Transports and Interfaces: Siebel Enterprise Application Integration

Version 8.0, Rev. C
February 2011

ORACLE®

Contents

Chapter 5: EAI Java Business Service

Chapter 6: EAI JMS Transport

Chapter 7: EAI HTTP Transport

Chapter 8: Integrating Siebel Business Applications with Java Applications

Chapter 9: EAI DLL and EAI File Transports

Chapter 10: Transcode Service Business Service

Index

1 What's New in This Release

What's New in Transports and Interfaces: Siebel Enterprise Application Integration, Version 8.0, Rev. C

Table 1 lists changes described in this version of the documentation to support release 8.0 of Oracle's Siebel software.

Table 1.   New Product Features in Transports and Interfaces: Siebel Enterprise Application Integration, Version 8.0, Rev. C

Topic	Description
"Inbound Methods for a Transport Business Service" on page 17	Modified topic. Added a caution not to set the OM - Preload SRF Data parameter to True for single-threaded Siebel Application Object Manager components, such as Siebel Enterprise Application Integration (EAI) receivers.
"Features Not Supported for Use with the Siebel JMS Transport" on page 64	Modified topic. The use of Secure Sockets Layer (SSL) with the Siebel JMS Transport is not supported because the Java Message Service (JMS) standard is not bound to transport layers. For information on enabling and using SSL with the Siebel JMS Transport, contact the vendor of your JMS system.
"Roadmap for Configuring JMS Messaging Between Siebel Business Applications and Oracle SOA Suite Using Oracle Advanced Queuing" on page 81	Modified topic. Revised it to use Oracle Enterprise Manager for resource provider configuration and to correct the list of required Java Archive (JAR) files.
"Mapping a JCA Thread to a Siebel Server Task and Log File" on page 159	New topic. You can find the Siebel Server task and log file from the Java EE Connector Architecture (JCA) logging information.

What's New in Transports and Interfaces: Siebel Enterprise Application Integration, Version 8.0, Rev. B

Table 2 lists changes described in this version of the documentation to support release 8.0 of the software.

Table 2.   New Product Features in Transports and Interfaces: Siebel Enterprise Application Integration, Version 8.0, Rev. B

Topic	Description
"Increasing the Maximum Message Length on IBM WebSphere MQ" on page 30	New topic. If you are handling large messages, you can change the MaxMsgLength queue manager and queue attributes.
"Receiving Inbound Messages with MSMQ Receiver" on page 46	Modified topic: ■ Removed the Receiving Messages from MSMQ topic. ■ Added MSMQ Receiver named subsystem steps to the examples. ■ Removed the EAI MSMQ Transport steps from the workflow examples because the MSMQ Receiver, which uses the EAI MSMQ Transport, is defined in the receiver subsystem steps.
"Prerequisites for Implementing a Java Business Service" on page 51 "Restrictions for Implementing JBS" on page 58	Modified topic. You cannot use 64-bit Java Virtual Machines (JVMs) with Java business services in Oracle's Siebel Business Applications, which are 32-bit.
"About Synchronous and Asynchronous Invocation" on page 62	Modified topic. Added a note that in-process re-entrance is not supported for the JMS Receiver.
"About the JMS Receiver" on page 73	Modified topic. Added details on JMS Receiver parameters.
"Roadmap for Configuring JMS Messaging Between Siebel Business Applications and Oracle SOA Suite Using Oracle Advanced Queuing" on page 81	New topic. This topic describes a process for communicating between Siebel Business Applications and Oracle SOA Suite using Java Message Service (JMS) as the transport mechanism.
"Troubleshooting for the JMS Transport" on page 105	Modified topic: ■ Expanded the descriptions of the CheckJNDIObjects and CheckJMSServer debugging methods. ■ Added a note that while the classpath is limited to 1024 characters, it might be truncated when displayed in the user interface and command-line interface.

Table 2.   New Product Features in Transports and Interfaces: Siebel Enterprise Application Integration, Version 8.0, Rev. B

Topic	Description
"Selecting the Appropriate Business Service for HTTP" on page 108	Modified topic. Do not use the Web Engine HTTP TXN business service for inbound HTTP transport sessions.
"Using the EAI HTTP Transport for Inbound Integration" on page 116	Modified topic: ■ Clarified how session cookies are handled in session mode. ■ Clarified where special characters must be encoded for use in URLs and where encoding is not required.
"Providing Client Certificate Information for SSL Mutual Authentication" on page 133	New topic. A Siebel eScript example is shown, which provides client-authentication parameters.
"EAI File Transport Named Subsystem" on page 166	New topic. Added information on the FileTranspSubsys named subsystem and an example of its use.
Chapter 10, "Transcode Service Business Service"	New chapter. The Transcode Service business service converts data from one character-set encoding to another. It can also validate conversions before they are performed.

What's New in Transports and Interfaces: Siebel Enterprise Application Integration, Version 8.0, Rev. A

Table 3 lists changes described in this version of the documentation to support release 8.0 of the software.

Table 3.   New Product Features in Transports and Interfaces: Siebel Enterprise Application Integration, Version 8.0, Rev. A

Topic	Description
"Using Named Subsystems for Transport Parameters" on page 18	Modified topic. Added a note that you must specify named subsystem parameters by their aliases.
"Exposing MQMD Headers as Properties" on page 25	Modified topic. Updated the lists of supported inbound and outbound headers. Corrected the description of the MsgFlags MQMD message header in Table 9 on page 27.
"About Defining Integration Objects" on page 39	Removed the procedure from this topic. The creation of integration objects is documented in *Integration Platform Technologies: Siebel Enterprise Application Integration*.

Table 3.   New Product Features in Transports and Interfaces: Siebel Enterprise Application Integration, Version 8.0, Rev. A

Topic	Description
"Prerequisites for Implementing a Java Business Service" on page 51	Modified topic. Added complete Java Runtime Environment (JRE) library paths for various operating systems.
"Troubleshooting the Java Business Service" on page 59	Modified topic. Added the names of Windows and UNIX utilities that you can use to troubleshoot CLASSPATH errors.
"Configuring the JMS Transport" on page 72	Modified topic. Added a note that you can have only one Java Virtual Machine (JVM) loaded in a process, and therefore only one JVM subsystem. If you try to load more than one JVM subsystem, an error will occur. If you want multiple JVM subsystems, then you must configure additional components.
"Configuring Against TIBCO" on page 103	Returned this topic to the documentation, because TIBCO is supported in Siebel Business Applications version 8.0 and higher.
"Using POST and GET" on page 109	Modified topic. Added descriptions of the POST and GET methods.
"EAI HTTP Transport Method Arguments" on page 110	Removed the HTTPIsSecureConn parameter. This parameter is not necessary, because the Web server automatically uses the Secure Sockets layer (SSL) protocol when the URL begins with https://.
"EAI HTTP Transport for Inbound Messages" on page 123	Modified topic. Added a note that you can make the business service available by deploying it to the run-time database from Siebel Tools, as well as by compiling the SRF file. Clarified the note about the HTTP response for inbound requests. If the process properties are set as In/Out (the default), the values will appear as HTTP headers on the HTTP response from the Siebel Server.
"Specifying Parameters as Business Service User Properties" on page 127	Modified topic. You can deploy the business service to the run-time database from Siebel Tools.
"Connection Parameters for the SiebelDataBean" on page 148	Modified topic. Added the maximum values for siebel.conmgr.txtimeout and siebel.conmgr.sesstimeout. Changed their values in the sample siebel.properties file to more realistic numbers.
"About JCA Logging" on page 156	New topic. Enhancements have been made to J2EE Connector Architecture (JCA) logging in Siebel Business Applications version 8.0 and higher.

Additional Changes

Workflows are created in Siebel Tools in version 8.0 and higher. Each procedure in this guide that describes how to create a workflow has been revised to reflect this change.

Changes that were made to *Transports and Interfaces: Siebel Enterprise Application Integration* for Siebel Business Applications version 8.1 and higher have also been made in this guide where they apply to version 8.0.

2 EAI Transports and Interfaces Overview

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

This chapter includes the following topics on these mechanisms:

About EAI Transports

Transports allow Siebel Business Applications 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, 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, 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 if needed. For information on the Transcode Service business service, see Chapter 10, "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, and HTTP. EAI Transports allow Siebel Business Applications 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, 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 on these transports, see Chapter 3, "EAI MQSeries Server Transport."

■ EAI MSMQ Transport. For information on this transport, see Chapter 4, "EAI MSMQ Transport."

■ EAI HTTP Transport. For information on this transport, see Chapter 7, "EAI HTTP Transport."

■ EAI DLL Transport and EAI File Transport. For information on these transports, see Chapter 9, "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 on transport re-entrance, see "About EAI MQSeries Transport Re-Entrance" on page 32.

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" on page 17

■ "Inbound Methods for a Transport Business Service" on page 17

For each method, there are a number of common parameters, as shown on Table 5 on page 20, 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 need a response. This is an asynchronous request method (with the exception of the EAI HTTP Transport, which expects a correct HTTP response), because the Siebel application does not need to wait for a response before continuing with the process.

