	“SOV_Group”
SOV_User	Expression	“SOV_User”

Property Name	Type	Output Argument
SIEBEL_JMS:TLFXGroupId	Output Argument	SOV_Group
SIEBEL_JMS:TLFXUserId	Output Argument	SOV_User

An input argument (SOV_Group and SOV_User in the example) can be any string, with the requirement that the same string must be used as the output argument.

Because the process properties are defined as In/Out, they are passed back to the caller (the JMS Receiver in this case). The JMS Transport includes them in the output message as JMS properties.

For more information about creating workflow processes, see *Siebel Business Process Framework: Workflow Guide* .

Sending Custom Properties in Outbound Messages

In the standard application, outbound messages are sent to the JMS queue using the EAI JMS Transport business service (Send and SendReceive methods).

The standard BS though does not have the ability to set custom JMS properties, but it is extremely easy to create a new clone of the EAI JMS Transport BS to handle those.

To set custom JMS properties in outbound messages

1. In Siebel Tools, create and open a workspace.
2. Copy the EAI JMS Transport business service, then give the copy a new name and display name, such as My EAI JMS Clone.
3. In the new business service, add business service method arguments to the Send method as follows:

Name	Data Type	Type
SIEBEL_JMS:*name*	String	Input

4. Repeat the previous step for the SendReceive method, but enter Input / Output for the Type property.

Using Input / Output as the Type is necessary if the external system modifies the JMS properties and the new values are read into the Siebel application.

5. Deliver the workspace.

The new business service can be used in any workflow process. You can pick the custom JMS properties as input argument names when defining workflow steps, and the custom JMS properties are added to the JMS message. For more information about creating workflow processes, see *Siebel Business Process Framework: Workflow Guide* . For more information about business services, see *Integration Platform Technologies: Siebel Enterprise Application Integration* .

Enabling Authentication and Authorization for the EAI JMS Transport

Authentication and authorization can be configured on JMS servers to protect JMS destinations. Oracle supports the following scenarios for use in the Siebel application:

- Require username and password to perform a JNDI lookup.
- Require username and password to create connections to the JMS server.
- Require username and password to send, receive, publish, subscribe from, or subscribe to JMS destinations that have the authorization enforced by a JMS server.

The responsibility of the Siebel EAI JMS Transport business service as a JMS client is twofold:

- Provides configuration mechanism and read credentials from the Siebel application configuration file.
- Establishes proper security context for executing privileged operations.

The following authentication and authorization topics are also discussed:

- *About JMS Credential Specification*
- *Configuring Credentials in JNDI*
- *Configuring Credentials in JMS*
- *Configuring Against Oracle WebLogic Server*
- *Configuring Against TIBCO Enterprise Message Service*
- *Configuring Against IBM WebSphere MQ*
- *About Security Configuration on the JMS Server*

About JMS Credential Specification

The following method arguments are added to the EAI JMS Transport business service methods to use when completing the JMS credential specification:

- **ConnectionUsername and ConnectionPassword.** The credentials used to create JMS connections (applicable for use with IBM WebSphere MQ only, see *Configuring Against IBM WebSphere MQ*).
- **SendUsername and SendPassword.** The credentials used to send messages to SendQueue (applicable for use with Oracle WebLogic only, see *Configuring Against Oracle WebLogic Server*).
- **ReceiveUsername and ReceivePassword.** The credentials used to receive messages from ReceiveQueue (applicable for use with Oracle WebLogic only, see *Configuring Against Oracle WebLogic Server*).

- **TopicUsername and TopicPassword.** The credentials used to publish/subscribe to/from Topic (applicable for use with Oracle WebLogic only, see *Configuring Against Oracle WebLogic Server*).

Send and receive credentials are specified separately because some JMS business service methods (SendReceive and ReceiveDispatchSend) contain both send and receive operations, and it is possible that SendQueue and ReceiveQueue are protected by different credentials.

Configuring Credentials in JNDI

JNDI credentials are specified in the jndi.properties file by setting java.naming.security.principal to the username and java.naming.security.credentials to the password. For more details, refer to the JNDI specification. The construction of the naming context automatically reads the credentials from the jndi.properties file and uses those credentials to connect to a JNDI server if authentication is required to perform JNDI lookup.

For security reasons, Siebel CRM requires that the value of java.naming.security.credentials (representing the JNDI password) in the jndi.properties file be encrypted. The JNDIEncryptionCheck parameter in the JMSSubsys named subsystem is set to TRUE by default to enforce the encryption requirement. In this case, Siebel CRM decrypts the encrypted value of java.naming.security.credentials.

> **Note:** JNDIEncryptionCheck is TRUE in all newly created named subsystems based on JMSSubsys. For any older subsystems in which JNDIEncryptionCheck is not set to TRUE, the java.naming.security.credentials value in the jndi.properties file is not treated as an encrypted value.

Customers encrypt the value of java.naming.security.credentials in the jndi.properties file by using the following encryption utilities:

- EncryptJndi.sh, found in the `<SIEBEL_ROOT>/ses/siebsrvr/bin` folder on UNIX
- EncryptJndi.bat, found in the `<SIEBEL_ROOT>\ses\siebsrvr\bin` folder on Windows

Configuring Credentials in JMS

JMS-related credentials (those listed in the JMS credential specification) are passed in through a Siebel application-defined configuration mechanism. For configuring JMS-related credentials, see *Configuring the EAI JMS Transport*.

JMS Password Encryption

When passwords are provided through service input properties (ConnectionPassword, SendPassword, ReceivePassword, or TopicPassword), they are encrypted manually using the Siebel encryptstring utility. The EAI JMS Transport business service attempts to decrypt the password before using it. Passwords supplied using the name server have already been encrypted by the server manager; therefore, it is not necessary to encrypt it again with encryptstring.

> **Note:** The encryptstring utility is located in the `BIN` directory of your installation of the Siebel Server. For more information, see *Siebel Security Guide* .

Configuring Against Oracle WebLogic Server

The following instructions let you configure the EAI JMS Transport business service against Oracle WebLogic Server.

> **Note:** For detailed information relevant to client configuration tasks for similar products, such as Oracle SOA Suite, see vendor documentation from Oracle.

To configure the EAI JMS Transport business service against Oracle WebLogic Server

1. Authorize a user to send from SendQueue using SendUsername and SendPassword.
2. Authorize a user to receive from ReceiveQueue using ReceiveUsername and ReceivePassword.
3. Authorize a user to publish and subscribe to and from Topic using TopicUsername and TopicPassword.

   By default, the Oracle WebLogic server does not require a username or password to connect to or lookup JNDI objects. If the server does require this, then configure the EAI JMS Transport business service following Step 4 and Step 5.
4. ConnectionUsername and ConnectionPassword are set to a user who can connect to the JMS server, but the user has no privileges for any JMS destinations.

   ConnectionUsername and ConnectionPassword can also be left blank if the JMS server accepts anonymous connections.
5. If JNDI lookup is protected, then the jndi.properties file contains the java.naming.security.principal and the java.naming.security.credentials parameters that are used to perform the JNDI lookup.

   > **Note:** The JNDI principal and credentials are set to a user who can only perform the JNDI lookup, but has no privileges for any JMS destinations.

Configuring Against TIBCO Enterprise Message Service

For the TIBCO Enterprise Message Service (EMS) client, no separate security context is needed for each operation. Once a connection is established, with the proper credential, all requests sent through the same connection use the same connection security context. This means that switching the security context requires switching connections.

For the ReceiveDispatchSend method, the implication is that the receive credentials must be the same as the send credentials. Receive and send must be executed on the same session or connection to remain a single transaction.

To configure the EAI JMS Transport business service against TIBCO EMS

1. ConnectionUsername and ConnectionPassword are set to proper credentials for executing the JMS operations specified by the JMS business service method.

   For example, in the Send method, both ConnectionUsername and ConnectionPassword are set to the credentials that are authorized to send messages to SendQueue.

   In the ReceiveDispatchSend method, ConnectionUsername and ConnectionPassword are set to the credentials that can both send to SendQueue and receive from ReceiveQueue.
2. Set the following input properties to empty:

- SendUsername
- SendPassword
- ReceiveUsername
- ReceivePassword
- TopicUsername
- TopicPassword

3. The jndi.properties file contains the java.naming.security.principal and java.naming.security.credentials properties that are used to connect to the EMS server and to lookup JNDI objects.

   However, the connection to the EMS server, and the ability to lookup JNDI objects, does not occur if anonymous access is enabled by TIBCO EMS. For more information, see the TIBCO EMS documentation.

   Note: These JNDI credentials are set separately from ConnectionUsername and ConnectionPassword.

Configuring Against IBM WebSphere MQ

For the IBM WebSphere MQ client, no separate security context is needed for each operation. Once a connection is established, all requests sent through the same connection use the same connection context.

Note: The IBM WebSphere MQ server does not perform authentication by default. By default, passwords are not validated. Setup authentication for IBM WebSphere MQ is a task for the IBM WebSphere MQ administrator, not the Siebel application administrator.

For the ReceiveDispatchSend method, the implication is that the receive credentials must be the same as the send credentials. Receive and send must be executed on the same session or connection to remain a single transaction.

To configure the EAI JMS Transport business service against IBM WebSphere MQ

1. Set the ConnectionUsername and ConnectionPassword to the proper credentials to execute the JMS operations specified by the JMS business service method. For example, in the Send method, both ConnectionUsername and ConnectionPassword must be set to the credentials that are authorized to send messages to SendQueue.

   Note: ConnectionUsername and ConnectionPassword are required for the Windows 2012 Server platform and recommended for all other Windows platforms.

2. In the ReceiveDispatchSend method, set the ConnectionUsername and ConnectionPassword to the credentials that can both send to SendQueue and receive from ReceiveQueue.
3. Make sure the jndi.properties file contains the java.naming.security.principal and java.naming.security.credentials properties that are used to connect to the EMS server and to look up JNDI objects.

   Note: These JNDI credentials are set separately from ConnectionUsername and ConnectionPassword.

For more information about configuring the EAI JMS Transport business service against IBM WebSphere MQ, see 828113.1 (Article ID) on My Oracle Support.

About Security Configuration on the JMS Server

For information about how to protect JMS resources on the JMS server, see the specific vendor documentation.

Troubleshooting for the JMS Transport

Several diagnostic methods are present in the EAI JMS Transport to assist in troubleshooting CLASSPATH, JNDI, and problems connecting to the JMS server:

- **CheckClasspath.** Iterates through the JVM's classpath, checking for the existence of each directory in the file system.

  > **Note:** The length of the classpath is limited to 1024 characters. However, it might be truncated when displayed in the user interface and srvrmgr command-line interface. To see the entire classpath, examine the log file. For information about logging, see *About Logging for the JMS Transport*.

- **CheckJNDIContext.** Creates a JNDI InitialContext based on parameters (context factory class, URL) in the jndi.properties file.

  Lists the parameters and the entries found in the context, as well as the names and classes of the administered objects.

- **CheckJNDIObjects.** Retrieves administered objects (connection factory, queue, topic) from JNDI.

  If CheckJNDIObjects finishes without errors, then JNDI binding is proper.

  If CheckJNDIObjects finishes with errors, then it means that the JNDI binding has not been done properly. Rebind the JNDI objects or check the jndi.properties file to see if the provider URL is pointing to the correct location.

- **CheckJMSServer.** Invokes JMS methods directly and simply. If SendQueue is specified, then CheckJMSServer sends a message and then receives a message. If SendQueue is not specified and Topic is specified, then it creates a durable subscriber, publishes a message, receives it, and then unsubscribes.

  If CheckJMSServer finishes without errors, then both the queuing system and JMS are communicating properly.

  If CheckJMSServer finishes with errors, then it means that the JMS queue in the queuing system is not functioning properly. Check the corresponding queue in the queuing system.

- **CheckAll.** Executes all checks: CheckClasspath, CheckJNDIContext, CheckJNDIObjects, CheckJMSServer.

The following table contains more details on arguments used with some of the JMS Transport debugging methods. The arguments listed are used by all three methods.

Method	Argument	Display Name	Type	Description
CheckJNDIObjects CheckJMSServer	ConnectionFactory	Connection Factory	Input	JNDI name for the JMSConnectionFactory

Method	Argument	Display Name	Type	Description
CheckAll	SendQueue	Send Queue	Input	JNDI name for the queue (optional)
	Topic	Topic	Input	JNDI name of the topic (optional)

About Logging for the JMS Transport

The JMS Transport logs messages to a file if the Java system property jms.log is set. This property is specified among the VMOPTIONS in the JVM subsystem using the -Djms.log option.

The -Djms.log option must specify the path and file name but not the extension, because the JMS Transport automatically adds the .txt extension plus some information about the PID and thread ID.

For example, by using:

```
VMOPTIONS="-Djms.log=C:\temp\mylog"
```

The log file generated is:

```
C:\temp\mylog_xxx_yyy.txt
```

For more information about JMS logging, refer to the JMS vendor's documentation.

About Caching for the JMS Transport

JMS Receiver connections are cached in Siebel CRM. JNDI objects are also cached for performance and reliability. Caching eliminates the JNDI service as a point of failure.

JNDI object caching is active by default. To turn off caching (that is, to force JNDI lookup every time), use the DisableJNDIObjectCache business service method argument for any EAI JMS Transport business service method (operation). When DisableJNDIObjectCache is set to true, JNDI objects are not cached.

7 EAI HTTP Transport

EAI HTTP Transport

This chapter discusses EAI HTTP Transport, its methods, and workflow examples illustrating using EAI HTTP Transport with different methods. This chapter includes the following topics:

- *About the EAI HTTP Transport*
- *Using POST and GET*
- *EAI HTTP Transport Named Subsystems*
- *EAI HTTP Transport Method Arguments*
- *Sending a Message Using the EAI HTTP Transport*
- *Using the EAI HTTP Transport for Inbound Integration*
- *Process of Using the EAI HTTP Transport for Inbound Messages*
- *Handling EAI HTTP Transport Business Service Errors*
- *Processing and Sending Outbound XML Documents*
- *Sending and Receiving Messages with the EAI HTTP Transport*
- *Examples Using HTTP Request*
- *Creating Custom Headers for the EAI HTTP Transport Service*
- *About Sending and Receiving Messages Through HTTP*
- *About Transport Headers and HTTP Response Headers*

About the EAI HTTP Transport

The use of the Internet protocols and technologies for business (such as HTTP, HTML, and XML) has created a requirement for automatically sending Siebel data to external sites, either on the Internet or outside the enterprise firewall to external Web sites. To meet this need, the technologies built into Siebel EAI provide a way to send and receive messages over HTTP. Siebel EAI HTTP Transport business service lets you send XML messages over HTTP to a target URL (Web site). The Siebel Application Interface (AI) serves as the transport to receive XML messages sent over the HTTP protocol to a Siebel application.

The EAI HTTP Transport business service is based on the CSSHTTPTransService class. You can use one of the following two methods with this transport:

- **Send.** This method supports outbound messages (XML documents sent from a Siebel application to an external system). The Send method means that the response coming back from the external application is not interpreted by the Siebel application, but the Web server returns a correct HTTP response.
- **SendReceive.** This method supports outbound messages (XML documents sent to a Siebel application from an external system). This method is called *Send and Receive a Response* and the HTTP response body is the response for the request.

Each method has its own arguments, techniques, and applications. The EAI HTTP Transport allows you to send messages across the Internet using the standard HTTP protocol. Using this transport, you can send messages to any

URL. The XML document sent can then be acted upon by any Web-based application, including those written in Java, JavaScript, VBScript, or any other Web-enabled technology.