■ **Send and Receive (SendReceive).** Sends a message from the Siebel application when the Siebel application needs to receive a response before continuing. This is a synchronous request and response method, because it requires 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) on top of the inbound methods that run as Siebel Server tasks. When running an EAI receiver such as the SAP IDOC Receiver, MQSeries Server, or MSMQ Receiver (using the methods ReceiveDispatch or ReceiveDispatchSend), if the dispatch service has an error, the receiver shuts down. Check the Status column on the Component Tasks for details about the cause of the error.

CAUTION: Do not set the OM - Preload SRF Data parameter to True for single-threaded Application Object Manager components, which include EAI receivers, such as MQSeries Server Receiver. If OM - Preload SRF Data is True, the Siebel Server will try to initialize the EAI receiver twice, causing the EAI receiver to fail.

Using Named Subsystems for Transport Parameters

Named subsystems are groupings of defined enterprise parameters that are stored in the Siebel Gateway Name Server. 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 specify named subsystem parameters by the values of their Alias fields in the Profile Parameters list.

NOTE: Parameters specified in business service user properties no longer work *as is*. You need to create named subsystems and specify the parameters for the subsystems. Then, you need to specify the named subsystems you created, for example, as business service user properties in a workflow or through scripting. Note that business service user properties work for the SAP Connector business services.

The following subtopics are discussed in this topic:

■ "Rules of Precedence for Parameter Specification" on page 18

■ "Common EAI Transport Parameters" on page 19

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, the business service user property takes precedence over the run-time arguments.

NOTE: For additional information on 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, Siebel EAI will always look 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, the run-time specification will be 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 need to create named subsystems for data handling and connection parameters, as presented in Table 4.

Table 4.   Dispatch Parameter Usage

When You Need to...	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 Table 5 on page 20. These parameters are common to every transport method. After you create the named subsystems, you then need to specify these named subsystems as parameters in the service method argument or the business service user property.

Table 5.   Common Data Handling Parameters for Transport Methods

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. Default is None. Use the default value for this parameter for self-describing 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, 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 that not any arbitrary service may be designated to be 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.

Table 5.   Common Data Handling Parameters for Transport Methods

Parameter Name	Description
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, the transport service propagates the error.
Impersonate	Default is False. This parameter indicates whether or not the receiver will execute 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, the receiver will analyze the incoming XML document (<SiebelMessage> element) for the eaiusername and eaipassword credential attributes. If these credentials are found, the receiver will attempt 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, an error message will be 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 and MSMQ. If this parameter is set to False, the transaction support is turned off at the transport level. This setting means that if the transaction fails, then there will not be a 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 Chapter 8, "Integrating Siebel Business Applications with Java Applications."

Database-Level Interfacing

In addition to transports and object interfaces, Siebel Business Applications provide 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 on interface tables, see *Siebel Enterprise Integration Manager Administration Guide*.

3 EAI MQSeries Server Transport

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

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 Transport Business Service

The Siebel EAI MQSeries Server Transport provides a messaging solution to help you integrate data between Siebel Business Applications 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 need to 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 the "Outbound Methods for a Transport Business Service" on page 17 and "Inbound Methods for a Transport Business Service" on page 17.

The following topics are also described:

■   "EAI MQSeries Server Transport Named Subsystem" on page 29

NOTE: The EAI MQSeries Transport business service, which was provided in Siebel 6.x releases, is not available in Siebel 7.x releases and higher. Customers of previous Siebel versions using the name EAI MQSeries Transport must upgrade their workflows to use the EAI MQSeries Server Transport business service.

EAI MQSeries Server Transport Parameters

In addition to supporting the common transport parameters presented in Table 5 on page 20, the EAI MQSeries Server Transport uses the parameters shown in Table 6. These can be specified as either 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.

Table 6.   EAI MQSeries Server Transport-Specific Parameters

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, 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, 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. **NOTE:** Using an alias queue will work. However, since the alias queue does not have a backout queue defined, the receiver cannot roll back to the backout queue.

Table 6.   EAI MQSeries Server Transport-Specific Parameters

Argument	Display Name	Description
MqQueueManagerName	Queue Manager Name	Name of the MQSeries queue manager. If this parameter is not specified, 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 Table 9 on page 27.

In the outbound direction, that is, when a message is placed on a queue, the EAI MQ Server Transport supports the headers shown in Table 7 to be set by the caller:

Table 7.   Valid Outbound (Input) Headers

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

Table 7.   Valid Outbound (Input) Headers

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	In Siebel Business Applications version 8.1 and higher, the only settable value is MQRO_NONE. If Report is not set, it defaults to the binary OR product of MQRO_EXCEPTION_WITH_DATA | MQRO_EXPIRATION_WITH_DATA | MQRO_PAN | MQRO_NAN. In versions earlier than 8.1, Report has only the default value.
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, 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, 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 Table 8.

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

Table 8.   Examples of Input Arguments for Outbound MQMD Headers

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, 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 Table 8, MsgType is set to TestMsgHeader.

Table 9 summarizes the MQMD message headers that are exposed as properties in a property set.

Table 9.   MQMD Message Headers

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 data	Input and output
CorrelId	MQBYTE24	Correlation identifier	Output

Table 9.   MQMD Message Headers

Field	Data Type	Description	Input or Output Property?
Encoding	MQLONG	Numeric encoding of message data	Input and output
Expiry	MQLONG	Message lifetime	Input and output
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 put the message	Output
PutApplType	MQLONG	Type of application that put the message	Output
PutDate	MQCHAR8	Date when message was put	Output
PutTime	MQCHAR8	Tine when message was put	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	MQCHAR12	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 Chapter 2, "EAI Transports and Interfaces Overview."

■ For more information on 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, the business service will not raise an error but the TimedOut value will be true.

Dispatch Error Handling for the EAI MQSeries Server Transport

When using the ReceiveDispatch and ReceiveDispatchSend methods, you need to be aware of specific MQSeries behavior that 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 the following conditions are met, 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, MQRC_MSG_TOO_BIG_FOR_Q is returned. Similarly, if the message is too large for the queue manager, MQRC_MSG_TOO_BIG_FOR_Q_MGR is returned.

If you are handling large messages, 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 tries 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, 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 used as the memory segment number. Possible legal 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 run into this error even with the memory segment number set as 12 if the operating system has nondeterministically allocated segment 12 to the EAI MQSeries Server process ahead of the queue manager. If this is the case, a different segment number may need to 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, you can configure the AIX environment to run Siebel Server with less memory using environment variable LDR_CNTRL. After you have finished, follow the procedures in the preceding section.

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 Table 10.

Table 10.   Output Argument for Send Step

Type	Output Argument
Output Argument	MQSeries Message Identifier

This captures the Message IDs that 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, the workflow assumes that the error step or the error process will handle the error and the workflow will not send the error out. In order to capture the error, you need to insert a stop step into your workflow. Note that by putting in 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 will be the name of the process property you created in the previous step.

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

4 EAI MSMQ Transport

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

- About MSMQ on page 35
- Configuring the EAI MSMQ Transport Servers on page 37
- Configuring EAI MSMQ Transport for Various Send and Receive Scenarios on page 38

About 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 will still be 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" on page 35
- "Methods for Sending and Receiving Messages" on page 36
- "EAI MSMQ Transport Named Subsystems" on page 37

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 machine 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 may or may 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, 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 Chapter 2, "EAI Transports and Interfaces Overview." For more information on named subsystems, see *Siebel System Administration Guide*.

Configuring the EAI MSMQ Transport Servers

The instructions in this section are for configuring the EAI MSMQ Transport servers. Use a two-server setup, configured as listed in the following section. 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 *Siebel System Requirements and Supported Platforms* on Oracle Technology Network)

■ MSMQ Server

■ As many MSMQ queues as needed

■ Relevant ODBC driver

■ Siebel Server

■ Siebel Gateway Name Server

■ 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 *Siebel System Requirements and Supported Platforms* on Oracle Technology Network)

■ MSMQ Client

■ As many MSMQ queues as needed

■ The relevant ODBC driver

■ Siebel Server

■ Siebel Gateway Name Server

■ Siebel Web Client

NOTE: The MSMQ Server can reside on either machine. This functionality is independent of the underlying database. You can use any of the supported databases, including IBM DB2, DB2 UDB for z/OS, Oracle, 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" on page 38

■ "EAI MSMQ Transport Parameters" on page 39

■ "About Defining Integration Objects" on page 39

■ "Sending Outbound Messages with EAI MSMQ Transport" on page 40

■ "Receiving Inbound Messages with MSMQ Receiver" on page 46

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

Table 11 presents the parameters used for configuring the EAI MSMQ Transport.

Table 11.   EAI MSMQ Transport Parameters

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	Machine that owns the queue specified by the physical queue name.
MsmqRespQueueMachineName	Machine 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, the TimedOut argument in the Output Property set will be set to True.
IgnoreCorrelationId	Default is False. Set to ignore Correlation Id value on the inbound messages. If this flag is True, 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 4MB) 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 on 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, as shown in the following illustration.