Note: When using the EAI HTTP Transport with the Transport Layer Security (TLS) protocol, you might have to install certificates on the Siebel Server. For more information, see *Siebel Security Guide* .

System Requirements for Using the EAI HTTP Transport

To use the EAI HTTP Transport, you must install and configure the following components of Siebel CRM, and make sure that they are operational:

- **Siebel Application Interface (AI).** To provide the necessary HTTP listening services and invoke the requisite workflow through a business service method.
- **Workflows.** To accept incoming XML documents and pass them through an integration object into the business object to update Siebel data.
- **Business services.** To execute the necessary actions.

Selecting the Appropriate Business Service for HTTP

The business service required to process a given XML document that is received from an external system using the EAI HTTP Transport depends on the processing you perform on the data. The way to approach this is to accept the output of the EAI HTTP Transport and store it as a process property that you define, and process it later in the workflow based on the format of the data.

For example, you could pass the string into a custom business service that you build to parse the input, query some data in a Siebel application based on the data, and then update the appropriate field in the Siebel application. If the data is formatted as a SiebelMessage, then you could use the EAI XML Converter business service with the XMLDocToIntObjHier method to pass an integration object instance to the EAI Siebel Adapter for further processing.

Note: Do not use the Web Engine HTTP TXN business service for inbound HTTP transport sessions. This business service is intended only for Siebel user interface sessions in the Siebel Web Client or Siebel Mobile Web Client. It is not compatible with invocation from the EAI Application Object Manager task. For information about the Web Engine HTTP TXN business service, see *Siebel Portal Framework Guide* .

Using POST and GET

The HTTP protocol supports the GET and POST methods. You might be familiar with these methods if you have ever built a Web-based CGI form:

- **GET.** Requests a representation of the specified resource. GET is the most common method used on the Web today.
- **POST.** Submits data to be processed, such as from an HTML form, to the identified resource. The data is included in the body of the request. This might result in the creation of a new resource, updates to existing resources, or both.

The EAI HTTP Transport imposes certain restrictions on your use of transport features when using the POST or GET method. The following table identifies restrictions on these HTTP methods.

Method	Restriction
Get	The HTTP Body has no significance when using GET. During a GET process, only the universal resource locator (URL) is used for the request. **Note:** Passing user credentials in the URL is not supported in Siebel CRM.
Post	The HTTP Body is relevant only when using POST. The HTTP Body is encoded with a default mechanism used to encode URLs. The HTTP Content-Type **`application/xxx-form-urlencoded`** is the default content type used for request bodies. The content is sent as it is without any special content encoding, such as Base64.

EAI HTTP Transport Named Subsystems

The EAI HTTP Transport, like every other Siebel transport, reads required parameters from a named subsystem instead of the configuration (.cfg) file. The eai.cfg file entries list the external service name and the name of the named subsystem to be used. For example:

```
SiebelQuery = SiebelQueryDispatch
```

There is no [Properties] section for SiebelQueryDispatch in the .cfg file. The name is used to look up the named subsystem list and dispatch accordingly. Use named subsystems for property specification. Predefined named subsystems have been created for you already, such as:

- SiebelQueryDispatch
- SiebelExecuteDispatch
- SiebelUpsertDispatch

Note: You can create additional named subsystems as needed using Siebel Server Manager.

For a discussion of named subsystems for Siebel EAI, see *EAI Transports and Interfaces Overview*. For more information about named subsystems, see *Siebel System Administration Guide* .

EAI HTTP Transport Method Arguments

In addition to the method arguments (data handling parameters) in *Common EAI Transport Parameters*, EAI HTTP Transport methods take the arguments presented in the following table. Parameters are optional unless specified as required.

Parameter	Display Name	Description
`<Value>`	*User-Defined Message Text*	Input and Output data passed as a string. This is the value stored in the Value field of the property set, either input or output. If you specify the HTTPRequestBodyTemplate, then the `<Value>` parameter is ignored and the HTTPRequestBodyTemplate parameter is used instead.
CharSetConversion	Character Set Conversion for Text Data	Character set conversion from the external system. The default is None.
ConnectionSubsystem	Connection Subsystem	Subsystem containing connection parameters.
ConverterService	Converter Service	Business service used to serialize and unserialize hierarchical data to raw buffer and the reverse. Must implement the DocToHier and HierToDoc methods. The default is EAI XML Converter.
DataHandlingSubsystem	Data Handling Subsystem	Subsystem containing data handling parameters.
EndOfData	End of Data	Output parameter whose value is True if the end of the data has been reached.
HTTPAccept	HTTP Accept	Default is `text/*`. The explicit value for the Accept: header to override the default. Specifies the MIME types accepted by the sender.
HTTPAllowCaching	Allow Caching	Default is N. By default, the responses for specific URL addresses are not cached by the EAI HTTP Transport. Set this flag to Y to enable caching. Note that this can lead to undesirable side effects, as old data from earlier requests can be exposed from the cache buffer.
HTTPAllowPersistentCookies	Allow Persistent Cookies	Default is N. A session cookie is used to tie requests and logoff operations to the user session started at the login, when communicating with any session-cookie-based system. Leaving this flag set to N leaves the persistence of cookies in the control of the EAI HTTP transport, which is the default behavior. All session cookies persist in memory only as long as the current session. Session cookies are not written to disk. If you want to use persistent cookies, that is, if persistence between logins is required and you want cookies to be written to disk, then set the parameter to Y.
HTTPCertAuthority	HTTP Cert Authority	The name of the authority that issues the mutual authentication certificate, in RDN (Relative Distinguished Name) format. For example: `CN=ServerName123, OU=Department,`

Parameter	Display Name	Description
		O=organization, L=Location, C=Country, E=email@example.com represents a certificate issued by Microsoft Certificate Authority running on the server ServerName123. RDN notation is case insensitive. For information about configuring client TLS authentication, see *Siebel Security Guide* .
HTTPCertSerialNo	HTTP Cert Serial No	The mutual authentication certificate serial number, in hexadecimal format as a string without space characters in between. For example, the serial number **"19 8b 11 d1 3f 9a 8f fe 69 a0"** must be provided as: **198b11d13f9a8ffe69a0** Serial numbers are case insensitive. For information about configuring client TLS authentication, see *Siebel Security Guide* .
HTTPContentType	HTTP Content Type	Default is **application/xxx-form-urlencoded**. The explicit value for the Content-Type: header to override the default. Specifies the type of data sent in the body of the request.
HTTPImplicitCharsetDetection	Implicit Character Set Detection	Default is False. This is the implicit character set detection for incoming data. Do not set it to True for self-describing documents such as XML. If set to True, then this overrides the CharSetConversion parameter.
HTTPLoginBodyTemplate	Login Body Template	Specifies the HTTP request body that is used when HTTPLoginURLMethod is POST. By putting login information into the HTTP body (as opposed to putting it into the URL) for sending, this method provides stronger security than sending the login information in the URL. Generally, the login parameters in a login query are specified in the body of the request that uses the POST method. Required for session mode only if the HTTPLoginMethod parameter is set to POST.
HTTPLoginMethod	Login Method	HTTP method to be used for logging in. If no Login Method is specified, then this parameter defaults to the HTTPRequestMethod value. Required for session mode.
HTTPLoginURLTemplate	Login URL Template	Template for the URL used for the login operation. This operation is separate from the request operation and assumes communication mode is session mode. If there is a separate login, then one or more request and response messages are expected. Required for session mode.

Parameter	Display Name	Description
HTTPLogoffMethod	Log Off Method	Defaults is HTTPLoginMethod. HTTP method to be used for logging off. Required for session mode.
HTTPLogoffURLTemplate	Log Off URL Template	Template for the URL that is used for the logoff operation. This operation is separate from the request operation and assumes that the mode of communication is session mode. If it is set, then the logoff operation is completed. Otherwise, logoff is skipped. The purpose of the logoff operation is to end a session that was started with the corresponding login. Required for session mode.
HTTPMaxIdleSeconds	Max Idle Seconds	Maximum number of seconds to allow connections to be idle. After the elapsed max idle time, the connection is invalidated and restarted.
HTTPNoAutoRedirect	No Auto Redirect	Default is N. This means auto-redirect is enabled. Setting this parameter to Y disables auto-redirection of messages to other URLs.
HTTPRequestBodyTemplate	Request Body Template	HTTP Body to use with the POST method. This overrides any request body specified in the Value field of the input property set.
HTTPRequestMethod	Request Method	HTTP method to use with the data request, such as POST or GET. Required for both session and sessionless modes.
HTTPRequestURLTemplate	Request URL Template	Template for the request URL, which is the address to which the data is sent or from which a response is requested. Required for both session and sessionless modes.
HTTPSleepTime	Sleep Time	Default is 120000 milliseconds. The timeout interval on login, send, and logoff requests in milliseconds.
HTTPUserAgent	HTTP User Agent	Default is `Mozilla/4.0`. The explicit value for the User-Agent: header to override the default. Specifies the name/version of the client program.
IgnoreCharSetConvErrors	Ignore Character Set Conversion Errors	Ignore character set conversion errors if True. Else, propagate the errors to the caller (default behavior).
TimedOut	Timed Out	True if receive timed out and no data was available. False if request completed.

Sending a Message Using the EAI HTTP Transport

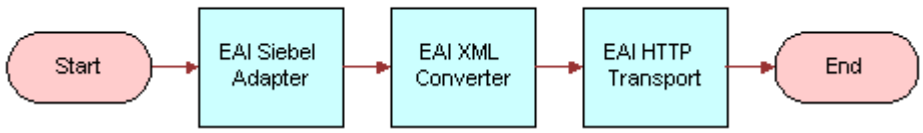
The following procedure demonstrates how to send information from a Siebel application to another Web-based application using the EAI HTTP Transport.

To send a message

1. Create an integration object in Siebel Tools based on a given business object.
2. Refine the integration object created in the previous step to specify just those business components and fields that you want to exchange with the external application.

   Note: For details about integration objects, see *Integration Platform Technologies: Siebel Enterprise Application Integration* .

3. Set up a workflow in Siebel Tools containing the following steps, as shown in the following image, to send this information to an external system:
   a. EAI Siebel Adapter
   b. EAI XML Converter
   c. EAI HTTP Transport



Note: For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide* .

   a. Create the following process properties:

Name	Data Type	In/Out	Value
Account Message	Integration Object	In/Out	Not applicable
Account XML	Binary	In/Out	Not applicable
Error Code	String	In/Out	Not applicable
Error Message	String	In/Out	Not applicable
Object Id	String	In/Out	Row Id of an account

Name	Data Type	In/Out	Value
Siebel Operation Object Id	String	In/Out	Not applicable

b. Set up the first step of the workflow after Start to use the EAI Siebel Adapter with the Query method to query the information from the Siebel Database, using the following input and output arguments:

Input Argument	Type	Value	Property Name	Property Data Type
OutputIntObjectName	Literal	Sample Account	Not applicable	Not applicable
PrimaryRowId	Process Property	Not applicable	Object Id	String

Property Name	Type	Output Argument
Account Message	Output Argument	SiebelMessage

c. Set up the second step to use the EAI XML Converter with the IntObjHierToXMLDoc method to convert the data extracted from the Siebel Database to XML format, using the following input and output arguments:

Input Argument	Type	Property Name	Property Data Type
SiebelMessage	Process Property	Account Message	Integration Object

Property Name	Type	Output Argument
Account XML	Output Argument	<Value>

d. Set up the third step to use the EAI HTTP Transport with the Send method to send the information to the external system, using the following input and output arguments:

Input Argument	Type	Value	Property Name	Property Data Type
<Value>	Process Property	Not applicable	Account XML	String

Input Argument	Type	Value	Property Name	Property Data Type
HTTPRequestMethod	Literal	POST	Not applicable	Not applicable
HTTPRequestURLTemplate	Literal	http://$web_address$/$request_param$	Not applicable	Not applicable

Property Name	Type	Output Argument
Account XML	Output Argument	<Value>

e. Save the workflow and run it from the Workflow Simulator.

4. Specify how this workflow is invoked, using one of the following methods:
   - Configure the RunTime Events to trigger the workflow.
   - Create a button on the appropriate view in the Siebel application to call this workflow.
   - Use workflow policies on the opportunity business object to trigger the workflow.

Using the EAI HTTP Transport for Inbound Integration

The EAI HTTP Transport uses the Siebel Application Interface (AI) to provide inbound messaging from an application that uses HTTP. The EAI HTTP Transport can be used in session or sessionless mode.

The following topics are discussed:

- *Preparing to Use the EAI HTTP Transport for Inbound Integration*
- *Specifying HTTP Parameters for Inbound Integration*
- *Using the EAI HTTP Transport in Session Mode*
- *Using the EAI HTTP Transport in Sessionless Mode*

Preparing to Use the EAI HTTP Transport for Inbound Integration

To use the EAI HTTP Transport for inbound integration, you must perform certain tasks that might not be required when using the EAI HTTP Transport for outbound integration:

1. Install and configure the Siebel Application Interface (AI), Siebel Gateway, and Siebel Server.
2. Start the Siebel Application Interface (AI), Siebel Gateway, and Siebel Server.
3. Start the Siebel Application Interface (AI) to be able to use the EAI HTTP Transport.
4. Configure AI to run the EAI HTTP Transport for inbound integration. See *Specifying HTTP Parameters for Inbound Integration*.
5. Set certain configuration parameters for whatever Siebel Server you are using.

The server component you are running must be a Siebel Application Object Manager component.

Note: You can type `http://Web_Server_Name/siebel/app/eai-/lang` in a Web browser on any computer that has connectivity to the Application Interface to check the connectivity between the computer issuing the URL (for the EAI HTTP Transport) and AI. This URL brings up the login page of the Siebel application corresponding to ObjectManager_lang, confirming the connectivity between AI and the URL-issuing computers.

Specifying HTTP Parameters for Inbound Integration

The EAI HTTP Transport is built into Siebel Application Interface (AI). To use it, you set certain configuration parameters of the AI profile of the application interface. Your Siebel application installation includes a configuration file called AI profile. Review the configuration file to make sure that the parameters are set properly. Use named subsystems to dispatch to a workflow as described in *Using Named Subsystems for Transport Parameters*.

To configure AI to run the EAI HTTP Transport for inbound integration

Note: For instructions on how to create or modify the Application Interface (AI) profile, see *Siebel Installation Guide* .

1. Launch Siebel Management Console (SMC).
2. Open the AI profile deployed to AI.
3. Look for the application `eai (lang)`. Where lang is the three-letter language code for the language you are using, such as enu for U.S. English.

   If this application does not exist then add one with name as eai and Object Manager as `EAIObjmgr_lang` .
4. In the Basic Information section of the `eai (lang)` application, select Configure EAI HTTP Inbound Transport parameter to enable the HTTP inbound transport.
5. Submit the AI profile.

Using the EAI HTTP Transport in Session Mode

The session mode uses the HTTP session cookie to retain the session information between the HTTP requests. The session mode can be viewed when a sequence of calls is supported from an HTTP application into the EAI HTTP Transport.

To use the EAI HTTP Transport in session mode

1. Log in to the Siebel application. If successful, then an HTTP session cookie named _sn is returned in an HTTP set-cookie header.
2. Submit one or more subsequent requests.

   Each request is intended as a call to a Siebel business service or workflow depending on the configuration of the named subsystem in use. Requests must contain the session cookie (_sn) from the previous step in either the HTTP cookie header or the URL string as a parameter.
3. Log off. The request must contain the session cookie from Step 1. The cookie refers to the session to be closed.