The MSMQ queue you create will appear in the list of queues.
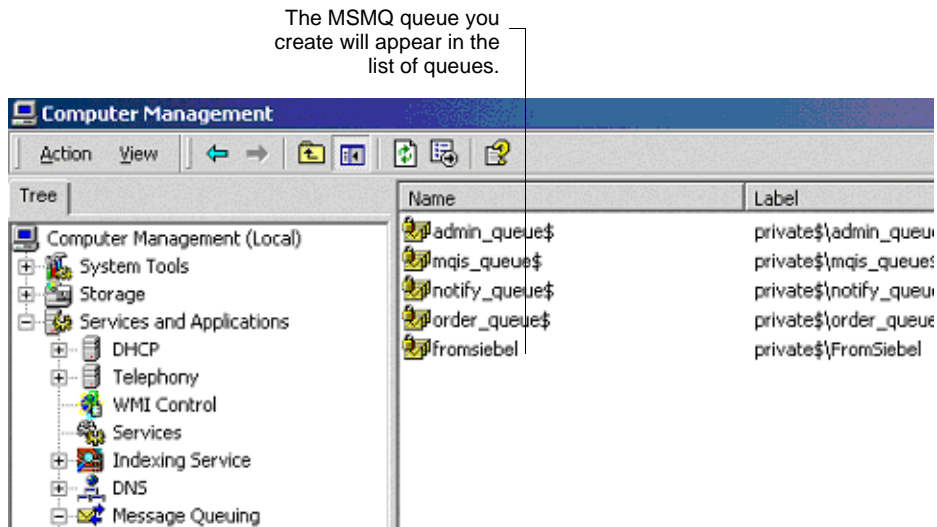
3 Set the queue to be Transactional.

NOTE: This flag allows Siebel Business Applications 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 In Siebel Tools, set up a workflow for sending a message to MSMQ. Define the flow as shown in the following figure:
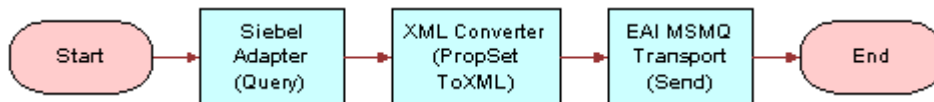
NOTE: For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide*.

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

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* Machine 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 a message will be in the fromSiebel queue and will contain 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 In Siebel Tools, set up a workflow for sending a message out and receiving a message in response using EAI MSMQ Transport. Define the flow as shown in the following figure:
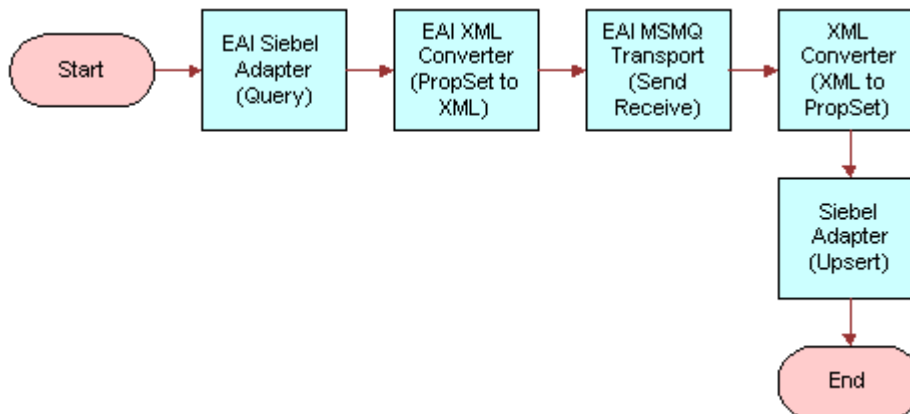
NOTE: For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide*.

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 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
MsmqPhysicalQueueName	Literal	fromsiebel	Not applicable	Not applicable
MsmqQueueMachineName	Literal	*SiebelServer1* Machine 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 machine 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 put a valid message for the Siebel application into 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 In Siebel Tools, set up a workflow for receiving and dispatching a message from MSMQ as shown in the following figure:

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 put a valid message for the Siebel application in 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 put a valid message for the Siebel application into 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 In Siebel Tools, set up a workflow for receiving and dispatching a message from MSMQ as shown in the following figure:
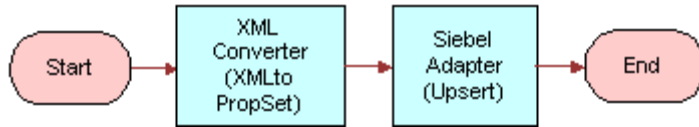
NOTE: For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide*.

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 put a valid message for the Siebel application into 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 will be in the queue specified by the MSMQRespQueueName and MSMQRespQueueMachineName arguments.

5 EAI Java Business Service

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

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, 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 by creating a 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.

Prerequisites 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

The named subsystem supplies the parameters to the JBS. There are three parameters:

■ **DLL.** The complete path of the JRE library, as shown in Table 12:

Table 12.   Complete JRE Library Paths for Various Operating Systems

Operating System	JRE Library	Typical Location on Server and Environment Variable Setting
AIX	libjvm.so	/usr/java5/jre/bin/j9vm Both /usr/java5/jre/bin and /usr/java5/jre/bin/j9vm must be included 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:/siebel/siebsrvr/mgmtagent/lib:/usr/java5/jre/bin:/usr/java5/jre/bin/j9vm ■ 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:/siebel/siebsrvr/mgmtagent/lib:/usr/java5/jre/bin:/usr/java5/jre/bin/j9vm
HP-UX	libjvm.sl	/opt/java1.4/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/java1.4/jre/lib/PA_RISC2.0; export SHLIB_PATH In siebmtshw: LD_PRELOAD=/opt/java1.4/jre/lib/PA_RISC2.0/server/libjvm.sl export LD_PRELOAD

Table 12.   Complete JRE Library Paths for Various Operating Systems

Operating System	JRE Library	Typical Location on Server and Environment Variable Setting
Linux	libjvm.so	/usr/java/jdk1.5.0_05/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/jdk1.5.0_05/jre/lib/i386/server: /usr/java/jdk1.5.0_05/jre/lib/i386
Solaris	libjvm.so	/usr/jdk/instances/jdk1.5.0/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/jdk1.5.0/jre/lib/sparc/server:/platform/SUNW,Sun-Fire-V440/lib:{LD_LIBRARY_PATH} ■ siebenv.sh: LD_LIBRARY_PATH=/usr/jdk/instances/jdk1.5.0/jre/lib/sparc/server:/platform/SUNW,Sun-Fire-V440/lib:{LD_LIBRARY_PATH};export LD_LIBRARY_PATH
Windows	jvm.dll	JDK installation directory No environment variable

NOTE: You cannot use 64-bit JVMs with Java business services in Siebel Business Applications, which are 32-bit.

■ **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 will not load successfully into the Application Object Manager without this option.

The following topics are also discussed here:

Creating a Java Subsystem by Using the Siebel Server Manager

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

```
create named subsystem JAVA for subsystem JVMSubSys with
DLL="D:\jdk1.5.0\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 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. Only do this with the Siebel Mobile and Developer Web Clients, and not the Siebel Server.

```
[JAVA]
DLL = D:\jdk1.5.0\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 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 top 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 bottom 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:\bea\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, DLL= `D:\jdk1.5.0\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.

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 /bin and /bin/j9vm directories. 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:/siebel/siebsrvr/mgmtagent/lib:/ usr/java5/jre/bin:/usr/java5/jre/bin/j9vm`

   Also, make sure that the LD_LIBRARY_PATH is set correctly.

■   **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/java1.5/jre/lib/PA_RISC2.0:/opt/java1.5/jre/lib/PA_RISC2.0/ server:${SHLIB_PATH}`

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

■   **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 Chapter 6, "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" on page 56

■ "About Implementing a Business Service in Java" on page 56

■ "About Exception Handling for the Java Business Service" on page 57

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).

   NOTE: Checking the Cache column when you are creating the new Java business service will cause the same Java object to be reused by subsequent invocations within the same session. (See "About the Lifecycle of a Java Business Service" on page 57.)

2 Add a business service user property named @class, whose value is the fully qualified name of the Java class (for example, com.mycompany.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 may be used for programmatic handling in Siebel eScript when the business service is called. Both the error code and the error message will be 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 will be employed.