Note: For session mode inbound HTTP requests, the expiration date of the cookie sent to the client application is not set, because it is expected that this cookie is used to send multiple requests within the same session.

Example Requests for the HTTP Protocol in Session Mode

HTTP protocol requests can be represented as URLs for HTTP GET, and as a combination of URL and request body for HTTP POST. The following topics explain in detail how each of the session mode calls is configured.

The following table describes each of the Login HTTP Request variables for session mode.

Variable	Description
webserver	URL of the Web server that has Siebel Application Interface (AI) installed, such as www.myserver.com.
path	Virtual path on the server referring to the specific AI profile configuration. The default is `/siebel/app/eai/lang` , where lang is the language in which you are running the applicable Siebel Application Object Manager.
source	Named subsystem as specified in the `[HTTP Services]` section in the application configuration (.cfg) file.
username	Siebel user name for the Application Object Manager login. **Note:** Passing user credentials in the URL is not supported in Siebel CRM.
password	Password for the login user name.

Login HTTP Request Example

In this example, if the call completes successfully, then it returns a session cookie:

- Using HTTP POST:

```
URL = http://webserver/path
HTTP Body =
SWEExtSource=source&SWEExtCmd=ExecuteLogin&UserName=username&Password=password
```

Example Login URL:

```
http://www.example.com/siebel/app/eai/enu
```

Note: Passing user credentials in the URL is not supported in Siebel CRM.

Data Exchange HTTP Request Example

In this example, for the call to complete successfully, it must include the session cookie from the login:

- Using HTTP GET:

```
URL = http://webserver/path?SWEExtData=data text
```

where data text is the business service input data. Most of the time, this is the text of an XML document that on the server side is converted to a property set and passed to the business service.

With GET requests, the XML document is included in the URL. Therefore the XML document must be URL-encoded. For example, the URL encoding for a space is %20.

To make sure that the decoded XML document passed to the XML Converter is valid, use an escape code for any special characters (that is, use an ampersand, followed by the special character's escape characters, followed by a semi-colon) before encoding them for the URL. For more information, see the topic on special (escape) characters in *XML Reference: Siebel Enterprise Application Integration*

- Using HTTP POST:

```
URL = http://webserver/path
HTTP Body = data text
```

where data text is the business service input data. Most of the time, this is the text of an XML document that on the server side is converted to a PropertySet and passed to the business service.

Data that is sent as part of the URL must be in Unicode format before it is encoded for the URL. POST requests can send the data without URL encoding but must include the Content-Type HTTP header. The Content-Type must specify the character set of the incoming data, for example:

```
Content-Type=text/xml;charset="UTF-8"
```

> **Note:** For XML messages being received by way of the Inbound HTTP Transport, only a Unicode (UTF-8 or UTF-16) format (with accordant encoding XML-processing header attribute and encoded XML data) is allowed. No ISO or Windows code pages are accepted.

- Example Request URL:

```
http://www.exampleserver.com/siebel/app/eai/enu?SWEExtData=<?xml version="1.0"
encoding="UTF-8"?>

<SiebelMessage MessageId="" MessageType="Integration Object"
IntObjectName="Sample Account">

<ListofSampleAccount>

 <Account>
 <Name>A. K. Parker Distribution</Name>
 <ListOfContact>
 <Contact>
 <FirstName>Stan</FirstName>
 <LastName>Graner</LastName>
 </Contact>
 </ListOfContact>
 </Account>

</ListofSampleAccount>
```

```
</SiebelMessage>
```

Logoff HTTP Request

This request must include the session cookie from the login request.

- Using HTTP GET:

  ```
  URL = http://webserver/path?SWEExtCmd=Logoff
  ```

  Note: Always use HTTP GET for the Logoff HTTP Request.

- Example Logoff URL:

  ```
  http://www.example.com/siebel/app/eai/enu?SWEExtCmd=Logoff
  ```

Using the EAI HTTP Transport in Sessionless Mode

Using the EAI HTTP Transport in sessionless mode allows you to use one URL to perform Login, Request, and Logoff in a single HTTP request. This mode does not use session cookies because there is no login session between the HTTP requests. The disadvantage of this mode is the overhead incurred by the Application Object Manager needing to log in with every request.

The following table describes each of the variables for sessionless mode.

Variable	Description
webserver	URL of the Web server that has Siebel Application Interface (AI) installed, such as www.myserver.com.
path	Virtual path on the server referring to the specific AI profile configuration. The default is `siebel/app/eai/lang` , where lang is the language in which you are running the applicable Siebel Application Object Manager.
source	Named subsystem as specified in the `[HTTP Services]` section in the application configuration (.cfg) file.
username	Siebel user name for the Siebel Application Object Manager login. **Note:** Passing user credentials in the URL is not supported in Siebel CRM.
password	Password for the login user name.
data text	Business service input data. Most of the time, this is the text of an XML document that on the server side is converted to a PropertySet and passed to the business service. For more information about how to pass Properties and PropertySet to Business Services, see *Siebel Business Process Framework: Workflow Guide* .

Example Request for the HTTP Protocol in Sessionless Mode

In this example using HTTP POST, the URL describes the parameters for the HTTP Inbound Transport call over HTTP. Unlike session mode, the SWEExtCmd is Execute, not ExecuteLogin.

```
URL = http://webserver/path

HTTP Body = SWEExtSource=source&SWEExtCmd=Execute&UserName=username&Password=password&SWEExtData=data text
```

Note: When using sessionless mode with the POST method, the XML data text must be URL-encoded to prevent any errors.

When using the sessionless mode with the POST method, the data text includes the login credentials as well as the XML document. Therefore, it is recommended that the data text be URL-encoded and that the Content-Type header be set to `application/x-www-form-urlencoded` without specifying the character set (for example, `;charset=UTF-8`).

Use an escape code for any special characters (that is, use an ampersand, followed by the special character's escape characters, followed by a semi-colon) before encoding them for the URL. For more information, see the topic on special (escape) characters in *XML Reference: Siebel Enterprise Application Integration* .

Example for Sessionless Mode

```
URL = http://www.example.com/siebel/app/eai/enu

HTTP Body =
SWEExtSource=SiebelQuery&SWEExtCmd=Execute&UserName=user1&Password=login123
&SWEExtData=<?xml version="1.0" encoding="UTF-8"?>

<SiebelMessage MessageId="" MessageType="Integration Object" IntObjectName="Sample
Account">

<ListofSampleAccount>
 <Account>
 <Name>A. K. Parker Distribution</Name>
 <ListOfContact>
 <Contact>
 <FirstName>Stan</FirstName>
 <LastName>Graner</LastName>
 </Contact>
 </ListOfContact>
 </Account>
</ListofSampleAccount>

</SiebelMessage>
```

Process of Using the EAI HTTP Transport for Inbound Messages

To use the EAI HTTP Transport for inbound messages, you complete two tasks:

1. *Setting Up the Business Service*
2. *Creating the Workflow to Receive Messages*

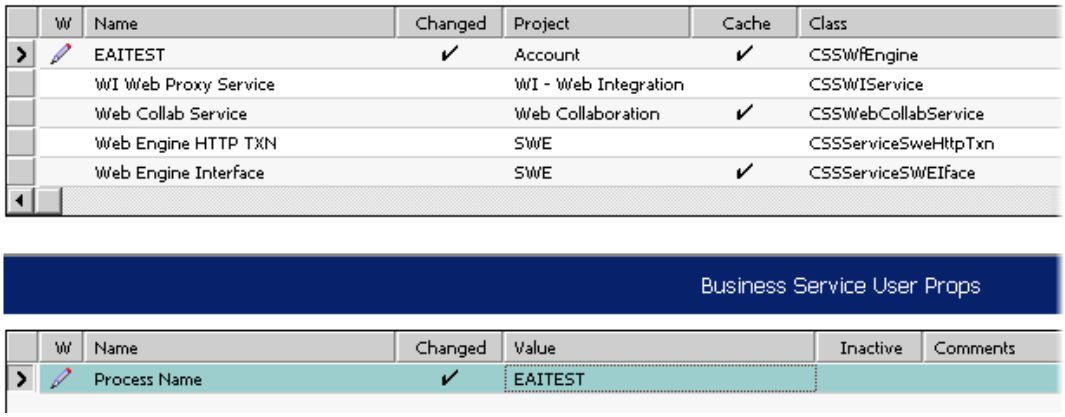
Both tasks are explained in this topic. This scenario assumes incoming XML. Your business requirements dictate whether and how you adapt these steps to fit your needs.

Setting Up the Business Service

First you set up the business service for use in the workflow.

To set up the business service

1. Start Siebel Tools, connecting to the server.
2. Create or open a workspace.
3. Find the business service named Workflow Process Manager.
4. Copy this record and rename the copy EAITEST.
5. In the Business Service User Props list, add a new record as shown in the following image:
   a. Enter ProcessName in the Name column.
   b. Enter EAITEST in the Value column.



W	Name	Changed	Project	Cache	Class
>	EAITEST	✔	Account	✔	CSSWfEngine
	WI Web Proxy Service		WI - Web Integration		CSSWIService
	Web Collab Service		Web Collaboration	✔	CSSWebCollabService
	Web Engine HTTP TXN		SWE		CSSServiceSweHttpTxn
	Web Engine Interface		SWE	✔	CSSServiceSWEIface

Business Service User Props

W	Name	Changed	Value	Inactive	Comments
>	Process Name	✔	EAITEST		

6. Deliver the workspace.

   Note: You can also deploy the business service to the run-time database to make it available. For more information, see *Integration Platform Technologies: Siebel Enterprise Application Integration* .

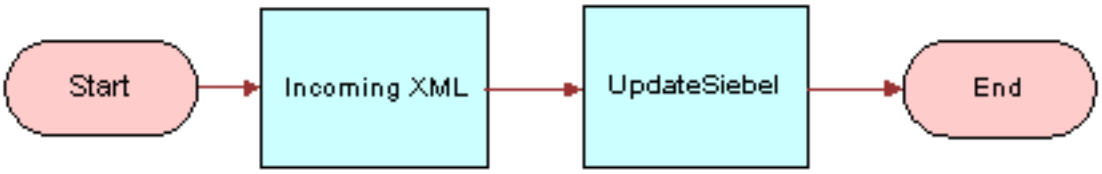
7. Restart the Siebel Server.
8. Verify that the EAI Object Manager has started.

Creating the Workflow to Receive Messages

After you set up the business service, you create a workflow to receive messages.

To create the new workflow to receive messages

1. Set up a new workflow in Siebel Tools containing the following steps, as shown in the following image, and give it a unique name such as EAITEST:
   a. Incoming XML
   b. UpdateSiebel



For information about the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide* .

2. Create the following process properties:

Name	Data Type	Default String	In/Out	Description
IncomingXML	Binary	<Value>	In/Out	By creating the IncomingXML process property, anything that is sent as data is placed in this variable. This allows you to then perform a given action on that data. If the POST method was used, then the data sent in the Body is stored in this property. If the GET method was used, then the data sent in the URL is stored in this property.
Account Message	Hierarchy	Not applicable	In/Out	This is hierarchy format of the incoming XML.
<Value>	Binary	Not applicable	In/Out	Used to get the XML string that has been read or converted.
Content-Type	String	text/html	Out	It indicates the content type of the response body. If you want to see the response in the same Web page, then you must set the Default String parameter to text/html.

3. Set up the Incoming XML step to use the EAI XML Converter with the XMLDocToIntObjHier method. This step converts the message, using the following input and output arguments:

Input Argument	Type	Property Name	Property Data Type
<Value>	Process Property	IncomingXML	Hierarchy

Property Name	Type	Output Argument
Account Message	Output Argument	SiebelMessage

4. Set up the UpdateSiebel step to use the EAI Siebel Adapter with the Insert or the Update method and the following input and output arguments to update the Siebel Database.

Input Argument	Type	Property Name	Property Data Type
SiebelMessage	Process Property	Account Message	Hierarchy

Property Name	Type	Value	Output Argument
<Value>	Literal	Update Completed	Not applicable

Note: The HTTP response for inbound requests is determined by looking at the `<Value>` portion of the output property set. HTTP response headers can be set by setting properties on the output property set. If the process properties are set as In/Out (the default), then the values appear as HTTP headers on the HTTP response from the Siebel Server. Set each process property that you do not want as an HTTP header to In or None (the latter if the process property is only for use inside the workflow).

5. Save your workflow and test it using the Workflow Simulator.

   For information about the Workflow Simulator, see *Siebel Business Process Framework: Workflow Guide* .

Handling EAI HTTP Transport Business Service Errors

A business service that is called by the EAI HTTP Transport might return an error when standard HTTP headers are used to send error information back to the caller. Each of the headers has a sequence number at the end to support the return of multiple errors. The text of each error message is captured in the Siebel-Error-Message header, and the Siebel error symbol is set in the Siebel-Error-Symbol header as follows:

```
Siebel-Error-Message-1: Error: error message text
Siebel-Error-Symbol-1: ERR_SYMBOL
...
Siebel-Error-Message-n:
Siebel-Error-Symbol-n:
```

Inbound HTTP also returns HTTP Error 500 (Internal Server Error) to indicate that there was an error from a business service. Examine the error headers for additional error information.

Note: To troubleshoot an Inbound HTTP request, run the Siebel Workflow Simulator or Business Service Simulator. For information about the Workflow Simulator, see *Siebel Business Process Framework: Workflow Guide* . For information about the Business Service Simulator, see *Integration Platform Technologies: Siebel Enterprise Application Integration* .

Processing and Sending Outbound XML Documents

This topic explains how to use Siebel Tools and the Siebel application to set up the EAI HTTP Transport to process and send outbound XML documents. When you want to send XML messages based on Siebel integration objects to an external system across Internet-support protocols, you use the EAI HTTP Transport business service.

You can specify the parameters that control the behavior of transports in the following ways:

- *Specifying Parameters as Business Service User Properties*
- *Specifying Parameters as Subsystem Parameters*
- *About Parameters as Run-Time Properties*
- *About Parameters in Parameter Templates*

Specifying Parameters as Business Service User Properties

You specify parameters as business service user properties in Siebel Tools. These parameters go into effect after you have delivered the changes or deployed the business service to the run-time database. When using this method, keep the following in mind:

- These parameters stay in effect as long as you continue to use the same run-time business service and do not create a newer specification for the business service parameters.
- If you define the same parameter as a subsystem parameter or as a run-time property, then the subsystem parameter or run-time property overrides any values you have defined in Siebel Tools and delivered or deployed to the run-time database.

For more information about deploying business services to the run-time database, see *Integration Platform Technologies: Siebel Enterprise Application Integration* .

Specifying Parameters as Subsystem Parameters

You specify parameters in the Siebel client.

To specify the subsystem parameters

1. In the Siebel client, navigate to the Administration - Server Configuration screen, Enterprises view.
2. In the first list applet, select the Enterprise Server that you want to configure.
3. In the middle applet, click the Profile Configuration tab.
4. Click New to create a new component profile, then set the following parameters:

Name	Value
Profile	HTTP_test
Alias	HTTP_test

Name	Value
Subsystem Type	HTTPSubSys

5. In the Profile Parameters list applet (the last applet), specify the parameters required for the type of operations the subsystem supports:

Name	Value
HTTPRequestURLTemplate	"http://www.example.com"
HTTPRequestMethod	"GET"

Then, in the workflow on the Siebel Web Client, you specify the Connection Subsystem input argument to the HTTP Transport, and the value is the named subsystem that you created. For the case given here, it is HTTP_test. You can test the workflow in the Workflow Simulator.