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

About the Lifecycle of a 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 will 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.mycompany.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());
        }
    }
}
```

Restrictions for Implementing JBS

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

■ You cannot use 64-bit JVMs with Java business services in Siebel Business Applications, which are 32-bit.

■ Each business service method invocation is atomic and stateless.

■ The explicit creation of threads is discouraged. It is not recommended that customers invoke a multithreaded 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, 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 on filemon, see:

http://www.microsoft.com

■ UNIX: truss/strace

6 EAI JMS Transport

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

- About the EAI JMS Transport Business Service on page 61
- About Synchronous and Asynchronous Invocation on page 62
- About the JMS Publish-and-Subscribe Model on page 62
- About Operations (Methods) of the JMS Transport on page 63
- Features Not Supported for Use with the Siebel JMS Transport on page 64
- About JMS Message Types on page 64
- About Sending and Receiving XML on page 65
- About Multistep Operations Within a JMS Session on page 65
- About Undeliverable Messages in JMS Transport on page 66
- Detailed Input and Output Specifications for the EAI JMS Transport on page 66
- Configuring the JMS Transport on page 72
- Receiving, Dispatching, and Sending JMS Messages on page 78
- Roadmap for Configuring JMS Messaging Between Siebel Business Applications and Oracle SOA Suite Using Oracle Advanced Queuing on page 81
- Creating JMS Queues on page 82
- Process of Configuring the OJMS Resource Adapter on the OC4J Container on page 84
- Process of Installing Required Files for JMS Messaging on page 89
- Process of Configuring JMS Messaging on the Siebel Server on page 92
- Enabling Authentication and Authorization for the EAI JMS Transport on page 100
- Troubleshooting for the JMS Transport on page 105
- About Logging for the JMS Transport on page 106
- About Caching for the JMS Transport on page 106

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 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 in order to use the EAI JMS Transport. It will contain entries for the desired queues and topics.

The API of 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.

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/index-jsp-142945.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 desired JMS queues (implementers of javax.jms.Queue) and topics (implementers of javax.jms.Topic) 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. A value of zero for this argument disables the timeout, causing the method to wait indefinitely for a message to arrive. 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 JMS Transport

The following is a summary of supported operations for use with the 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 section “Detailed Input and Output Specifications for the EAI JMS Transport” on page 66. All methods require the JNDI name of JMS ConnectionFactory and the JNDI name of 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 Siebel's 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 may result in the appearance of additional headers or additional obscure data in the body of the message.

■ **Secure Sockets Layer (SSL).** The Siebel JMS Transport is primarily designed to support message exchange with external messaging systems (providers) using the JMS API 1.0.2b Java EE 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 SSL. For information on enabling and using SSL with the Siebel JMS Transport, contact the vendor of your JMS system. For information on the JMS API 1.0.2b standard, see:

http://java.sun.com/products/jms/docs.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, a *Unsupported Message Type* error 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 the 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 will be sent.

If XML is sent as a TextMessage, the characters will be 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* guide.

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, the session is immediately rolled back. If the operation is successful, then the session is committed, unless the operation is part of a larger multi-step operation. In the case of multi-step 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 may 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, 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." Weblogic, for example, can be configured to limit the number of times it will attempt 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 section 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" on page 66
- "Input Arguments Used by the Dispatch Step" on page 67
- "About the Output of the JMS Transport" on page 71

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 Table 13 on page 67, Table 14 on page 69, and Table 15 on page 71.

A JMS message may also be assigned *properties*. These may 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 may 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: prepended; the value is the string obtained by converting the original value to a Java String. Conversely, 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*.

Input Arguments Used by the Dispatch Step

Table 13 shows the options for each input argument of the JMS Transport methods, except user-defined properties and arguments used by the Dispatch step. R denotes a required argument; NR denotes an optional argument; and I denotes an argument that is ignored.

Table 13.   Dispatch Step Input Arguments

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[1]	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
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[3]	I	I	I	NR[2]	I
JMSType+	NR	NR	NR	I	I	I	NR[2]	I
JMSCorrelationID+	NR	NR	NR	I	I	I	I[4]	I
Dispatch								
Connection Subsystem++	NR	NR	NR	NR	NR	NR	NR	NR

Table 13.   Dispatch Step Input Arguments

Input Argument	Send	Publish	Send Receive	Receive	Subscribe	Receive Dispatch	Receive Dispatch Send	Subscribe Dispatch
DataHandling Subsystem+++	I	I	I	I	I	NR	NR	NR
DispatchService	I	I	I	I	I	R5	R5	R5
DispatchMethod	I	I	I	I	I	R5	R5	R5
DispatchWorkflow Process	I	I	I	I	I	R5	R5	R5
DispatchRuleSet	I	I	I	I	I	R5	R5	R5
ConverterService++++	I	I	I	I	I	NR	NR	NR

Some special notes regarding particular values in :

■ R^1: When the JMSReplyTo header is used, the SendQueue value is ignored.

■ NR^2: These values are assigned to the reply message during the Send step.

■ I^3: The JMSReplyTo header of the sent message is set to the value of the ReceiveQueue argument.

■ I^4: The JMSCorrelationID 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.

■ R^5: One of 3 combinations is required for these method arguments: (DService && Dmethod), DWProcess, or DRuleSet.

■ *: The ConnectionUsername and ConnectionPassword input parameters apply to IBM WebSphere MQ only.

   NOTE: When sending messages to IBM WebSphere MQ, ConnectionUsername and ConnectionPassword are recommended for all Windows platforms.

■ **: The SendUsername, SendPassword, ReceiveUsername, ReceivePassword, TopicUsername, and TopicPassword input parameters apply to Oracle WebLogic only.

■ +: This input argument can also be used as an output argument.

■ ++: For this input argument, a subsystem may be provided instead of the connection parameters. However, it must contain the same required method arguments as used for the connection parameters.

■ +++: For this input argument, a subsystem may be provided instead of the dispatch parameters. However, it must contain the same required method arguments as used for the dispatch parameters.

■ ++++: This 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 may 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" on page 73 for more information about that named subsystem.

JMS message properties are also supported as input arguments (properties) as described in "JMS Headers and Properties" on page 66.

Table 14 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.

Table 14.   Values for Input Arguments

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 or equal to 0	0: Wait indefinitely	Noninteger defaults to 3000 Integer less than 0 defaults to 0
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

Table 14.   Values for Input Arguments

Input	Default	Allowable Values	Special Values	If Value Invalid
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
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 section, "Input Arguments Used by the Dispatch Step" on page 67 for details about typing.

■ JMS properties of the received message (if the method involves receiving a message), as described in the section "JMS Headers and Properties" on page 66.

■ Certain JMS headers of the message sent or received, as described in Table 15.

■ The special properties TimedOut (if the method involves receiving a message) and DispatchError (if the method involves dispatching), as described in Table 15. Each property is either True or False.

Table 15 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.

Table 15.   Dispatch Step Output Arguments

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	No	No	No
JMSCorrelation ID+	No	No	Yes	Yes	Yes	No	No	No
JMSRedelivered	No	No	Yes	Yes	Yes	No	No	No
JMSTimestamp	No	No	Yes	Yes	Yes	No	No	No
JMSMessageID	Yes[1]	Yes[1]	Yes[2]	Yes[2]	Yes[2]	No	Yes[1]	No
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: prepended to the original property name, and the value is converted to a String.

For the multi-step 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 JMS Transport

The 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 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, an error will occur.

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, refer to "Troubleshooting for the JMS Transport" on page 105.

The following JMS Transport configuration topics are also discussed here:

■ "About the JMSSubsys Named Subsystem" on page 73

■ "About the JMS Receiver" on page 73

■ "Creating a JMS Subsystem by Using the Siebel Web Client" on page 74

■ "Sending and Receiving Messages with the JMS Transport" on page 75

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 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, 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, there must be a properly configured instance of the type JVMSubSys named JAVA.

About the JMS Receiver

The JMS Receiver is a Siebel Server component that makes it possible for the JMS Transport to be invoked asynchronously. The JMS Receiver listens for messages arriving on a JMS Queue or Topic and takes action whenever a message arrives. The JMS Receiver repeatedly invokes a single method of the JMS Transport: ReceiveDispatch, ReceiveDispatchSend, or SubscribeDispatch.

An instance of the JMS Receiver is configured with the parameters of a JMSSubsys named subsystem, which fix the queue or topic to listen to, as well as 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 could 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" on page 78.

For a discussion of named subsystems for Siebel EAI, see Chapter 2, "EAI Transports and Interfaces Overview."

For more information on administering named subsystems, 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 top 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 bottom applet), specify the parameters required for the type of operations the subsystem will need to support (for example, Receive or ReceiveDispatchSend).

For example, if this subsystem needed to support the ReceiveDispatchSend operation, 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

Sending and Receiving Messages with the JMS Transport

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 In Siebel Tools, set up a workflow for sending a message out and receiving a message in response using the EAI JMS Transport. Define the flow as shown in the following figure.
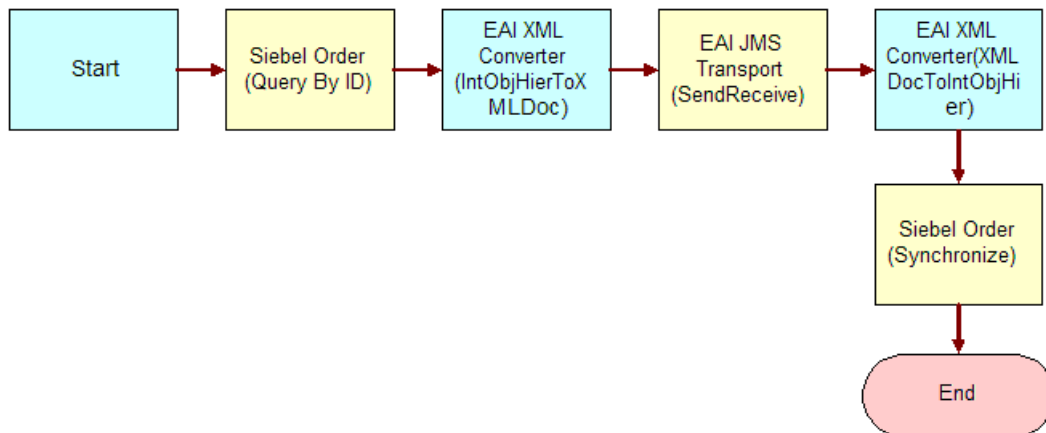
NOTE: For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide*.

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
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
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 below 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 In Siebel Tools, set up a workflow to process the incoming XML request.

The following workflow receives the incoming XML document and converts it to an integration object, executes a query using Siebel Order application service, and converts the response to an XML document as shown in the following figure:
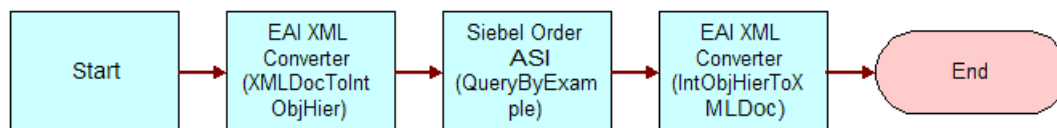
NOTE: For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide.*

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 will convert 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 will query 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 will convert 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: The Siebel Server needs to be restarted for the new subsystem to be available.

Following is an example using SrvrMgr:

NOTE: ConnectionFactory, ReceiveQueue and SendQueue require JNDI names, which will vary 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 defined as previously defined (JMS Query Order):

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

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

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 will query 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>
```

Roadmap for Configuring JMS Messaging Between Siebel Business Applications and Oracle SOA Suite Using Oracle Advanced Queuing

Service-Oriented Architecture (SOA) facilitates the development of enterprise applications as modular business services that can be integrated and reused. Oracle SOA Suite, a component of Oracle Fusion Middleware, provides a set of service infrastructure components for creating, deploying and managing SOA applications.