About Parameters as Run-Time Properties

You specify HTTP parameters as run-time properties by passing them as values in an input property set to the EAI HTTP Transport business service. You can pass the values to the business service by way of a workflow or through a program that calls the EAI HTTP Transport business service directly.

Note: Subsystem parameters take precedence over run-time parameters.

About Parameters in Parameter Templates

Parameter templates allow you more flexibility in specifying parameters. You can use variables to specify certain elements of a given parameter value. The following example shows how to specify a variable for a login password, rather than hard-coding a password into the parameter.

```
HTTPLoginURLTemplate = http://www.example.com/
login.jsp?Username=ronw&Password=$PWD$
```

where

PWD is 421ax7 (for example)

The business service, EAI HTTP Transport in this case, receives the parameter template. The token, shown here as PWD, indicates that the business service looks for a parameter called PWD from a user property or run-time parameter. Dollar signs ($) delimit the token in the template definition. The token specifies the actual password variable. The token is case-sensitive: Pwd is different from PWD or pwd.

The token must be defined as either a business service user property or as a run-time parameter in the input property set. For example, you could specify the HTTPLoginURLTemplate as a user property of the business service, and *username* and *password* as run-time properties. Any logins that specify the template always use the same template, but different users can specify unique user names and passwords at run time.

Sending and Receiving Messages with the EAI HTTP Transport

You can use the EAI HTTP Transport to send and receive messages. The following procedure illustrates how you can use EAI HTTP Transport with the SendReceive method to query employee information from the Siebel Database, send it out, echo it using the Workflow Utilities ECHO service, and send it back to the workflow to write the response back to a file.

To create a workflow to send and receive messages

1. Create a named subsystem HTTPsendreceive_conn for subsystem HTTPSubSys using the following lines:

```
HTTPLoginMethod=GET

HTTPLoginURLTemplate="http://websrvr.example.com:16007/myapplication/
login.jsp?usr=V1&psw=v2"

HTTPLogoffMethod=GET

HTTPLogoffURLTemplate="http://websrvr.example.com:16007/myapplication/
logoff.jsp"

HTTPRequestMethod=POST

HTTPRequestURLTemplate="http://websrvr.example.com:16007/myapplication/
data.jsp"
```

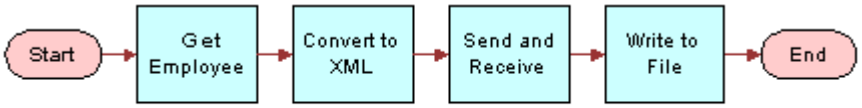
2. Create a named subsystem MyEchoSubsys for subsystem EAITransportDataHandlingSubsys using the following lines:

```
DispatchService="Workflow Utilities"
DispatchMethod=ECHO
```

3. In your eai.cfg file, add the following line in the [HTTP Services] section:

```
MyEcho = MyEchoSubsys
```

4. Set up a new workflow in Siebel Tools containing the following steps, as shown in the following image:
   a. Get Employee
   b. Convert to XML
   c. Send and Receive
   d. Write to File



> **Note:** For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide* .

5. Create the following process properties:

Name	Data Type	In/Out
Employee Message	Hierarchy	In/Out
Employee XML	Binary	In/Out
Error Code	String	In/Out
Error Message	String	In/Out
Object Id	String	In/Out
Response	Binary	In/Out

6. Retrieve the employee message using the EAI Siebel Adapter with the Query method to query the information from the database using the following input and output arguments.

Input Argument	Type	Value	Property Name	Property Data Type
OutputIntObjectName	Literal	Sample Employee	Not applicable	Not applicable
PrimaryRowId	Process Property	Not applicable	Object Id	String

Property Name	Type	Output Argument
Employee Message	Output Argument	SiebelMessage

7. Convert the message to XML using the EAI XML Converter with the Integration Object Hierarchy to XML Document method and the following input and output arguments to convert the message.

Input Argument	Type	Property Name	Property Data Type
SiebelMessage	Process Property	Employee Message	Hierarchy

Property Name	Type	Output Argument
Employee XML	Output Argument	<Value>

8. Send and receive the converted XML message using the EAI HTTP Transport with the Send and Receive Response method and the following input and output arguments.

Input Argument	Type	Value	Property Name	Property Data Type
<Value>	Process Property	Not applicable	Employee XML	String
ConnectionSubsystem	Literal	HTTPsendreceive_conn	Not applicable	Not applicable

Property Name	Type	Output Argument
Response	Output Argument	<Value>

9. Write the message to the file using the EAI File Transport with the Send method and the following input arguments.

Input Argument	Type	Value	Property Name	Property Data Type
<Value>	Process Property	Not applicable	Response	Binary
FileName	Literal	C:\SendRec.txt	Not applicable	Not applicable

10. Save your workflow and test it using the Workflow Simulator.

Examples Using HTTP Request

This topic provides the following examples of using the EAI HTTP Transport business service:

- *Controlling Login Sessions with Session Mode*
- *Sending Requests in Sessionless Mode*
- *Accessing a URL Protected by Basic Authentication*
- *Providing Client Certificate Information for TLS Mutual Authentication*

Controlling Login Sessions with Session Mode

The session mode allows control over login sessions. In this mode you log in first and open a session. Any message can be exchanged without having to log in again until you explicitly log off.

The following example shows parameters for Request and Logoff in a session mode HTTP request. Session cookies are required in a case such as this.

Note: You enter each of the following URLs as a continuous line of code.

- The following URL passes a query string as the SWEExtData value along with the GET request:

```
HTTPRequestURLTemplate = "http://$ServerPath$/
start.swe?SWEExtData=<Prop>somedata</Prop>HTTPRequestMethod='GET'"
```

- The following URL logs off from the server:

```
HTTPLogoffURLTemplate = "http://$ServerPath$/start.swe?SWEExtCmd=Logoff"
```

In these URL examples, the following parameter is used:

- `ServerPath = "siebel1/eai"`

In the examples, the `ServerPath` variable value of `siebel1/eai` is substituted for the token `$ServerPath$`.

Any XML document represented by the entry for SWEExtData can be put into the body. This would change the sample code so that HTTPRequestURLTemplate would read as:

```
HTTPRequestURLTemplate = "http://$ServerPath$/start.swe?"
```

Sending Requests in Sessionless Mode

The following example includes a Request Method, a Request, and a Login for a sessionless mode request. In this example, the request is simply passed to the secure server using the POST command. Unlike the Session Mode example, this request sends data in the body of the request. This request does not require cookies.

```
HTTPRequestMethod = "POST"

HTTPRequestURLTemplate = "https://accounts.mypartnerexample.com/server/login.asp"

HTTPRequestBodyTemplate = "Acct=ABCIntl&User=$Username$&pwd=$Password$"

Username = "acctuser"

Password = "123456789abcdefg"
```

Accessing a URL Protected by Basic Authentication

Siebel CRM supports server, or basic, authentication. You can use basic authentication with the EAI HTTP Transport to send messages. For more information about authentication, see *Siebel Security Guide* .

The format for accessing a URL protected by basic authentication with HTTP Outbound is:

```
http://username:password@host/rest of the URL
```

For example:

```
http://Administrator:manage@127.0.0.1:5555/example.com/stuff
```

Note: The EAI HTTP Transport business service does not provide standard parameters to support the use of Digest HTTP Authentication.

Providing Client Certificate Information for TLS Mutual Authentication

In certain versions, Siebel CRM supports client authentication for TLS-based communications (also known as mutual authentication) using the EAI HTTP Transport business service, and for workflows and outbound Web service calls that call the EAI HTTP Transport business service.

Note: For information about the specific versions that support mutual authentication, see 560965.1 (Article ID) on My Oracle Support.

CAUTION: It is strongly recommended to use Transport Layer Security (TLS) for best security, where possible. Using Secure Sockets Layer (SSL) is not supported for secure environments. For current information about TLS support, see 1944467.1 (Article ID) on My Oracle Support. See also *Siebel Security Guide* .

If client authentication is enabled, then the Siebel Server presents a client certificate to an external Web server by supplying values for the EAI HTTP Transport parameters HTTPCertSerialNo and HTTPCertAuthority.

If the EAI HTTP Transport business service is invoked directly by Siebel eScript or a workflow, then you can specify the HTTPCertSerialNo and HTTPCertAuthority parameters by setting input properties (business service method arguments).

The following is an example of the code used to call the EAI HTTP Transport business service using Siebel eScript:

```
var oService = TheApplication().GetService("EAI HTTP Transport");
var oInputs = TheApplication().NewPropertySet();
var oOutputs = TheApplication().NewPropertySet();

oInputs.SetProperty("HTTPRequestMethod", "GET");
oInputs.SetProperty("HTTPRequestURLTemplate", sUrl);

// Set the Serial Number of the Client Certificate
oInputs.SetProperty("HTTPCertSerialNo", "00d802dc387dd867b9");

// Set the RDN for the CA of the certificate
oInputs.SetProperty("HTTPCertAuthority","E=cacert@oracle.com,CN=somecertcomputer,
OU=ca,O=oracle,L=boston,C=usa");

// Invoke EAI HTTP Transport
oService.InvokeMethod("SendReceive", oInputs, oOutputs);
```

Note: If the EAI HTTP Transport business service is invoked indirectly by an outbound Web service, then you can specify the HTTPCertSerialNo and HTTPCertAuthority parameters as input arguments for the outbound Web Service Dispatcher. For information about setting parameters for the EAI HTTP Transport business service for outbound Web services, see *Integration Platform Technologies: Siebel Enterprise Application Integration* .

Note: On the UNIX (MainWin) operating system, SHA-2 encryption is not supported for the EAI HTTP Transport.

For more information about configuring TLS mutual authentication using the EAI HTTP Transport, see *Siebel Security Guide* .

Creating Custom Headers for the EAI HTTP Transport Service

Custom headers can be created when sending a request through the EAI HTTP Transport service using a script or a workflow.

To create custom headers for the EAI HTTP Transport service

- Create a new input property in the input to the HTTP transport.

  The name of the property must have a prefix of HDR. or HDR_ followed by the name of the custom header, for example:

```
httpIn.SetProperty("HDR.CustomHttpHeader","MyValue");
httpSvc.InvokeMethod("SendReceive", httpIn, httpOut);
```

A custom HTTP header with a name of "CustomHttpHeader" and a value of "MyValue" is the result.

Note: The HDR_ prefix can be useful in workflows for avoiding interference with the period (.) notation used in creating property sets.

About Sending and Receiving Messages Through HTTP

To send and receive messages through HTTP, you set up a workflow with the SendReceive method.

The Receive part of that method receives the response in an output argument of that method. You can then use the response to perform an upsert operation using an integration object and EAI Siebel Adapter, or display the response to your user. In this scenario, none of your quote integration design uses the eai.cfg or the Application Interface. You are performing an outbound HTTP call and waiting for a response synchronously.

You can then communicate the response to the user by displaying the returned error message in a browser alert or use the new User Interact step of the workflow to refresh the view and show any new updates to fields to the user. The User Interact step can run synchronously or asynchronously, in the local Siebel Application Object Manager or on the server.

About Transport Headers and HTTP Response Headers

This topic describes how transport headers and HTTP response headers work with HTTP Transport (outbound) to form a cookie handling system. HTTP Transport handles the cookie it receives from the server by storing and then creating a valid request transport header that it sends back to the server as a part of the request.

By exposing all the HTTP response headers as a part of output property set, you can handle the response accordingly. You can have all the HTTP response headers, as well as HTTP Status code, as part of the output property set.

Transport headers are preserved across various connections and are a part of the transport service and not the HTTP connection.

Features of Transport Headers

Transport headers have the following features:

- Every connection has its own transport header.
- The transport header separately stores each cookie sent by the server during a connection.

  For example, each name, domain, value pair, along with path, and other attributes (if present) are stored as a separate cookie in the transport header.
- Each cookie in the transport header has a distinct name.

  Two cookies with the same name cannot be present in the transport header at the same time. The second cookie overwrites the first one. Therefore, since the transport header is implemented as a CSSMapStringToPtr class, each cookie is hashed in the transport header based on its name.
- The transport header classifies cookies into two categories:
  - Type HTTP Version 1 and later.
  - Preliminary Netscape cookie spec type.
- When a ToString function is called on the transport header, it scans through the header and collects all the cookies in the header and creates a request transport header (based on the cookie category).
- The transport header is cleared when the connection is terminated.
- During SendReceive, the HTTP response has HTTP headers associated with it. Expose those response HTTP headers as properties of the output property set.

  All of these HTTP header properties are distinguished from other properties by adding the prefix HDR. in front of the property (header) name.
- Also, HTTP Status code for the HTTP request sent by way of EAI HTTP Transport is exposed as a property in the output property set. The property is called StatusCode.

8 Integrating Siebel CRM with Java Applications

Integrating Siebel CRM with Java Applications

This chapter discusses the integration of Java applications with Siebel CRM. It includes the following topics:

- *About Siebel CRM and Java Applications*
- *About the JDB Business Service API*
- *About the Siebel Code Generator*
- *About Running the Java Data Bean*
- *About the Siebel Resource Adapter*

About Siebel CRM and Java Applications

Many enterprises develop Java applications to meet a variety of business requirements. Typically, these applications combine existing enterprise information systems with new business functions to deliver services to a broad range of users. Oracle supports integration of its business services and business objects using the Siebel Java Data Bean. The Siebel Java Data Bean can be used for interaction with various kinds of Siebel application objects:

- Business objects and business components
- Business services and property sets
- Integration objects

In all cases, the Java code acts as client-side proxy stub to the corresponding object on the Siebel Server. It does not implement the functionality of the object in Java.

For ease of use, the Siebel Code Generator can be used to produce Java code based on the Siebel Java Data Bean for any specific business service or integration object. This generated code has an API specific to the chosen business service or integration object.

Additionally, Siebel CRM supports the Java EE Connector Architecture (JCA) with the Siebel Resource Adapter. The Siebel Resource Adapter supports the invocation of business services.

About the JDB Business Object API

The Java Data Bean provides an API to Siebel business objects and their business components. The API is similar in function to the API provided for other platforms, such as COM.

Example of the Business Object and Business Component Interface

Following is a code sample demonstrating use of the business object API. The sample shows how the Java Data Bean might be used to search for a Contact with a particular login name.

The first step in using the Siebel Java Data Bean is to log in to the Object Manager of the Siebel Server. The first parameter, the connection string, specifies the protocol, server name, enterprise name, and Application Object Manager name. Once logged into the Object Manager, the methods getBusObject and getBusComp are used to obtain business objects and their business components.

The code sample activates fields to allow the query to retrieve data for the specific fields, specifies the search criteria, and executes the query. If the query is successful, then the first and last name of the contact are printed to the standard output.

```
import com.siebel.data.*;

public class ObjectInterfaceExample {
 public static void main(String[] args) throws SiebelException {
 String connectString = siebel://examplecomputer:2321/siebel/SCCObjMgr_enu";

 SiebelDataBean dataBean = new SiebelDataBean();
 dataBean.login(connectString, "USER", "PWD", "enu");
 SiebelBusObject busObject = dataBean.getBusObject("Contact");
 SiebelBusComp busComp = busObject.getBusComp("Contact");

 busComp.setViewMode(3);
 busComp.clearToQuery();
 busComp.activateField("First Name");
 busComp.activateField("Last Name");
 busComp.activateField("Id");
 busComp.setSearchSpec("Login Name", "thomas");
 busComp.executeQuery2(true,true);

 if (busComp.firstRecord()) {
 System.out.println("Contact ID: " + busComp.getFieldValue("Id"));
 System.out.println("First name: " + busComp.getFieldValue("First Name"));
 System.out.println("Last name: " + busComp.getFieldValue("Last Name"));
 }
 busComp.release();
 busObject.release();
 dataBean.logoff();
 }
```

If the query results in multiple records, then the record set can be iterated as follows:

```
if (busComp.firstRecord()) {
 // obtain the fields/values from this record
 while (busComp.nextRecord()){
 // obtain the fields/values from the next record
 }
}
```

About the JDB Business Service API

Aside from the business object and business component API, the primary point of integration with the Siebel application is by using business services.

There are several ways to invoke a business service. The simplest way is using the Siebel Java Data Bean directly, as shown in the following example. Alternatively, Siebel Tools provides a Code Generator which creates, for any business service, Java classes that invoke the business service. The generated code can invoke the business service either using the Siebel Java Data Bean or using the Siebel Resource Adapter. The creation and use of generated code is described in the next topic. The Siebel Resource Adapter is part of the Java EE Connector Architecture, which is described in *About the Siebel Resource Adapter*.

The following is an example of invoking a business service directly using the Siebel Java Data Bean.

```
import com.siebel.data.SiebelDataBean;
import com.siebel.data.SiebelException;
import com.siebel.data.SiebelPropertySet;
import com.siebel.data.SiebelService;
public class BasicDataBeanTest {
public static void main(String[] args) throws SiebelException {
SiebelDataBean dataBean = new SiebelDataBean();
dataBean.login("siebel://examplecomputer:2321/siebel/SCCObjMgr_enu", "USER", "PWD", "enu");
SiebelService businessService = dataBean.getService("Workflow Utilities");
SiebelPropertySet input = new SiebelPropertySet();
SiebelPropertySet output = new SiebelPropertySet();
input.setValue("Please echo this");
businessService.invokeMethod("Echo", input, output);
System.out.println("Output: " + output.toString());
}
}
```

About the Siebel Code Generator

JavaBeans for invoking a particular business service can be generated using the Siebel Code Generator. These JavaBeans provide a uniform mechanism for interacting with the Siebel application from a Java or Java EE application. The JavaBean for a particular business service provides facilities for creating inputs and invoking methods. The JavaBean representing a business service can be based on either the Siebel Java Data Bean or on the Siebel Java EE Connector Architecture (JCA) Resource Adapter.

For business services whose methods have integration objects as input or output, JavaBeans representing the integration objects must be generated separately. These beans provide facilities for creating the integration objects and setting their fields.

The business services most commonly used for integration are EAI Siebel Adapter and various ASI business services based on the data sync service. The methods of these business services typically have inputs and outputs that are property sets of a special type called *integration objects*. Siebel Java integration provides special support for working with integration objects.

The following Siebel Code Generator topics are also discussed:

- *Invoking the Siebel Code Generator*

- *Code Generated for a Business Service*
- *Connect String and Credentials for the SiebelDataBean*
- *Connection Parameters for the SiebelDataBean*

Invoking the Siebel Code Generator

This topic describes how to invoke the Siebel Code Generator to create JavaBeans for either a Siebel business service or a Siebel integration object.

To invoke the Siebel Code Generator

1. Start Siebel Tools.

   > **Note:** For information about how to use Siebel Tools, see *Using Siebel Tools* .

2. Select Business Service or Integration Object in the Object Explorer.

   > **Note:** If Integration Object is not present, then add it by checking Integration Object on the Object Explorer tab of the Development Tools Options window opened by selecting View, then Options.

3. Select the desired business service or integration object.

   For example, at the first section of the Integration Object list, there is a set of three buttons: Synchronize, Generate Schema, and Generate Code.
4. Click Generate Code.
5. Complete the Code Generator wizard:

   a. Leave the business service as is. There is only one available, the Siebel Code Generator.
   b. Select either Java(JDB) (Java Data Bean) or Java(JCA) (Java EE Connector Architecture/Siebel Resource Adapter) for the Supported Language.
   c. Browse to select an existing folder as the output folder. Your Java code for the selected business services or integration objects is stored in subdirectories there, as explained next.
   d. Click Finish.

      The code is generated and the wizard closes, returning you to the Business Service or Integration Object form.

Code Generated for a Business Service

The code generated for a business service includes a class representing the business service itself as well as classes representing inputs and outputs of its methods. These classes are described in detail in this topic.

ASI business services based on the data sync service have integration objects as part of the input or output of their methods. The JavaBeans representing these integration objects must be generated separately from the business service.

The classes for a given business service reside in a package in one of the following:

- com.siebel.service.jdb.business service name or
- com.siebel.service.jca.business service name

Depending on whether the beans are based on the Java Data Bean or the Siebel JCA Resource Adapter. For example, generated JDB code for the EAI Siebel Adapter resides in the package com.siebel.service.jdb.eaisiebeladapter.

The Code Generator creates the standard Java directory structure reflecting the package structure. As shown in the following image, a subfolder named `com` is created in the folder specified during the generation process. The `com` folder contains a folder named `siebel`, which in turn contains a folder named `service`. Under the `service` is a folder named `jdb` (or `jca`), containing a folder named for the business service. This last folder contains the classes for the business service. Each class is defined in its own file. The folder created under `jdb` (or `jca`) for every business service generated contains several Java files.



One Java class is generated to represent the business service itself. The name of the class is the name of the business service with all special characters replaced by underscores (_) and BusServAdapter appended to the end. For example, the class representing EAI Siebel Adapter is EAI_Siebel_AdapterBusServAdapter.

The Java class has one method for each method of the business service. Its name is the name of the method with m prefixed. For code based on the Java Data Bean, the class is a subclass of com.siebel.integration.adapter.SiebelJDBAdapterBase. For code based on the Siebel Resource Adapter, the class is a subclass of com.siebel.integration.adapter.SiebelJCAAdapterBase.

Additionally, for each method of the business service defined in Siebel Tools, one Java class is created for the method's input and one for the method's output. The name of the class is the name of the method with Input or Output appended. The class encapsulates all input (or output) arguments for the method. Each argument is represented as a field whose name is that of the argument with f prefixed. For each field, public set and get methods are provided Java methods for reading and writing their values.

For example, the business service CC XML Converter, which has two methods, PropSetToXML and XMLToPropSet, generates the following four classes:

- CC_XML_Converter BusServiceAdapter
- PropSetToXMLInput
- PropSetToXMLOutput
- XMLToPropSetInput

The first class, CC_XML_Converter BusServiceAdapter, represents the business service as a whole; it has methods mPropSetToXML and mXMLToPropSet. The other three classes represent the input or output parameters of the two methods. (Notice there is no class XMLToPropSetOutput because that method has no outputs.) Those three classes each have methods to read and write the individual parameters, as well as methods to convert to and from a com.siebel.data.SiebelPropertySet.

About Methods of Java Classes Generated for a Business Service

The tables in the following topics describe the methods that are present in the generated Java code for every business service. Generic names (for example, GenericService and GenericMethod) are substituted for the actual names of the business service, methods, and arguments.

- *Methods for Java class com.siebel.service.jdb.GenericServiceBusServAdapter*,
- *Methods for Java class com.siebel.service.jdb.GenericMethodInput*, and
- *Methods for Java class com.siebel.service.jdb.GenericMethodOutput Methods*

Methods for Java class com.siebel.service.jdb.GenericServiceBusServAdapter

The following table lists the methods in the Java class com.siebel.service.jdb.GenericServiceBusServAdapter generated for an example business service, GenericService, that has the business service method GenericMethod.

Method	Description
GenericServiceBusServAdapter()	Constructor that uses the default properties file, siebel.properties.
GenericServiceBusServAdapter(SiebelDataB	Constructor that reuses the resources of an existing SiebelDataBean.
GenericServiceBusServAdapter(String)	Constructor taking the name of the properties file to use.
GenericServiceBusServAdapter(String, String, String)	Constructor taking the username, password, and connect string.
GenericServiceBusServAdapter(String, String, String, String)	Constructor taking the username, password, connect string, and language.
GenericMethod(GenericMethodInput)	Invokes the specified business service method.

Methods for Java class com.siebel.service.jdb.GenericMethodInput

The following table lists the methods in the Java class com.siebel.service.jdb.GenericMethodInput generated for an example business service method, GenericMethod.

Method	Description
GenericMethodInput()	Constructor.
GenericMethodInput(SiebelPropertySet)	Constructor that sets its fields from the given property set.
fromPropertySet(SiebelPropertySet)	Copies field values from the given property set.

Method	Description
toPropertySet()	Returns a SiebelPropertySet with the properties and values corresponding to the fields of this object.
getfGenericArgument()	Returns the value of business service method argument.
setfGenericArgument(String)	Sets the value of a business service method argument.

Methods for Java class com.siebel.service.jdb.GenericMethodOutput Methods

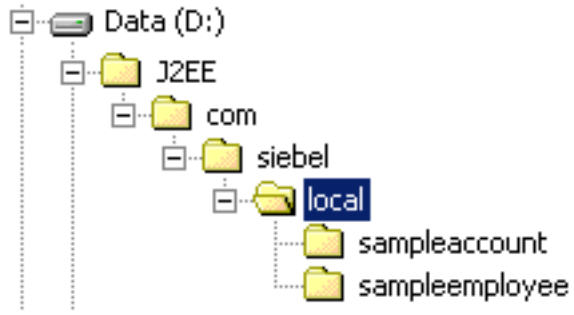
The following table lists the methods in the Java class com.siebel.service.jdb.GenericMethodOutput generated for an example business service method, GenericMethod.

Method	Description
GenericMethodOutput()	Constructor.
GenericMethodOutput(SiebelPropertySet)	Constructor that sets its fields from the given property set.
fromPropertySet(SiebelPropertySet)	Copies field values from the given property set.
toPropertySet()	Returns a SiebelPropertySet with the properties and values corresponding to the fields of this object.
getfGenericArgument ()	Returns the value of business service method argument.
setfGenericArgument ()	Sets the value of a business service method argument.

About the Code Generated for an Integration Object

Integration objects are special kinds of property sets that are the input and output of business services based on the data sync service. JavaBeans based on integration objects are designed to be used with those business services or with the EAI Siebel Adapter and can be used to query, delete, upsert, and synchronize information in the Siebel Server's database.

The integration object, and each of its components, has its own Java class, stored in the package com.siebel.local.*IntegrationObjectName*. The class for the integration object has IO appended to the end, and the class for an integration component has IC appended. The Code Generator creates the standard Java directory structure reflecting the package structure. In the selected folder, a subfolder named `com` is created, containing a subfolder `siebel`, containing a subfolder `local`, which contains one subfolder for each integration object that was generated. The Java files are stored in the lowest directory. This structure is shown in the following image. One folder is created under `local` for each integration object that is generated. The folder that is created contains all the Java files for that integration object.



For example, the integration object Sample Account; which has five components Account, Account Attachment, Account_Organization, Business Address, and Contact; generates the following six classes:

- Sample_AcccountIO
- AccountIC
- Account_AttachmentIC
- Account_OrganizationIC
- Business_AddressIC
- ContactIC

The first class, suffixed with IO, represents the entire integration object. It has methods to construct the object, to read and write fields, to add integration object components, and to convert to and from a SiebelPropertySet. The other five classes, suffixed with IC, represent the individual integration object components and provide methods that are for constructing the component to read and write fields and to convert to and from a SiebelPropertySet.

Methods of Java Classes Generated for an Integration Object

The following table describes the methods that are present in the generated Java code for every integration object, using the example integration object GenericIntObj.

Object	Description
addfintObjInst(SiebelHierarchy)	Adds an integration object component object to the integration object.
clone	Returns a copy of the integration object.
equals(Object)	Determines whether integration object has the same data as the integration object passed.
fromPropertySet(SiebelPropertySet)	Copies the data from the given property set to the integration object.
getfIntObjectFormat	Returns a String containing the format of the integration object.
getfIntObjectName	Returns the integration object name property.
getfintObjInst	Returns a Vector representation of the integration object.
getfMessageId	Returns the MessageId property of the integration object.

Object	Description
getfMessageType	Returns the MessageType property of the integration object.
getfOutputIntObjectName	Returns the OutputIntObjectName property of the integration object.
Generic_ObjectIO()	Default constructor.
Generic_ObjectIO(SiebelPropertySet ps)	Creates an integration object (and its hierarchy) based on a property set.
setfIntObjectFormat	Sets the IntObjectFormat property of the integration object.
setfIntObjectName	Sets the IntObjectName property of the integration object.
setfMessageId	Sets the MessageId property of the integration object.
setfMessageType	Sets the MessageType property of the integration object.
setfOutputIntObjectName	Sets the OutputIntObjectName property of the integration object.
toPropertySet	Returns a SiebelPropertySet representation of the integration object.

Methods of Java Classes Generated for an Integration Object Component

The following table describes the methods that are present in the generated Java code for every integration object component, using an example integration object component, GenericIntComp, that has the child components GenericIntCompChild and field GenericField.

Object Component	Description
addfGenericIntCompChildIC(GenericIntCom	Adds to the integration object component the given child integration object component.
clone	Returns a copy of the integration object.
equals(Object)	Determines whether the integration object component has the same data as the passed integration object component.
fromPropertySet(SiebelPropertySet)	Populates the integration object component based upon the contents of a property set.
getfGenericIntCompChildIC	Returns a Vector containing all child integration object components of type ChildIntObjComp associated with the integration object component.
getfGenericField()	Returns the value of the field GenericField.

Object Component	Description
GenericIntCompIC()	Default constructor.
GenericIntCompIC(SiebelPropertySet)	Creates an integration object component from a property set.
setfGenericField(val)	Sets the value of the field GenericField.
toPropertySet	Returns a property set representation of the integration object component.

About Running the Java Data Bean

Two Siebel .jar files are needed to compile and run a Java application that uses the Java Data Bean:

- Siebel.jar
- SiebelJI_lang.jar (lang is the installed language pack; for example, SiebelJI_enu.jar for English or SiebelJI_jpn.jar for Japanese.)

These jar files are provided with the standard Siebel installation under the directory `INSTALLED_DIR\classes`.

Documentation of individual classes is provided in the form of javadoc (Siebel_JavaDoc.jar), which is installed when installation option Siebel Java Integrator (a component of the Siebel Tools or the Siebel Server installer) is chosen. This .jar file contains the up-to-date javadoc for the Siebel Java Data Bean, Siebel Resource Adapter, and dependent classes.

> **Note:** The Siebel Data Bean is not thread-safe: simultaneous access by different threads is not supported. This restriction applies to all objects obtained from the same instance of SiebelDataBean. For example, if two instances of SiebelBusObj are obtained from the same instance of SiebelDataBean, then methods on them are not invoked simultaneously by different threads.

Connect String and Credentials for the SiebelDataBean

When using the SiebelDataBean directly, without any generated code, three arguments must be passed to the login method. A fourth argument, language code, is optional.

- connect string
- Siebel username
- Siebel password
- language code (default is enu)

The connect string has the following form:

```
siebel://SiebelServerName:SCBPort/EnterpriseName/XXXObjMgr_lang
```

For example:

```
siebel://examplecomputer:2321/mysiebelenterprise/SCCObjMgr_enu
```

When using generated code, these parameters can be taken from the siebel.properties file, which must be in the classpath of the Java Virtual Machine (JVM). These properties are read from siebel.properties at the time an instance of the generated business service class is created using that explicitly specifies siebel.properties, for example:

```
Siebel_AccountBusServAdapter svc = new
Siebel_AccountBusServAdapter("siebel.properties");
```

They can be overridden by calling the methods setConnectString, setUserName, setPassword, and setLanguage any time prior to calling initialize() or invoking a business service method (such as GenericMethod in *Methods for Java class com.siebel.service.jdb.GenericServiceBusServAdapter*). This is the behavior when the default (no-argument) constructor of the generated Java class is used.