This process describes how to integrate Siebel Business Applications with Oracle SOA Suite using Java Message Service (JMS) messaging and Oracle Advanced Queuing (AQ), the message queuing functionality of the Oracle database.

Figure 1 shows the flow of outbound JMS messaging from Siebel Business Applications to Oracle SOA Suite. The Siebel EAI Application Object Manager places the message in the outbound queue, then Oracle SOA Suite retrieves it.
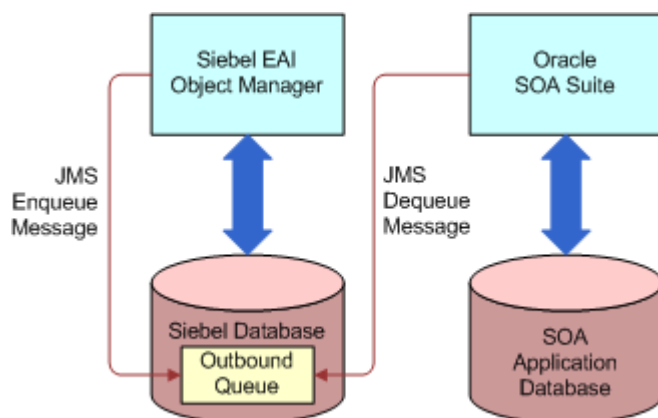

Figure 1.   Outbound JMS Message Flow

Figure 2 shows the flow of inbound JMS messaging from Oracle SOA Suite to Siebel Business Applications. Oracle SOA Suite places the message in the inbound queue, then the Siebel EAI Application Object Manager retrieves it.
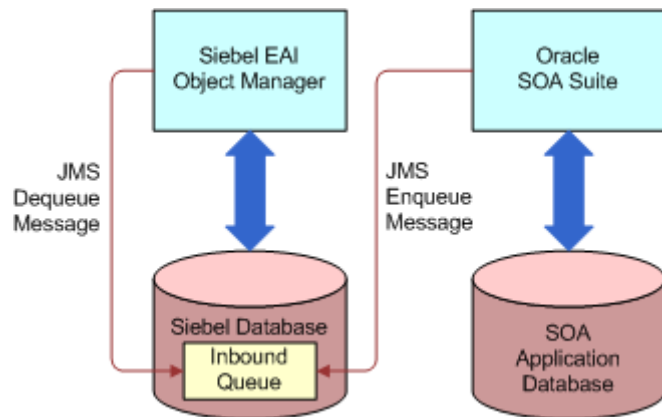

Figure 2.   Inbound JMS Message Flow

The Siebel Business Applications to SOA Suite integration requires the following tasks:

1   "Creating JMS Queues" on page 82

2   "Process of Configuring the OJMS Resource Adapter on the OC4J Container" on page 84

3   "Process of Installing Required Files for JMS Messaging" on page 89

4   "Process of Configuring JMS Messaging on the Siebel Server" on page 92

For information on connecting to Oracle Application Integration Architecture (AIA) using JMS messaging, see Article ID 971592.1 on My Oracle Support.

Creating JMS Queues

The first steps in configuring JMS messaging are creating the JMS user and queues in the Siebel Database. While not required, it is recommended to create the queues in the same database that Siebel Business Applications use, so that the queues and Siebel Server will be tightly coupled.

NOTE: All database character sets must be the same, otherwise you might encounter encoding and formatting issues with the messages.

This task is a step in "Roadmap for Configuring JMS Messaging Between Siebel Business Applications and Oracle SOA Suite Using Oracle Advanced Queuing" on page 81.

To create the JMS queues

1   Create the JMSUSER database user by running the following SQL script. The JMSUSER database user is used for queue access:

        DROP USER jmsuser CASCADE;

```
CREATE USER jmsuser IDENTIFIED BY jmsuser;

GRANT connect,resource,AQ_ADMINISTRATOR_ROLE TO jmsuser IDENTIFIED BY jmsuser;

GRANT execute ON sys.dbms_aqadm TO jmsuser;

GRANT execute ON sys.dbms_aq TO jmsuser;

GRANT execute ON sys.dbms_aqin TO jmsuser;

GRANT execute ON sys.dbms_aqjms TO jmsuser;

connect jmsuser/jmsuser;
```

NOTE: The commands must run with a user who has system privileges on the designated database (for example, SYSTEM).

2   Create the TESTJMSQTAB queue table by running the following SQL script. Queues use a queue table:

```
Begin

DBMS_AQADM.CREATE_QUEUE_TABLE(Queue_table =>'TESTJMSQTAB',Queue_payload_type
=>'SYS.AQ$_JMS_MESSAGE', sort_list => 'PRIORITY,ENQ_TIME',
multiple_consumers=>false, compatible =>'8.1.5');

End;
```

3   Create the TESTJMSQUEUE_IN inbound queue within TESTJMSQTAB by running the following SQL script:

```
Begin

DBMS_AQADM.CREATE_QUEUE(Queue_name =>'TESTJMSQUEUE_IN', Queue_table
=>'TESTJMSQTAB');

End;
```

4   Create the TESTJMSQUEUE_OUT outbound queue within TESTJMSQTAB by running the following SQL script:

```
Begin

DBMS_AQADM.CREATE_QUEUE(Queue_name =>'TESTJMSQUEUE_OUT', Queue_table
=>'TESTJMSQTAB');

End;
```

5   Start the queues by running the following SQL scripts:

```
Begin

sys.dbms_aqadm.start_queue(queue_name => 'TESTJMSQUEUE_IN', enqueue => TRUE,
dequeue => TRUE);

End;

Begin
```

```
sys.dbms_aqadm.start_queue(queue_name => 'TESTJMSQUEUE_OUT', enqueue => TRUE,
dequeue => TRUE);

End;
```

Process of Configuring the OJMS Resource Adapter on the OC4J Container

The Oracle Java Messaging Service (OJMS) resource adapter deployed on the Oracle Containers for J2EE (OC4J) container is used to send messages to AQ-JMS queues. The OJMS resource adapter acts as a bridge between Siebel Business Applications and the AQ-JMS queue.

You perform the following tasks to configure the OJMS resource adapter:

1 "Creating the Connection Pool" on page 84

2 "Creating the Data Source" on page 86

3 "Creating the Resource Adapter" on page 86

4 "Creating the Connection Factory" on page 87

5 "Creating the Administered Object" on page 87

6 "Verifying the Resource Adapter Configuration" on page 88

7 "Configuring EJB RMI Client Access" on page 88

Configuring the OJMS resource adapter on the OC4J container is a step in "Roadmap for Configuring JMS Messaging Between Siebel Business Applications and Oracle SOA Suite Using Oracle Advanced Queuing" on page 81.

NOTE: The instructions given here are for Oracle Enterprise Manager 10*g* Release 3 (10.1.3.3).

Creating the Connection Pool

You create the connection pool in Oracle Enterprise Manager. This is a step in "Process of Configuring the OJMS Resource Adapter on the OC4J Container" on page 84.

To create the connection pool

1 Log in to Oracle Enterprise Manager.

2 Select the OC4J instance, for example oc4j_soa.

3 On the Administration tab, select Services, then JDBC Resources.

4 On the JDBC Resources page, under Connection Pools, click Create.

5 On the Create Connection Pool - Application page, accept the defaults and click Continue.

6 On the Create Connection Pool page, enter the following information:

Parameter	Value
Name	TestJmsConn
JDBC URL	jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=on) (ADDRESS=(PROTOCOL=tcp)(HOST=*hostname*)(PORT=*port*)))(CONNECT_ DATA=(SERVICE_NAME=*servicename*)))
Username	jmsuser
Password (clear text)	jmsuser

7 Click Finish.

The new connection pool is displayed on the JDBC Resources page.

8 Open the data-sources.xml file, which is found in the *OracleAS_HOME*/j2ee/*OC4J_container*/ config directory, in a text editor.