Alternatively, the generated class provides the following four constructors with arguments:

- One String argument: the name of the property file to be used.
- Three String arguments: the connect string, username, and password. No properties file is used.
- Four String arguments: the connect string, username, password, and language. No properties file is used.
- SiebelDataBean argument: the SiebelDataBean passed already has parameters assigned and its login method executed.

Connection Parameters for the SiebelDataBean

Regardless of how the SiebelDataBean is invoked, certain parameters of the connection can be set using the properties file. These are siebel.conmgr.txtimeout, siebel.conmgr.poolsize, siebel.conmgr.sesstimeout, siebel.conmgr.retry, and siebel.conmgr.jce.

Other connection parameters can also be specified in the properties file, but they are used only in conjunction with generated code (subclasses of com.siebel.integration.adapter.SiebelJDBAdapterBase or SiebelJCAAdapterBase).

Property	Description
siebel.conmgr.txtimeout	The number of milliseconds to wait after sending a request to the Siebel Server. Must be a positive integer; other values are ignored. The default value is 600000 milliseconds (10 minutes); the maximum value is 2,147,483,647 ms (approximately 25 days).
siebel.conmgr.poolsize	For each Application Object Manager process, a pool of open connections is maintained and shared by all users of that process. This parameter specifies the maximum number of connections that are stored in the pool. Its value must be a positive integer less than 500; other values are ignored. The default is 2.
siebel.conmgr.sesstimeout	The number of seconds the Siebel Server waits before disconnecting an idle client session. Its value must be a nonnegative integer. The default is 2700 seconds (45 minutes); the maximum value is 2,147,483,647 s (approximately 68 years).
siebel.conmgr.jce	Determines whether encryption of transmissions is done using Java Cryptography Extension (JCE) or RSA (if the connection uses encryption). 1 indicates JCE; 0 indicates RSA. The default is 0.
siebel.conmgr.retry	The number of attempts to be made at establishing a connection (opening a session) before giving up. Must be a positive integer. The default is 3.

Property	Description

siebel.conmgr.virtualhosts	A listing of virtual servers representing a group of like servers that perform the same function, for example, call center functions.
	
	An incoming login for the call center virtual server tries servers from the list in a round-robin fashion.
	
	An example of such a list follows:
	
	`VirtualServer1=sid1:host:port,sid2:host:port...;`
	`VirtualServer2=...`
	
	where VirtualServer1, VirtualServer2, and so on, are assigned lists of real Siebel Servers with host names and port numbers (of the local SCBroker component).

siebel.connection.string	The Siebel connect string. For information about the syntax of the connect string, see *Siebel Object Interfaces Reference* .
siebel.loglevel	The level of messages to be logged. Must be a positive integer less than 6. Other values are ignored or throw an exception. 0 causes only FATAL messages to be logged; 1 ERROR; 2 WARN; 3 INFO; 4 DETAIL; 5 DEBUG. The default is 0. **Note:** The siebel.loglevel parameter is used only in conjunction with the generated code for the SiebelJCAAdapterBase subclass.
siebel.logfile	The name of a file to which logging is directed. Strings that cause a FileNotFoundException cause an error to be logged and are ignored. The default is to print to the JVM's standard output. **Note:** The siebel.logfile parameter is used only in conjunction with the generated code for the SiebelJCAAdapterBase subclass.
siebel.user.name	The Siebel username to be used for logging in to the Application Object Manager.
siebel.user.password	The Siebel password to be used for logging in to the Application Object Manager.
siebel.user.language	The language code indicating the natural language to be used for messages and other strings. Default is enu.
siebel.jdb.classname	The name of a subclass of com.siebel.data.SiebelDataBean to use instead of SiebelDataBean. Strings that do not specify a valid class or specify a class that is not a subclass of SiebelDataBean cause an error log to be logged and SiebelDataBean to be used instead.

Here is a sample siebel.properties file:

```
siebel.connection.string = siebel://examplecomputer:2321/siebel/EAIObjMgr_enu
siebel.user.name = User1
siebel.user.password = password
siebel.user.language = enu
siebel.user.encrypted = false
siebel.conmgr.txtimeout = 300000
siebel.conmgr.poolsize = 5
siebel.conmgr.sesstimeout = 3600
siebel.conmgr.retry = 5
```

```
siebel.conmgr.jce = 1
siebel.loglevel = 0
```

Examples Using Generated Code for Integration Objects

The following code examples use the code generation facilities provided in Siebel Tools. For more information, see *About the Siebel Code Generator*, for both business services and integration objects. By using the code generation facilities, many of the complexities of the Siebel property sets and business service interfaces have been abstracted, providing a standards-based JavaBean interface.

Siebel Account Business Service Example

The following is a code sample invoking the QueryByExample method of the Siebel Account business service. In addition to the generated code for Siebel Account (resident in com.siebel.service.jdb.siebelaccount), the sample uses the generated code for the Account Interface integration object (resident in com.siebel.local.accountinterface).

The code invokes the QueryByExample method of the Siebel Account business service. The parameter to this method is formed from an instance of the Account Interface integration object, which serves as the example, essentially specifying a search criterion of all accounts that start with the letters Ai. The output integration object is converted to a Vector and iterated through to print the names of matching accounts.

```
import com.siebel.data.SiebelDataBean;
import com.siebel.data.SiebelException;
import com.siebel.service.jdb.siebelaccount.Siebel_AccountBusServAdapter;
import com.siebel.service.jdb.siebelaccount.QueryByExampleInput;
import com.siebel.service.jdb.siebelaccount.QueryByExampleOutput;
import com.siebel.local.accountinterface.Account_InterfaceIO;
import com.siebel.local.accountinterface.AccountIC;

public class JDBSiebelAccount {
public static void main(String[] args) throws SiebelException {
Siebel_AccountBusServAdapter svc = new Siebel_AccountBusServAdapter("USER",
"PWD","siebel://examplecomputer:2321/siebel/SCCObjMgr_enu","enu");

// Create the example-accounts starting with "Ai":
AccountIC acctIC = new AccountIC();
Account_InterfaceIO acctIO = new Account_InterfaceIO();
acctIO.addfintObjInst(acctIC);
acctIC.setfName("Ai*");
QueryByExampleInput qbeIn = new QueryByExampleInput();
qbeIn.setfSiebelMessage(acctIO);

// Call QueryByExample
QueryByExampleOutput qbeOut = svc.mQueryByExample(qbeIn);
acctIO = new Account_InterfaceIO(qbeOut.getfSiebelMessage().toPropertySet());
Vector ioc = acctIO.getfintObjInst();

// print the name of each account returned:
if (!ioc.isEmpty()) {
for(int i=0; i < ioc.size(); i++) {
acctIC = (AccountIC) ioc.get(i);
System.out.println(acctIC.getfName());
}
}
}
```

EAI Siebel Adapter Business Service Example

The following example uses the generated code for the EAI Siebel Adapter business service. An instance is instantiated using the constructor that takes an instance of SiebelDataBean. The QueryPage method is called; its output is actually an Account Interface integration object, but the object returned is not strongly typed and instead is used to construct an Account Interface instance. The generated code for Account Interface is also needed for this example.

```
import com.siebel.data.SiebelDataBean;
import com.siebel.data.SiebelException;
import com.siebel.local.accountinterface.Account_InterfaceIO;
import com.siebel.local.accountinterface.AccountIC;
import com.siebel.service.jdb.eaisiebeladapter.EAI_Siebel_AdapterBusServAdapter;
import com.siebel.service.jdb.eaisiebeladapter.QueryPageInput;
import com.siebel.service.jdb.eaisiebeladapter.QueryPageOutput;

public class DataBeanDemo {
public static void main(String[] args) throws SiebelException {
SiebelDataBean m_dataBean = new SiebelDataBean();
String conn = "siebel://examplecomputer:2321/siebel/SCCObjMgr_enu";
m_dataBean.login(conn, "USER", "PWD", "enu");

// Construct the EAI Siebel Adapter, using the data bean
EAI_Siebel_AdapterBusServAdapter svc =
new EAI_Siebel_AdapterBusServAdapter(m_dataBean);
svc.initialize();
try {
// Set values of the arguments to the QueryPage method.
QueryPageInput qpInput = new QueryPageInput();
qpInput.setfPageSize(Integer.toString(10)); // Return 10 records.
qpInput.setfOutputIntObjectName("Account Interface");
qpInput.setfStartRowNum(Integer.toString(0)); // Start at record 0.
QueryPageOutput qpOutput = svc.mQueryPage(qpInput);

// Construct the integration object using the QueryPage output
Account_InterfaceIO acctIO =
new Account_InterfaceIO(qpOutput.getfSiebelMessage().toPropertySet());

// Convert the results to a vector for processing
Vector ioc = acctIO.getfintObjInst();

// Print name of each account
if (!ioc.isEmpty()) {
for (int i = 0; i < ioc.size(); i++) {
AccountIC acctIC = ((AccountIC) ioc.get(i));
System.out.println(acctIC.getfName());
}
}
}
catch (SiebelException e) {}
finally {
m_dataBean.logoff();
}
}
}
```

About the Siebel Resource Adapter

The Siebel Resource Adapter is for use within the Java EE Connector Architecture (JCA) by Java EE-based applications (EJBs, JSPs, servlets) that are deployed on containers. JCA provides clients with a standard interface to multiple enterprise information services such as the Siebel application.

The Siebel Resource Adapter implements system-level contracts that allow a standard Java EE application server to perform services such as pooling connections and managing security. This is referred to as operation within a *managed environment*.

The Java EE Connection Architecture also provides for operation in a *nonmanaged environment*, where the client need not be deployed in a Java EE container, but instead uses the adapter directly. In this case, the client takes responsibility for services such as managing security.

The Siebel Resource Adapter has transaction support level *NoTransaction*. This means that the Siebel Resource Adapter does not support local or JTA transactions. For more information about JCA, see:

http://jcp.org/en/jsr/detail?id=322

The following Siebel Resource Adapter topics are also discussed:

- *Using the Resource Adapter*
- *About the Connect String and Credentials for the Java Connector*
- *About JCA Logging*

Using the Resource Adapter

When deploying the Siebel Resource Adapter to a Java EE application server (for example, Oracle Application Server, Oracle WebLogic Server, or IBM WebSphere MQ), you must make sure that the necessary Siebel JAR files are included. The Siebel JAR files that must be added to the classpath are:

- SiebelJI.jar
- SiebelJI_lang.jar (lang is the installed language pack; for example, SiebelJI_enu.jar for English or SiebelJI_jpn.jar for Japanese.)

The resource adapter archive, or RAR file, might also be required for deployment. Refer to the documentation of the Java EE application server for more information about deploying a JCA adapter on the server.

The following topics contain code samples for both managed and nonmanaged environments.

About the Connect String and Credentials for the Java Connector

The Java Connector Architecture allows for credentials to be supplied using either *Container-Managed Sign-on* or *Application-Managed Sign-On*.

With Container-Managed Sign-On, the application server's container identifies the principal and passes it to the JCA adapter in the form of a JAAS Subject. Application servers provide their own system of users and roles; such a user must be mapped to Siebel user and password for the purpose of the JCA adapter. Application servers allow the specification of such mappings. With Container-Managed Sign-On, the Siebel connect string and language must be specified in the

deployment descriptor of the adapter (ra.xml). If a Siebel user name and password are present in the descriptor, then they are used by the application server only to create an initial connection to the Siebel application when the application server is started, which is not necessary.

With Application-Managed Sign-On, the client application must provide the credentials and connect string. This is done just as for the Java Data Bean, as described in *About Running the Java Data Bean*, by either supplying them in siebel.properties or setting them programmatically using setUserName, setPassword, setConnectString, and setLanguage. If any of these parameters are supplied using Application-Managed Sign-On, then supply all four of them in that manner.

Note: Connection parameters beginning with siebel.conmgr are read from siebel.properties, whether the adapter is being used in managed or nonmanaged mode.

Managed Code Sample Using the Siebel Resource Adapter

The following is a code sample using the Siebel Resource Adapter in a managed environment. The sample is a servlet that makes a simple invocation to a business service using the generated JCA code. (For more information about generating code, see *About the Siebel Code Generator*.)

The JCA ConnectionFactory is obtained through JNDI. Credentials are obtained at run time from the JAAS Subject passed to the servlet. The connect string and language are obtained from the deployment descriptor (ra.xml). Other connection parameters are obtained from the siebel.properties file.

Note: The siebel.properties file must be in the JVM classpath and must be specified explicitly when the business service instance is created.

```
import javax.naming.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.siebel.integration.jca.cci.SiebelConnectionFactory;
import com.siebel.service.jca.eaifiletransport.*;

public class ManagedConnectionServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException,ServletException {
PrintWriter reply = response.getWriter();

try {
// Specify siebel.properties in the constructor.
EAI_File_TransportBusServAdapter bs = new
EAI_File_TransportBusServAdapter("siebel.properties"?);
InitialContext jndi = new InitialContext();
SiebelConnectionFactory scf =
(SiebelConnectionFactory)jndi.lookup("siebelJCA");
bs.setConnectionFactory(scf);

// Username and password obtained from JAAS Subject passed by server at runtime.
// Connect string and language obtained from deployment descriptor, ra.xml.
ReceiveInput input = new ReceiveInput();
input.setfCharSetConversion("UTF-8");
input.setfFileName("D:\\helloWorld.txt");
ReceiveOutput output = bs.mReceive(input);
reply.println(output.getf_Value_());
}
catch (Exception e) {
reply.println("Exception:" + e.getMessage());
}
```

```
}
}
```

Nonmanaged Code Sample Using the Siebel Resource Adapter

The following is a code sample using the Siebel Resource Adapter in a nonmanaged environment. The sample performs the same function as the Managed sample; it is a servlet that makes a simple invocation to a business service using the generated JCA code. (For more information about generating code, see *About the Siebel Code Generator*.)

The JCA ConnectionFactory is created directly. The username, password, connect string, and language are obtained from siebel.properties or set programmatically. Other connection parameters are obtained from the siebel.properties file.