9 Add the following parameters to the connection-pool tag for the TestJmsConn connection pool:

    abandoned-connection-timeout="-1"

    inactivity-timeout="60"

    used-connection-wait-timeout="60"

10 Remove the following code if present:

    <property name="connectionCachingEnabled" value="False"/>

The code for the TestJmsConn connection pool will resemble the following example:

    <connection-pool name="TestJmsConn" abandoned-connection-timeout="-1"
    inactivitytimeout="60" used-connection-wait-timeout="60">

        <connection-factory factory-class="oracle.jdbc.pool.OracleDataSource"
        user="jmsuser"password="jmsuser"
        url="jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=on)(ADDRESS=
        (PROTOCOL=tcp)(HOST=localhost)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=
        orcl)))" commit-record-table-name="">

        </connection-factory>

    </connection-pool>

Creating the Data Source

You create the data source in Oracle Enterprise Manager. This is a step in .

To create the data source

1   On the JDBC Resources page in Oracle Enterprise Manager, under Data Sources, click Create.

2   On the Create Data Source - Application & Type page, accept the defaults and click Continue.

3   On the Create Data Source - Managed Data Source page, enter the following information:

Parameter	Value
Name	TestJmsDS
JNDI Location	jdbc/TestJmsDS
Connection Pool	TestJmsConn

4   Click Finish.

The new data source is displayed on the JDBC Resources page.

Creating the Resource Adapter

You create the resource adapter in Oracle Enterprise Manager. This is a step in .

To create the resource adapter

1   On the Administration tab for the OC4J instance in Oracle Enterprise Manager, select Services, then Enterprise Messaging Service, and then Database Persistence.

2   On the Database Persistence page, click Deploy.

3   On the Deploy Database Persistence Provider page, enter the following information:

Parameter	Value
Resource Adapter Module Name	TestJmsRA
Resource Provider Name	TestJmsRP
Datasource JNDI location	jdbc/TestJmsDS

4   Click OK.

5   On the Confirmation page, click Restart, then confirm that you want to restart the OC4J instance.

Creating the Connection Factory

You create the connection factory for the resource adapter in Oracle Enterprise Manager. This is a step in *"Process of Configuring the OJMS Resource Adapter on the OC4J Container" on page 84*.

To create the connection factory

1   On the Administration tab for the OC4J instance in Oracle Enterprise Manager, select Services, Enterprise Messaging Service, and then Database Persistence.

2   On the Database Persistence page, select the OJMS RA resource adapter.

3   On the Resource Adapter: OJMS RA page, select the Connection Factories tab, then click Create.

4   On the Create Connection Factory: Select Interface page, select javax.jms.QueueConnectionFactory from the drop-down menu, then click Continue.

5   On the Create Connection Factory page, enter the following information, then click Finish.

Parameter	Value
JNDI Location	TestJmsRA/MyQCF

The TestJmsRA/MyQCF connection factory is displayed on the Resource Adapter: OJMS RA page.

Creating the Administered Object

You create the administered object for the resource adapter in Oracle Enterprise Manager. This is a step in *"Process of Configuring the OJMS Resource Adapter on the OC4J Container" on page 84*.

To create the administered object

1   On the Resource Adapter: OJMS RA page, select the Administered Objects tab, then click Create.

2   On the first Create Administered Object page, keep the default and click Continue.

3   On the second Create Administered Object page, enter the following information, then click Finish.

Parameter	Value
JNDI Location	TestJmsRA/Admin

The TestJmsRA/Admin administered object is displayed on the Resource Adapter: OJMS RA page.

Verifying the Resource Adapter Configuration

You verify the resource adapter configuration in the application.xml file, which is found in the *OracleAS_HOME*/j2ee/*OC4J_container*/config directory. This is a step in "Process of Configuring the OJMS Resource Adapter on the OC4J Container" on page 84.

To verify the resource adapter configuration

1 Open the application.xml file in a text editor.

2 Verify that it contains the following:

```
<resource-provider name="TestJmsRP" class="oracle.jms.OjmsContext">

    <description></description>

    <property name="datasource" value="jdbc/TestJmsDS" />

</resource-provider>
```

Configuring EJB RMI Client Access

You must enable the user role and allow it access to the Enterprise Java Bean (EJB) Remote Method Invocation (RMI) client. This is a step in "Process of Configuring the OJMS Resource Adapter on the OC4J Container" on page 84.

For information on allowing the user role access to the EJB RMI client, see the topic on permitting EJB RMI client access in *Oracle Containers for J2EE Security Guide*. For information on enabling the anonymous user for testing purposes, see Article ID 971592.1 on My Oracle Support.

Process of Installing Required Files for JMS Messaging

You must perform the following tasks before you can configure JMS messaging:

1 "Installing the Java Virtual Machine" on page 89

2 "Installing Java Archive Files" on page 89

3 "Creating the jndi.properties File" on page 91

Installing required files is a step in "Roadmap for Configuring JMS Messaging Between Siebel Business Applications and Oracle SOA Suite Using Oracle Advanced Queuing" on page 81.

NOTE: These procedures use Oracle Containers for J2EE (OC4J) 10*g* Release 3 (10.1.3.3.0).

Installing the Java Virtual Machine

A Java Virtual Machine (JVM) must be set up on the Siebel Server and on any Siebel Mobile Web Client or Developer Web Client machines. This is a step in "Process of Installing Required Files for JMS Messaging" on page 89.

Following the installation, verify the location of the JVM library file, which will be used in the configuration of the Siebel Java subsystem. For the locations of JVM library files on various operating systems, see "Prerequisites for Implementing a Java Business Service" on page 51.

For supported JVM versions, see *Siebel System Requirements and Supported Platforms* on Oracle Technology Network.

Installing Java Archive Files

Certain Java Archive (JAR) files are required to be installed on the computer that is running the Siebel Server (for example, in the D: \JMS directory) to communicate with Oracle SOA Suite. This is a step in "Process of Installing Required Files for JMS Messaging" on page 89.

Table 16 lists the required JAR files.

Table 16.   JAR Files Required for JMS Messaging

Source	JAR File
oc4j_extended_101330.zip	■ adminclient.jar ■ bcel.jar ■ dms.jar ■ ejb.jar ■ javax77.jar ■ jms.jar ■ jta.jar ■ mail.jar ■ oc4jclient.jar ■ ojdbc14dms.jar ■ optic.jar ■ pcl.jar
SIEBSRVR_ROOT\CLASSES	■ Siebel.jar ■ SiebelJI_*lang*.jar where *lang* is the language code of the default language of the Siebel installation
OracleAS_HOME/datadirect/sun/lib	■ jndi.jar
OracleAS_HOME/j2ee/home/lib	■ oc4j-internal.jar, com/evermind/util/JCAProperties.class file

To install JAR files

1   Download the OC4J Standalone 10*g* Release 3 (10.1.3.3.0) Zip archive, oc4j_extended_101330.zip, and extract its contents to your JMS directory, for example, D:\JMS.

   The oc4j_extended_101330.zip archive can be downloaded directly from the following URL:

   http://download.oracle.com/otn/java/oc4j/101330/oc4j_extended_101330.zip

   It is also part of the Oracle 10*g* R3 Companion (10.1.3.x) CD, which can be downloaded from the Oracle Application Server 10*g* Release 3 (10.1.3.x) Downloads site on Oracle Technology Network:

   http://www.oracle.com/technetwork/middleware/ias/downloads/101310-085449.html

2   Copy the following Siebel JAR files from the *SIEBSRVR_ROOT*\CLASSES directory to D:\JMS:

- Siebel.jar

- SiebelJI_*lang*.jar, where *lang* is the language code of the default language of the Siebel installation

3 Copy the jndi.jar file from the *OracleAS_HOME*/datadirect/sun/lib directory to the D:\JMS\j2ee\home\lib directory.

4 Copy the com/evermind/util/JCAProperties.class file from the oc4j-internal.jar file in the *OracleAS_HOME*/j2ee/home/lib directory to the D:\JMS directory. You can use the following command to extract the JCAProperties.class file while preserving the directory structure:

```
jar vxf oc4j-internal.jar com/evermind/util/JCAProperties.class
```

Creating the jndi.properties File

The jndi.properties file provides the JMS subsystem on the Siebel Server with a connection point (RMIInitialContextFactory) that it can use to look up queues and their connection factories dynamically. The jndi.properties file is created on the computer where the Siebel Server is installed, for example, in the D:\JMS directory. This is a step in "Process of Installing Required Files for JMS Messaging" on page 89.

NOTE: The full path of the jndi.properties file must be included in the CLASSPATH in "Creating the JVM Subsystem" on page 92.

To create the jndi.properties file

- Use a text editor to create the jndi.properties file (for example, in the D:\JMS directory) as follows:

    java.naming.factory.initial=oracle.j2ee.rmi.RMIInitialContextFactory

    java.naming.provider.url=opmn:ormi://*Oracle_Fusion_Middleware_server:port*/ *OC4J_instance*

    java.naming.security.principal=*username*

    java.naming.security.credentials=*password*

The following is an example of a jndi.properties file:

    java.naming.factory.initial=oracle.j2ee.rmi.RMIInitialContextFactory

    java.naming.provider.url=opmn:ormi://SOAserver.mycompany.com:6003/oc4j_soa

    java.naming.security.principal=oc4jadmin

    java.naming.security.credentials=welcome1

NOTE: When creating the jndi.properties file, make sure that the port is set correctly in the java.naming.provider.url parameter. The previous example shows 6003, but it might be a different value. Contact your Oracle Fusion Middleware administrator for help with determining the correct port.