> **Note:** The siebel.properties file must be in the JVM classpath and must be specified explicitly when the business service instance is created.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.siebel.integration.jca.cci.notx.SiebelNoTxConnectionFactory;
import com.siebel.service.jca.eaifiletransport.*;
public class BookshelfNonManagedConnectionSample extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException {

PrintWriter reply = response.getWriter();
try {
EAI_File_TransportBusServAdapter bs = new
EAI_File_TransportBusServAdapter("siebel.properties#?);
bs.setConnectionFactory(new SiebelNoTxConnectionFactory());
// Username, password, connect string, and language are read from
// siebel.properties, which must be in the classpath of the servlet
// and be specified in the constructor.
// Alternatively, they can be set here programmatically:
// bs.setUserName("USER");
// bs.setPassword("PWD");
// bs.setConnectString("siebel://examplecomputer:2321/siebel/
SCCObjMgr_enu");
ReceiveInput input = new ReceiveInput();
input.setfCharSetConversion("UTF-8");
input.setfFileName("D:\\helloWorld.txt");
ReceiveOutput output = bs.mReceive(input);

reply.println(output.getf_Value_());
}
catch (Exception e) {
reply.println("Exception:" + e.getMessage());
}
}
}
```

About JCA Logging

The following improvements have been made to JCA logging in Oracle's Siebel CRM version 8.0 and later:

- Appending JCA logs to one file, which is found in the working directory of the JVM.

Previously, each JCA thread would overwrite the same log file over and over again. Now all JCA threads log into one file. When the log file size exceeds 100 MB, it is renamed and a new one is started. For example, test.log is renamed to test_1166581351656.log, where the value is the number of milliseconds since 1970.

- Proper logging of call stacks for LOG_DEBUG.

  Previously, JCA log events in the LOG_DEBUG level (level 5) logged the call stack, but the call stack was often incomplete and cryptic. Now the call stack is a complete Java call stack.

- Logging of thread names.

  Previously, the JCA logs did not include the thread name. Now that all threads log to one file, each line contains the thread name. An example of a line in the log file is:

```
[SIEBEL INFO] Thread[Servlet.Engine.Transports : 4,5,main] [2010-11-04
15:58:38.058] [SiebelManagedConnection(2137125295)] Cleaning up 0 handles on
SiebelManagedConnection(2137125295)
```

- New logging in LOG_DETAIL (level 4):
  - When a listener thread is created (logs the host and port):

```
[SIEBEL DETAIL] Thread[Thread-1482,5,main] [2010-11-04 16:12:10.139] [] creating
socket for listening thread: host=xyz port=9312
```

  - When the main thread sends a request to the Siebel Server (logs the packet number):

```
[SIEBEL DETAIL] Thread[Thread-1482,5,main] [2010-11-04 16:12:56.521] [] set tx=2813

[SIEBEL DETAIL] Thread[Thread-1482,5,main] [2010-11-04 16:12:56.521] [] wait=1 tx=2813
```

  - When the main thread receives a response:

```
[SIEBEL DETAIL] Thread[Thread-1482,5,main] [2010-11-04 16:12:56.580] [] end loop tx=2813 isDone
```

  - Before the listener thread reads a packet (logs the number of bytes in the packet):

```
[SIEBEL DETAIL] Thread[Thread-54,5,Listener Threads] [2010-11-04 16:12:56.575] [] about to read to
 bytes: len=1800
```

  - As the listener thread reads the packet (logs the packet number and number of bytes read thus far):

```
[SIEBEL DETAIL] Thread[Thread-54,5,Listener Threads] [2010-11-04 16:12:56.575] [] read some bytes:
 tx=2813 len=1800 read=1800
```

- Logging call stacks when opening and closing a connection to the Siebel Server.

  Previously, the JCA logs for LOG_INFO (level 3) logged the opening and closing of a connection, but did not log the call stack. Now the call stack is logged, for example:

```
[SIEBEL INFO] Thread[Servlet.Engine.Transports : 2,5,main] [2010-11-05
07:53:26.078] [SiebelConnection(507473761)] Opening a new connection to Siebel
...
java.lang.Throwable
 at com.siebel.integration.util.a.trace(Unknown Source)
 at com.siebel.integration.util.SiebelTrace.trace(Unknown Source)
 at com.siebel.integration.jca.cci.SiebelConnection.a(Unknown Source)
 at com.siebel.integration.jca.cci.SiebelConnection.initialize(Unknown Source)
 at com.siebel.integration.jca.cci.SiebelConnection.<init>(Unknown Source)
 at com.siebel.integration.jca.cci.notx.SiebelNoTxConnection.<init>(Unknown Source)
```

```
 at
 com.siebel.integration.jca.spi.notx.SiebelNoTxManagedConnectionFactory.createManagedConnection(Unknown
 Source)
 at com.ibm.ejs.j2c.poolmanager.FreePool.createManagedConnectionWithMCWrapper(FreePool.java(Compiled
 Code))
 at com.ibm.ejs.j2c.poolmanager.FreePool.createOrWaitForConnection(FreePool.java(Compiled Code))
 at com.ibm.ejs.j2c.poolmanager.PoolManager.reserve(PoolManager.java(Compiled Code))
 at com.ibm.ejs.j2c.ConnectionManager.allocateMCWrapper(ConnectionManager.java(Compiled Code))
 at com.ibm.ejs.j2c.ConnectionManager.allocateConnection(ConnectionManager.java(Compiled Code))
 at com.siebel.integration.jca.cci.SiebelConnectionFactory.getConnection(Unknown Source)
 at com.siebel.integration.adapter.SiebelJCAAdapterBase.invoke(SiebelJCAAdapterBase.java(Compiled Code))
 ...
[SIEBEL INFO] Thread[Servlet.Engine.Transports : 2,5,main] [2010-11-05 07:53:26.243]
 [SiebelConnection(507473761)] Opened a new connection to Siebel (Siebel session :
 siebel.tcpip.none.none://myserver.example.com:2321/esblp01/SCCObjMgr_enu/!10.6373.3ba70.465c2246)

[SIEBEL INFO] Thread[Thread-56,5,main] [2010-11-05 07:54:38.484] [SiebelConnection(974516018)] Closing
 the connection

java.lang.Throwable
 at com.siebel.integration.util.a.trace(Unknown Source)
 at com.siebel.integration.util.SiebelTrace.trace(Unknown Source)
 at com.siebel.integration.jca.cci.SiebelConnection.a(Unknown Source)
 at com.siebel.integration.jca.cci.SiebelConnection.close(Unknown Source)
 at com.siebel.integration.jca.spi.SiebelManagedConnection.destroy(Unknown Source)
 at com.ibm.ejs.j2c.MCWrapper.destroy(MCWrapper.java:1380)
 at com.ibm.ejs.j2c.poolmanager.FreePool.cleanupAndDestroyMCWrapper(FreePool.java(Compiled Code))
 at com.ibm.ejs.j2c.poolmanager.PoolManager.reclaimConnections(PoolManager.java(Compiled Code))
 at com.ibm.ejs.j2c.poolmanager.PoolManager.executeTask(PoolManager.java(Compiled Code))
 at com.ibm.ejs.j2c.poolmanager.TaskTimer.executeTask(TaskTimer.java(Compiled Code))
 at com.ibm.ejs.j2c.poolmanager.TaskTimer.run(TaskTimer.java:113)
```

- Logging execution of a request in LOG_INFO (level 3).

  Previously, execution of a request was logged in LOG_DEBUG. Now the request is logged in LOG_INFO with no call stack, for example:

```
[SIEBEL INFO] Thread[Servlet.Engine.Transports : 2,5,main] [2010-11-05 07:53:26.244]
 [SiebelConnection(507473761)] Executing
com.siebel.integration.jca.client.SiebelInteractionSpec@1b6bef7c
```

Mapping a JCA Thread to a Siebel Server Task and Log File

From the JCA logging information, you can find the Siebel Server task and log file, which can be useful in diagnosing threads that use large amounts of CPU time.

To map a JCA thread to a Siebel Server task and log file

1. Examine the JCA log file to find the high-CPU thread, for example:

```
[SIEBEL INFO] Thread[Servlet.Engine.Transports : 2,5,main] [2010-11-05
07:53:26.243] [SiebelConnection(507473761)] Opened a new connection to Siebel
(Siebel session : siebel.tcpip.none.none://myserver.example.com:2321/esblp01/
SCCObjMgr_enu/!10.6373.3ba70.465c2246)
```

   The Siebel session URL takes the following form:

```
siebel[.transport][.encryption][.compression]://host[:port]/EnterpriseServer/
AppObjMgr_lang/!AppObjMgrID.ProcessID.TaskID.timestamp
```

where the Application Object Manager ID, process ID, task ID, and timestamp are represented by hexadecimal numbers.

2. Use the Siebel session URL to find the following parameters, converting hexadecimal numbers to decimal:

Parameter	Example
Host	myserver.example.com
Siebel Enterprise Server	esblp01
Application Object Manager_lang	SCCObjMgr_enu
Application Object Manager ID	10 (16 decimal)
Task ID	3ba70 (244336 decimal)

3. Find the corresponding Siebel Server log file, which is in the `SIEBEL_SERVER_ROOT/log` directory:
   - Windows:

     ```
     AppObjMgr_lang_AppObjMgrID_taskID.log
     ```

     For example:

     ```
     SCCObjMgr_enu_0016_244336.log
     ```

   - UNIX:

     ```
     AppObjMgr_lang_taskID.log
     ```

     For example:

     ```
     SCCObjMgr_enu_244336.log
     ```

9 EAI DLL and EAI File Transports

EAI DLL and EAI File Transports

This chapter discusses the EAI DLL Transport and EAI File Transport business services. It includes the following topics:

- *About the EAI DLL Transport*
- *About the EAI File Transport*

About the EAI DLL Transport

You use the EAI DLL Transport when you want to call a function that exists in an external DLL. You must know the exported function in the DLL that you want to invoke. You specify the EAI DLL Transport as one of the business services in your workflow.

> **Note:** The EAI DLL Transport only accepts String type as input or output to the external DLL. The external function also must return String type.

The following topics are discussed here:

- *EAI DLL Transport Methods*
- *EAI DLL Transport Parameters*
- *Creating a DLL to Call a Function in an External DLL*

EAI DLL Transport Methods

The EAI DLL Transport supports sending messages using the following methods:

- Send
- SendReceive

EAI DLL Transport Parameters

Use the Send or SendReceive method as needed when you want to pass data from the Siebel Database to an external system. These methods require an input property set. In addition to the common parameters described in *EAI Transports and Interfaces Overview*, the EAI DLL Transport takes the parameters presented in the following table

Argument	Description
DLLName	Name of the (request/response) DLL.

Argument	Description
ExternalFunction	Function in the DLL to invoke.
Return Value	The return value from the function called. This value is an output property.

Calling a Function in an External DLL

The following procedure shows how to call a function in an external DLL.

To call a function in an external DLL

1. Create a workflow.

   Note: For details on creating workflows, see *Siebel Business Process Framework: Workflow Guide* .

2. Set the first business service, after the Start, to use the EAI DLL Transport. Usually, this object is named Send.
3. Double-click to set the input properties for the EAI DLL Transport.
4. Select a method, either Send, or Send and Receive Response.
5. Select the input arguments that you want to use from the list that appears (the arguments are described in the table in *EAI DLL Transport Parameters*.
6. Enter any output arguments required and save your work.

Creating a DLL to Call a Function in an External DLL

The following procedure illustrates how to create a DLL to use the EAI DLL Transport business service to call a function in an external DLL.

As of Siebel Innovation Pack 2014, a new mechanism is provided to free memory allocated. The creator of the external DLL can now expose additional API functions to free memory. Two new business service method arguments, DLLExternalFunction and DLLExternalFunctionFreeMemory, are added to the Send and SendReceive methods. Both arguments are optional input arguments.

- To use the new mechanism for memory deallocation, you must use both of these arguments together: DLLExternalFunction and DLLExternalFunctionFreeMemory.
- Customers can optionally use the old approach of only exposing the old argument, ExternalFunction, instead of exposing the new memory freeing API functions. If you continue to use ExternalFunction instead of the new arguments, then the old mechanism is used for memory deallocation. With the old mechanism, failure might occur when the EAI DLL Transport business service performs the memory deallocation.

The signature for the new memory freeing function would resemble the following:

```
extern "C" int __declspec(dllexport) TestFree(void* Value)
```

To create a DLL

1. Open a VC++ project by choosing the Open menu, then New.
2. Select a Win32 Dynamic Link Library and give a name to the project, such as MyDLL.
3. In the next dialog box, select the option Simple dll project.

   The following files are created by default:

- MyDLL.cpp
- StdAfx.h
- StdAfx.cpp

4. Make the following changes in the StdAfx.h and Main.cpp files and check the results in the process simulator:

```
// MyDLL.cpp : Defines the exported functions for the DLL application.
//

#include "stdafx.h"


#include <string.h>
#include <stdio.h>
#include <io.h>
#include <malloc.h>


extern "C" int __declspec(dllexport) TestEAI(const XMLDataBuf* pValue, XMLDataBuf* Value)
{

 FILE *fp = NULL;
 int retf = 0;
 int rc = 0;

 if ((fp = fopen("testeai.txt", "wb")) != NULL)
 {
 fprintf(fp, "Before test");
 fwrite(pValue->pData, sizeof(char), (size_t)pValue->nLength, fp);
 fprintf(fp, " After Test");
 fclose(fp);
 }
 else return -1;

 if ((fp = fopen("testeai.txt", "rb")) != NULL)
 {
 rc = (int)_filelength(_fileno(fp));
 Value->pData = (void *)malloc((size_t)(rc + 1));
 rc = (int)fread(Value->pData, sizeof(char), (size_t)rc, fp);

 fclose(fp);
 Value->nLength = rc;
 ((char*)Value->pData)[rc] = (char)NULL;
 }
 else return -2;

 return rc;

}

extern "C" int __declspec(dllexport) TestFree(void* Value)
{
 if(Value != NULL)
 {
 free (Value);
 Value = NULL;
 }

 return 0;

}
```

About the EAI File Transport

The EAI File Transport helps move data between a Siebel application and an external file.

> **Note:** The EAI File Transport is different from EAI XML Read from File. The EAI XML Read from File uses a Siebel Message in Hierarchical format as the output property. When reading in data, the EAI File Transport uses a process property with Data Type of Binary as the output property by default; if CharsetConversion is set, then it uses a string output property instead.

The following topics are discussed here:

- *EAI File Transport Methods*
- *Using the EAI File Transport Methods*
- *Generating Unique Filenames*
- *EAI File Transport Parameters*
- *Enabling Write Access for the EAI File Transport*
- *EAI File Transport Named Subsystem*

EAI File Transport Methods

The EAI File Transport supports two transport modes: sending messages and receiving messages. It uses the following methods:

- Send
- SendReceive
- Receive
- ReceiveDispatch
- ReceiveDispatchSend

Using the EAI File Transport Methods

You create a workflow to use the EAI File Transport, defining and refining the workflow as needed to meet your unique business requirements.

To create a workflow using the EAI File Transport

1. Create a workflow in Siebel Tools.

   > **Note:** For details on creating workflows, see *Siebel Business Process Framework: Workflow Guide* .

2. Set up a step in the workflow to use the EAI File Transport. Usually, this object is named Send.
3. Double-click to set the input properties for the EAI File Transport.
4. Select a method that fits your business needs.

5. Select the input arguments that you want to use from the list of arguments. The full list is presented in the table in *EAI File Transport Parameters*.
6. Enter any output arguments required and save your work.

Generating Unique Filenames

When using the EAI File Transport, you can have the system generate unique file names for you, as needed. One way is to specify the directory name only. The other way is to include $$ in the filename.

Note: If a directory is not specified when using the EAI XML Write to File, EAI XML Read from File, or the EAI File Transport business service, then the FileName input argument defaults to the directory where the Siebel application is running.

- **Directory Only.** To generate the unique file name, only enter the directory name. For example, instead of specifying the filename as `d:\data\record1.xml`, just specify `d:\data`. For every call of the workflow, a unique name is generated in the directory. To find out the file name generated, specify FileName as an output argument for the File Transport Workflow Step.
- **Using $$.** For generating filenames based on the $$ wildcard, specify the filename in the form `d:\data\record$$.xml`. At run time, Siebel application replaces the $$ with a unique row ID, for example:

  ```
  d:\data\record3-149.xml
  ```

Note: The file name generated by using $$ is not returned as the output filename property.

EAI File Transport Parameters

In addition to the common parameters presented in *Common EAI Transport Parameters*, the EAI File Transport takes the parameters presented in the following table. These parameters can be specified as service method arguments, subsystem parameters, or user properties.

Display Name	Parameter	Description
Append To File	AppendToFile	Default is False. A value of True means that, if the file exists, then the method appends the message to the existing file. A value of False specifies that the method overwrites any existing file.
Delete File after Receive	DeleteFile	Default is False. A value of True means that an attempt is made to delete the file after receiving it. If permissions prevent deletion, then no error is given, but the information is traced.
File Name	FileName	The name of the file to be received by the file transport. For the Send method, if a file name is not provided, then a random name is used for the output file. You must specify an explicit path for file name. You can also use `$$` as the wildcard symbol in the file name. For example, if you specify a file

Display Name	Parameter	Description
		name of "`file$$.xml`", then Siebel CRM creates files like `file1-134.xml`, `fileA25.xml`, and `file242_12B.xml`. For the Receive method, a specific file name must be provided. The use of wildcards such as `$$` is not allowed. The source file is deleted upon receiving if DeleteFile is set to True. If DeleteFile is set to False (the default), then the source file is not deleted.
Response File Name	RespFileName	Name of the file containing the response when using the SendReceive Method.
Sleep Time	FileSleepTime	The timeout interval on receive calls, in milliseconds. This specifies the maximum amount of time that the service waits for a response. Default is 20000 milliseconds.

Enabling Write Access for the EAI File Transport

The EAIFileTransportFolders parameter allows you to enable write access for the EAI File Transport for specific folders within the Siebel file system. The EAIFileTransportFolders parameter can be set at the enterprise or server level as a semicolon-separated list.

By default, the Siebel temporary folder, `SIEBSRVR_ROOT\TEMP`, is a permitted folder and is not required to be explicitly configured with the EAIFileTransportFolders parameter. If the parameter is not configured, then writing is allowed only to the Siebel temporary folder; any attempt to write a file to a folder other than the Siebel temporary folder fails.

CAUTION: Do not allow write access to the `SIEBSRVR_ROOT\BIN` folder. Write access to the `BIN` folder allows anyone to overwrite Siebel system DLL files.

Configuring the EAIFileTransportFolders Parameter at the Enterprise Level

You use the srvrmgr utility to configure the EAIFileTransportFolders parameter at the enterprise level.

To configure the EAIFileTransportFolders parameter at the enterprise level

- Use the following command in srvrmgr:

```
change ent param EAIFileTransportFolders=\\fileserver\fs1;\\fileserver2\fs2
```

Configuring the EAIFileTransportFolders Parameter at the Server Level

You use the srvrmgr utility to configure the EAIFileTransportFolders parameter at the server level.

To configure the EAIFileTransportFolders parameter at the server level

- Use the following command in srvrmgr:

```
change param EAIFileTransportFolders=\\fileserver\fs1;\\fileserver2\fs2 for server servername
```

Configuring the EAIFileTransportFolders Parameter in the Application Configuration File

You add a new section to the application configuration file to configure the EAIFileTransportFolders parameter.

To configure the EAIFileTransportFolders parameter in the application configuration file

1. Open the application configuration file, such as uagent.cfg, in a text editor.
2. Add the following section:

```
[EAIFileTransportConfigSubsys]
EAIFileTransportFolders = \\fileserver\fs1;\\fileserver2\fs2
```

EAI File Transport Named Subsystem

The EAI File Transport can read parameters from a named subsystem. For the EAI File Transport, the named subsystem type is FileTranspSubsys.

The following example shows how to use the FileTranspSubsys named subsystem with EAI File Transport business service methods.

Receiving a Message and Writing It to a File

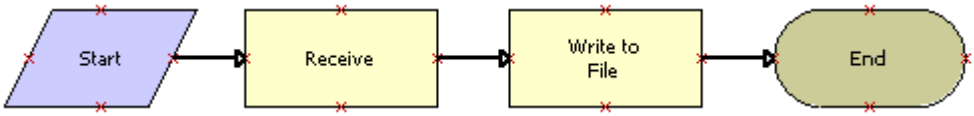
This example uses the Receive method of the EAI File Transport business service to receive a message as a file, then it uses the Send method of the EAI File Transport business service and the FileTranspSubsys named subsystem to write the message to a file.

To receive a message and write it to a file

1. Define an EAI File Transport named subsystem, for example:

```
create named subsystem FileConnSubsys_sub for subsystem FileTranspSubsys with
FileName="D:\temp\FileOut.txt", AppendToFile=true
```

2. Create a workflow containing the following steps, as shown in the following image:
   a. Receive
   b. Write to File



> **Note:** For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide* .

3. Define the following process properties:

Name	Data Type	In/Out	Default String
BinaryMsg	Binary	In/Out	Not applicable
Error Code	String	In/Out	Not applicable
Error Message	String	In/Out	Not applicable
Object Id	String	In/Out	Not applicable
Process Instance Id	String	In/Out	Not applicable
Siebel Operation Object Id	String	In/Out	Not applicable

4. Set up the first business service step to use the EAI File Transport business service with the Receive method and the following input and output arguments:

Input Argument	Type	Value
FileName	Literal	D:\temp\InputToFile.txt

Property Name	Type	Output Argument
BinaryMsg	Output Argument	<Value>

5. Set up the second business service step to use the EAI File Transport business service with the Send method and the following input arguments:

Input Argument	Type	Value	Property Name
<Value>	Process Property	Not applicable	BinaryMsg
ConnectionSubsystem	Literal	FileConnSubsys_sub	Not applicable

10 Transcode Service Business Service

Transcode Service Business Service

This chapter discusses the Transcode Service business service. It includes the following topics:

- *About the Transcode Service Business Service*
- *Transcode Service Business Service Methods*
- *Transcode Service Business Service Examples*

About the Transcode Service Business Service

The Transcode Service business service converts data from one character-set encoding to another. It can also validate conversions before they are performed.

The conversion implementation is portable, and does not rely on the operating system or any third-party products for codepage definitions. Supported error detection includes output-buffer overflow, memory-allocation failure, invalid data in the input encoding stream, and substitution in the output encoding stream.

Note: Windows fallback (that is, approximate) conversions are not supported.

The Transcode Service business service provides data conversion and validation of conversion between the following encodings:

- ASCII
- 874 (Thai)
- 932 (Japanese)
- 936 (Simplified Chinese)
- 949 (Korean)
- 950 (Traditional Chinese)
- 1250
- 1251
- 1252 (Western European)
- 1253
- 1254
- 1255
- 1256
- 1257
- 1258
- UTF-8

- UTF-16LE
- UTF-16BE
- UTF-16

For a list of the languages supported by Siebel CRM, and the supported code pages for each database, see 1513102.1 (Article ID) on My Oracle Support. See also *Siebel Global Deployment Guide* and see the Certifications tab on My Oracle Support. For information about the Certifications application, see 1492194.1 (Article ID) on My Oracle Support.

Transcode Service Business Service Methods

The Transcode Service business service has two methods:

- *Convert Method*
- *Validate Method*

Convert Method

This method converts the value in the input property set to the target encoding in the output. You use this method when data enters or leaves Oracle's Siebel CRM and conversion is required so that the next software component in the processing chain can recognize the data.

The Convert method has the method arguments shown in the following table.

Method Argument	Required	Description
<Value>	Yes	Data to convert.
ConversionMode	Yes	The mode can be StringToEncoding, EncodingToString, or EncodingToEncoding.
SourceEncoding	No	Encoding from which data is converted. Required for the EncodingToString and EncodingToEncoding modes.
TargetEncoding	No	Encoding to which data is converted. Required for the StringToEncoding and EncodingToEncoding modes.
IgnoreConversionErrors	No	To ignore character conversion errors (invalid-character errors or substitution errors), set IgnoreConversionErrors to TRUE. **Note:** This argument is not shown in Siebel Tools.

Validate Method

To avoid problems associated with relying on third-party applications to convert data, you can use the Validate method of the Transcode Service business service. The Validate method confirms the input property set hierarchy or the value of the input property set. You can use this method to check that a character is valid within a particular character set before performing the conversion. You can choose not to send the data to the external application if validation fails.

If validation fails, then the Transcode Service business returns a client-side error code (Error Code). The log file contains detailed information about what went wrong, including the failure type, first position in the input, and where conversion failed.

The Validate method has the method arguments shown in the following table.

Method Argument	Required	Description
ValidationMode	No	Can be Value or left blank. If the mode is Value, then only <Value> is validated. Otherwise, the entire property set hierarchy is validated.
SourceEncoding	No	Encoding from which data is converted. Required when ValidationMode is set to Value and the input value contains binary data. Conversion from binary data in SourceEncoding to binary data in TargetEncoding is implied.
TargetEncoding	Yes	Encoding to which data is converted.
<Value>	No	If <Value> is used (ValidationMode is set to Value), then only it is validated. Otherwise, the entire property set hierarchy is validated.
SiebelMessage	No	If the validation is for a hierarchy of type Siebel Message, for example, the output of the EAI Siebel Adapter, then this argument refers to the property set. **Note:** This argument is not shown in Siebel Tools.
XMLHierarchy	No	If the validation is for an XML hierarchy, for example, the output of the ReadXMLHier method of the EAI XML Read from File business service method, then this argument refers to the property set. **Note:** This argument is not shown in Siebel Tools.

Transcode Service Business Service Examples

The following examples show how to use the Validate and Convert methods of the Transcode Service business service:

- *Using the Validate Method*
- *Using the Convert Method*

Using the Validate Method

The following examples demonstrate the use of the Validate method of the Transcode Service business service:

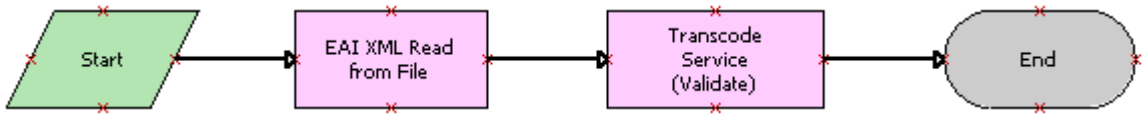
- *XML Hierarchy Example*
- *Siebel Message Example*

XML Hierarchy Example

In this workflow example, a file encoded in codepage 932 (Japanese) is read into an XML hierarchy, then validated for conversion into codepage 1252 (Western European).

To create the validation workflow (XML hierarchy example)

1. Create a workflow containing the following steps, as shown in the following image:
   a. EAI XML Read from File
   b. Transcode Service (Validate)



> **Note:** For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide* .

2. Define the following process properties:

Name	Data Type	In/Out
Error Code	String	In/Out
Error Message	String	In/Out
Siebel Operation Object Id	String	In/Out
XMLHier	Hierarchy	In/Out

3. Set up the first business service step to use the EAI XML Read from File business service with the ReadXMLHier method and the following input and output arguments:

Input Argument	Type	Value
FileName	Literal	c:\JPN_JIS.xml

Property Name	Type	Output Argument
XMLHier	Output Argument	XMLHierarchy

4. Set up the second business service step to use the Transcode Service business service with the Validate method and the following input arguments:

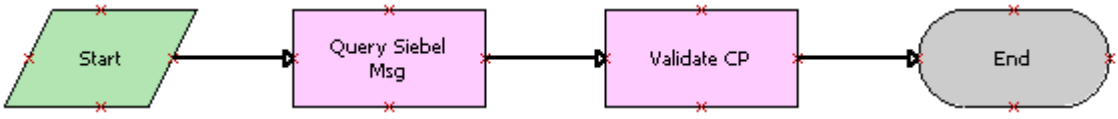
Input Argument	Type	Value	Property Name
SourceEncoding	Literal	CP932	Not applicable
TargetEncoding	Literal	CP1252	Not applicable
ValidationMode	Literal	Not applicable	Not applicable
XMLHierarchy	Process Property	Not applicable	XMLHier

Siebel Message Example

In this workflow example, an account record is read from an integration object by the EAI Siebel Adapter as a Siebel Message, then validated for conversion from UTF-8 (Unicode) to codepage 1252 (Western European).

To create the validation workflow (Siebel message example)

1. Create a workflow containing the following steps, as shown in the following image:
   a. Query Siebel Msg
   b. Validate CP



> **Note:** For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide* .

2. Define the following process properties:

Name	Data Type	In/Out
Error Code	String	In/Out
Error Message	String	In/Out
Object Id	String	In/Out
Process Instance Id	String	In/Out
Siebel Operation Object Id	String	In/Out
SiebelMsg	Hierarchy	In/Out

3. Set up the first business service step to use the EAI Siebel Adapter business service with the Query method and the following input and output arguments:

Input Argument	Type	Value	Property Name
OutputIntObjectName	Literal	Sample Account	Not applicable
PrimaryRowId	Process Property	Row ID of the account record	Object Id

Property Name	Type	Output Argument
SiebelMsg	Output Argument	SiebelMessage

4. Set up the second business service step to use the Transcode Service business service with the Validate method and the following input arguments:

Input Argument	Type	Value	Property Name
SourceEncoding	Literal	UTF-8	Not applicable
TargetEncoding	Literal	CP1252	Not applicable
ValidationMode	Literal	Not applicable	Not applicable

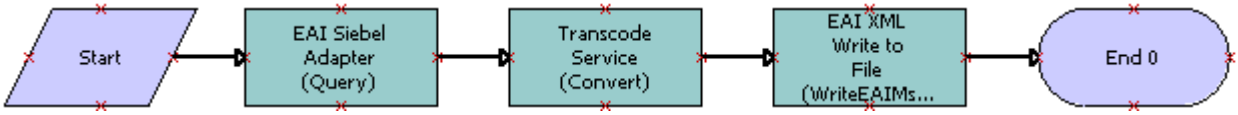
Input Argument	Type	Value	Property Name
SiebelMessage	Process Property	Not applicable	SiebelMsg

Using the Convert Method

The following workflow example demonstrates the use of the Convert method of the Transcode Service business service. An account record is read from an integration object by the EAI Siebel Adapter as a Siebel Message, converted from UTF-8 (Unicode) to codepage 932 (Japanese), and then written to an XML file.

To create the conversion workflow (using the Convert method)

1. Create a workflow containing the following steps, as shown in the following image:
   a. EAI Siebel Adapter (Query)
   b. Transcode Service (Convert)
   c. EAI XML Write to File (WriteEAIMs...)



> **Note:** For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide* .

2. Define the following process properties:

Name	Data Type	In/Out
Error Code	String	In/Out
Error Message	String	In/Out
Object Id	String	In/Out
Process Instance Id	String	In/Out
Siebel Operation Object Id	String	In/Out
SiebelMsg	Hierarchy	In/Out

Name	Data Type	In/Out
SiebelMsgJPN	Hierarchy	In/Out

3. Set up the first business service step to use the EAI Siebel Adapter business service with the Read Siebel Msg method and the following input and output arguments:

Input Argument	Type	Value	Property Name
OutputIntObjectName	Literal	Sample Account	Not applicable
PrimaryRowId	Process Property	Row ID of the account record	Object Id

Property Name	Type	Output Argument
SiebelMsg	Output Argument	SiebelMessage

4. Set up the second business service step to use the Transcode Service business service with the Convert method and the following input and output arguments:

Input Argument	Type	Value	Property Name
SourceEncoding	Literal	UTF-8	Not applicable
TargetEncoding	Literal	CP932	Not applicable
ConversionMode	Literal	EncodingToEncoding	Not applicable
<Value>	Process Property	Not applicable	SiebelMsg

Property Name	Type	Output Argument
SiebelMsgJPN	Output Argument	<Value>

5. Set up the third business service step to write the converted integration object hierarchy to an XML file using the EAI XML Write to File business service with the WriteEAIMsg method. This step requires the following input arguments:

Input Argument	Type	Value	Property Name
FileName	Literal	File to write, for example, d:\temp \acct_record_JPN.xml	Not applicable
<Value>	Process Property	Not applicable	SiebelMsgJPN