Process of Configuring JMS Messaging on the Siebel Server

Perform the following tasks to configure JMS messaging in Siebel Business Applications:

1 *"Creating the JVM Subsystem" on page 92*

2 *"Creating the JMS Transport" on page 95*

3 *"Creating the JMS Receiver" on page 96*

4 *"Sending Messages" on page 97*

5 *"Receiving Messages" on page 99*

6 *"Starting the JMS Receiver Server Task" on page 100*

Configuring JMS messaging on the Siebel Server is a step in *"Roadmap for Configuring JMS Messaging Between Siebel Business Applications and Oracle SOA Suite Using Oracle Advanced Queuing" on page 81*.

Creating the JVM Subsystem

You can create the JVM subsystem using either the Siebel client or the command-line interface with the Siebel Server Manager (srvrmgr) utility. This is a step in *"Process of Configuring JMS Messaging on the Siebel Server" on page 92*.

Creating the JVM Subsystem Using the Siebel Client

The following procedure shows how to use the Siebel client to create the JVM subsystem.

To create the JVM subsystem using the Siebel client

1 In the Siebel application, navigate to the Administration - Server Configuration screen, Enterprises view.

2 In the top 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.

5 Set the following parameters:

Name	Value
Profile	JAVA
Alias	JAVA
Subsystem Type	JVMSubsys

6 In the Profile Parameters list applet (the bottom applet), set the following parameters:

Name	Alias	Data Type	Value
JVM Classpath	CLASSPATH	String	Must contain the following: ■ JMS provider JAR files ■ Siebel.jar and SiebelJI_*lang*.jar ■ Folder that contains jndi.properties **CAUTION:** Do not include oc4j-internal.jar in the CLASSPATH. Doing so will cause a Java exception, and the JMS implementation will not work. On Windows, the files in the CLASSPATH must be separated by semicolons (;) and end with a semicolon and period (;.), for example: D:\JMS;D:\JMS\Siebel.jar;D:\JMS\SiebelJI_enu.jar;D:\JMS\j2ee\home\lib\adminclient.jar;D:\JMS\j2ee\home\lib\jndi.jar;D:\JMS\j2ee\home\lib\jta.jar;D:\JMS\j2ee\home\oc4jclient.jar;D:\JMS\j2ee\home\lib\bcel.jar;D:\JMS\j2ee\home\lib\jms.jar;D:\JMS\j2ee\home\lib\javax77.jar;D:\JMS\jdbc\lib\ojdbc14dms.jar;D:\JMS\lib\dms.jar;D:\JMS\opmn\lib\optic.jar;D:\JMS\j2ee\home\lib\ejb.jar;D:\JMS\j2ee\home\lib\mail.jar;D:\JMS\j2ee\home\lib\pcl.jar;. On UNIX, the files in the CLASSPATH must be separated by colons (:) and end with a colon and period (:.), for example: /usr/jms:/usr/jms/Siebel.jar:/usr/jms/SiebelJI_enu.jar:/usr/jms/j2ee/home/lib/adminclient.jar:/usr/jms/j2ee/home/lib/jndi.jar:/usr/jms/j2ee/home/lib/jta.jar:/usr/jms/j2ee/home/oc4jclient.jar:/usr/jms/j2ee/home/lib/bcel.jar:/usr/jms/j2ee/home/lib/jms.jar:/usr/jms/j2ee/home/lib/javax77.jar:/usr/jms/jdbc/lib/ojdbc14dms.jar:/usr/jms/lib/dms.jar:/usr/jms/opmn/lib/optic.jar:/usr/jms/j2ee/home/lib/ejb.jar:/usr/jms/j2ee/home/lib/mail.jar:/usr/jms/j2ee/home/lib/pcl.jar:. **NOTE:** The length of the CLASSPATH is limited to 1024 characters.

Name	Alias	Data Type	Value
JVM DLL Name	DLL	String	The path to the JVM library file, for example: `C:\PROGRA~1\Java\JDK15~1.0_0\jre\bin\server\jvm.dll`

JVM Options	VMOPTIONS	String	JVM-specific options for the log file, for example:
			
			■ Windows:
			
			`-Djms.log=D:\JMS\log\jms.log`
			
			■ Solaris:
			
			`-Xusealtsigs -Xrs -Djms.log=/usr/jms/log/jms.log`

7 Restart the Siebel Server.

Creating the JVM Subsystem Using Siebel Server Manager

The following procedure shows how to use Siebel Server Manager (srvrmgr) to create the JVM subsystem.

CAUTION: Do not include oc4j-internal.jar in the CLASSPATH. Doing so will cause a Java exception, and the JMS implementation will not work.

To create the JVM subsystem using Siebel Server Manager

1 In the srvrmgr utility, type the following command, for example:

■ Windows:

```
create named subsystem JAVA for subsystem JVMSubSys with
DLL=C:\Sun\SDK\jdk\jre\bin\server\jvm.dll, CLASSPATH =
D:\JMS;D:\JMS\Siebel.jar;D:\JMS\SiebelJI_enu.jar;
D:\JMS\j2ee\home\lib\adminclient.jar;D:\JMS\j2ee\home\lib\jndi.jar;
D:\JMS\j2ee\home\lib\jta.jar;D:\JMS\j2ee\home\oc4jclient.jar;
D:\JMS\j2ee\home\lib\bcel.jar;D:\JMS\j2ee\home\lib\jms.jar;
D:\JMS\j2ee\home\lib\javax77.jar;D:\JMS\jdbc\lib\ojdbc14dms.jar;
D:\JMS\lib\dms.jar;D:\JMS\opmn\lib\optic.jar;D:\JMS\j2ee\home\lib\ejb.jar;
D:\JMS\j2ee\home\lib\mail.jar;D:\JMS\j2ee\home\lib\pcl.jar;.,
VMOPTIONS="-Djms.log=D:\JMS\log\jms.log"
```

■ UNIX:

```
create named subsystem JAVA for subsystem JVMSubSys with DLL= /usr/jdk1.5.0_06/
jre/lib/sparc/client/libjvm.so, CLASSPATH = /usr/jms:/usr/jms/Siebel.jar:/usr/
jms/SiebelJI_enu.jar:/usr/jms/j2ee/home/lib/adminclient.jar:/usr/jms/j2ee/home/
lib/jndi.jar:/usr/jms/j2ee/home/lib/jta.jar:/usr/jms/j2ee/home/oc4jclient.jar:/
usr/jms/j2ee/home/lib/bcel.jar:/usr/jms/j2ee/home/lib/jms.jar:/usr/jms/j2ee/
home/lib/javax77.jar:/usr/jms/jdbc/lib/ojdbc14dms.jar:/usr/jms/lib/dms.jar:/
```

```
usr/jms/opmn/lib/optic.jar:/usr/jms/j2ee/home/lib/ejb.jar:/usr/jms/j2ee/home/
lib/mail.jar:/usr/jms/j2ee/home/lib/pcl.jar:.,
VMOPTIONS="-Xusealtsigs -Xrs -Djms.log=/usr/jms/log/jms.log"
```

2   Restart the Siebel Server.

Creating the JMS Transport

You can create the JMS Transport using either the Siebel client or the command-line interface with the Siebel Server Manager (srvrmgr) utility. This is a step in "Process of Configuring JMS Messaging on the Siebel Server" on page 92.

Creating the JMS Transport Using the Siebel Client

The following procedure shows how to use the Siebel client to create the JMS Transport.

To create the JMS Transport using the Siebel client

1   In the Siebel application, navigate to the Administration - Server Configuration screen, Enterprises view.

2   In the top 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.

5   Set the following parameters:

Name	Value
Name	Customer_JMS_Receive
Alias	Customer_JMS_Receive
Subsystem Type	JMSSubsys

6   In the Profile Parameters list applet (the bottom applet), set the following parameters:

Name	Alias	Data Type	Value
ConnectionFactory name	ConnectionFactory	String	java:comp/resource/TestJmsRP/ QueueConnectionFactories/myQCF
JVM Subsystem name	JVMSubsys	String	JAVA
ReceiveQueue name	ReceiveQueue	String	java:comp/resource/TestJmsRP/ Queues/TESTJMSQUEUE_IN

Name	Alias	Data Type	Value
SendQueue name	SendQueue	String	java:comp/resource/TestJmsRP/Queues/TESTJMSQUEUE_OUT
Receive Timeout	ReceiveTimeout	String	20000 (in milliseconds)

Creating the JMS Transport Using Siebel Server Manager

The following procedure shows how to use Siebel Server Manager (srvrmgr) to create the JMS Transport.

To create the JMS Transport using Siebel Server Manager

■ In srvrmgr, type the following command:

```
create named subsystem Customer_JMS_Receive for subsystem JMSSubsys with
ConnectionFactory="java:comp/resource/TestJmsRP/QueueConnectionFactories/
myQCF",  ReceiveQueue="java:comp/resource/TestJmsRP/Queues/TESTJMSQUEUE_IN",
SendQueue="java:comp/resource/TestJmsRP/Queues/TESTJMSQUEUE_OUT",
ReceiveTimeout=20000
```

Creating the JMS Receiver

You can create the JMS Receiver using either the Siebel client or the command-line interface with the Siebel Server Manager (srvrmgr) utility. This is a step in "Process of Configuring JMS Messaging on the Siebel Server" on page 92.

Creating the JMS Receiver Using the Siebel Client

The following procedure shows how to use the Siebel client to create the JMS Receiver.

To create the JMS Receiver using the Siebel client

1 In the Siebel application, navigate to the Administration - Server Configuration screen, Enterprises view.

2 In the top 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.

5 Set the following parameters:

Name	Value
Name	Customer_JMS_Datahandler

Name	Value
Alias	Customer_JMS_Datahandler
Subsystem Type	EAITransportDataHandlingSubsys

6   In the Profile Parameters list applet (the bottom applet), set the following parameter:

Name	Alias	Data Type	Value
Workflow to Execute	DispatchWorkflowProcess	String	Customer JMS-AQ Receive Message

Creating the JMS Receiver Using Siebel Server Manager

The following procedure shows how to use Siebel Server Manager (srvrmgr) to create the JMS Receiver.

To create the JMS Receiver using Siebel Server Manager

■   In srvrmgr, type the following command:

```
create named subsystem Customer_JMS_Datahandler for subsystem
EAITransportDataHandlingSubsys with DispatchWorkflowProcess="Customer JMS-AQ
Receive Message"
```

Sending Messages

The following workflow provides a simple example for sending JMS messages from Siebel Business Applications. This is a step in "Process of Configuring JMS Messaging on the Siebel Server" on page 92. For more information on creating workflows, see *Siebel Business Process Framework: Workflow Guide*.

To create a workflow for sending messages

1   In Siebel Tools, create the following workflow:

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

2 Create the following process properties:

Name	Data Type	In/Out	Default String	Comments
OrderXML	String	In	Not applicable	Not applicable
JMSConnectionFactory	String	In	java:comp/resource/ TestJmsRP/ QueueConnectionFactories /myQCF	JNDI name of the JMS connection factory
JMSSendQueue	String	In	java:comp/resource/ TestJmsRP/ Queues/ TESTJMSQUEUE_OUT	JNDI name of the outbound queue
Order Message	Integration Object	In	Not applicable	Not applicable

3 Configure the Query step to use the Siebel Order business service 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

4 Configure the Convert to XML step to use the EAI XML Converter business service 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
SiebelMessage	Process Property	Not applicable	Order Message
GenerateProcessingInstructions	Literal	False	Not applicable

Property Name	Type	Output Argument
OrderXML	Output Argument	<Value>

5   Configure the Send step to use the EAI JMS Transport business service with the Send method using the following input and output arguments:

Input Argument	Type	Property Name
<Value>	Process Property	OrderXML
ConnectionFactory	Process Property	JMSConnectionFactory
SendQueue	Process Property	JMSSendQueue

Property Name	Type	Output Argument
OrderXML	Output Argument	<Value>

6   Save the workflow.

7   Publish and activate the workflow.

Receiving Messages

The following workflow provides a simple example for receiving messages to Siebel Business Applications. This is a step in . For more information on creating workflows, see *Siebel Business Process Framework: Workflow Guide*.

To create a workflow for receiving messages

1   In Siebel Tools, create the following workflow named Customer JMS-AQ Receive Message:

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

2   Create the following process property:

Name	Data Type	In/Out	Default String	Comments
<Value>	Binary	In	<Value>	Contains the received message

3 Configure the Write to File step to use the EAI File Transport business service with the Send method to write the received message to a file. Use the following input arguments:

Input Argument	Type	Value	Property Name
<Value>	Process Property	Not applicable	<Value>
FileName	Expression	"JMS_Incoming_" + [&Process Instance Id] + ".xml"	Not applicable

4 Save the workflow.

5 Publish and activate the workflow.

Starting the JMS Receiver Server Task

You start the JMS Receiver task on the Siebel Server using the Siebel Server Manager (srvrmgr) utility from the command-line interface. This is a step in "Process of Configuring JMS Messaging on the Siebel Server" on page 92.

To start the JMS Receiver server task

■ In srvrmgr, type the following command:

```
start task for comp JMSReceiver with
ReceiverConnectionSubsystem=Customer_JMS_Receive,
ReceiverDataHandlingSubsystem=Customer_JMS_Datahandler,
ReceiverMethodName=ReceiveDispatch
```

Any messages available on the external message queue are received and written to the file system.

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" on page 101

■ "Configuring Credentials in JNDI" on page 101

■ "Configuring Credentials in JMS" on page 102

■ "Configuring Against Oracle WebLogic" on page 102

■ "Configuring Against TIBCO" on page 103

■ "Configuring Against IBM WebSphere MQ" on page 104

■ "About Security Configuration on the JMS Server" on page 104

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" on page 104).

■ **SendUsername and SendPassword.** The credentials used to send messages to SendQueue (applicable for use with Oracle WebLogic only, see "Configuring Against Oracle WebLogic" on page 102).

■ **ReceiveUsername and ReceivePassword.** The credentials used to receive messages from ReceiveQueue (applicable for use with Oracle WebLogic only, see "Configuring Against Oracle WebLogic" on page 102).

■ **TopicUsername and TopicPassword.** The credentials used to publish/subscribe to/from Topic (applicable for use with Oracle WebLogic only, see "Configuring Against Oracle WebLogic" on page 102).

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 username and java.naming.security.credentials to password. For more details, refer to the JNDI specification. The construction of the naming context will automatically read the credentials from the jndi.properties file and use those credentials to connect to a JNDI server if authentication is required to perform JNDI lookup.

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 JMS Transport" on page 72.

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 will try to decrypt the password before using it. Passwords supplied using the name server have already been encrypted by the server manager, thus you do not need to encrypt it again with encryptstring.

The encryptstring utility is located in the BIN directory of your installation of the Siebel Web Server Extension plug-in.

Configuring Against Oracle WebLogic

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

To configure the EAI JMS Transport business service against the Oracle WebLogic JMS 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, 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 will contain 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

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 will 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/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, will 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 will use the same connection context.

NOTE: The MQ server does not perform authentication by default. By default, passwords are not validated. Setup authentication for MQ is a task for the 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 2008 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 on configuring the EAI JMS Transport business service against IBM WebSphere MQ, see Article ID 828113.1 on My Oracle Support.

About Security Configuration on the JMS Server

For information on 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 on logging, see "About Logging for the JMS Transport" on page 106.

■ **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, CheckJMSServer sends a message and then receives a message. If SendQueue is not specified and Topic is, it then 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, 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.

Table 17 contains more details on arguments used with some of the JMS Transport debugging methods. The arguments listed are used by all three methods.

Table 17.   Arguments to Use with the JMS Transport for Debugging Methods

Method	Argument	Display Name	Type	Description
CheckJNDIObjects CheckJMSServer CheckAll	ConnectionFactory	Connection Factory	Input	JNDI name for the JMSConnectionFactory
	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 will log 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 on the PID and thread ID.

For example, by using:

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

the log file generated will be:

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

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

About Caching for the JMS Transport

JMS Receiver connections are cached in Siebel Business Applications.

7 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

The use of the Internet protocols and technologies for business (such as HTTP, HTML, and XML) has given rise to a need to automatically send 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 Web Engine (SWE) 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 will return 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 Secure Sockets Layer (SSL) protocol, you might need 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 Business Applications, and make sure that they are operational:

■ **Siebel Web Engine.** 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 you need to initialize to process a given XML document that is received from an external system using the EAI HTTP Transport depends on the processing you need to do 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, 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 on 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 POST or GET method. Table 18 identifies restrictions on these HTTP methods.

Table 18.   Restrictions on GET and POST Methods with EAI HTTP Transport

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.
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-url` encoded 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 previously specified properties by means of .cfg file entries. In version 8.0 and higher you must switch to using named subsystems because *.cfg file entries are not supported. Only named subsystems will work for new functionality such as Dispatch Service and Character Set Conversions. You can create additional named subsystems as needed using Siebel Server Manager.

For a discussion of named subsystems for Siebel EAI, see Chapter 2, "EAI Transports and Interfaces Overview." For more information on 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" on page 19, EAI HTTP Transport methods take the arguments presented in Table 19 on page 110. Parameters are optional unless specified as required.

Table 19.   EAI HTTP Transport Send and SendReceive Arguments

Parameter	Display Name	Description
<Val ue>	*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, 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.

Table 19.   EAI HTTP Transport Send and SendReceive Arguments

Parameter	Display Name	Description
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 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, O=organization, L=Location, C=Country, E=email@domain.com` represents a certificate issued by Microsoft Certificate Authority running on the server ServerName123. RDN notation is case insensitive. For information on configuring client SSL 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 on configuring client SSL authentication, see *Siebel Security Guide*.

Table 19.   EAI HTTP Transport Send and SendReceive Arguments

Parameter	Display Name	Description
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, 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, 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, one or more request and response messages are expected. Required for session mode.
HTTPLogoffMethod	Log Off Method	Defaults is HTTPLoginMethod. HTTP method to be used for logging off. Required for session mode.

Table 19.   EAI HTTP Transport Send and SendReceive Arguments

Parameter	Display Name	Description
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 set, the logoff operation will be 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 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 Step 1 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 In Siebel Tools, set up a workflow to send this information to an external system as shown in the following figure:
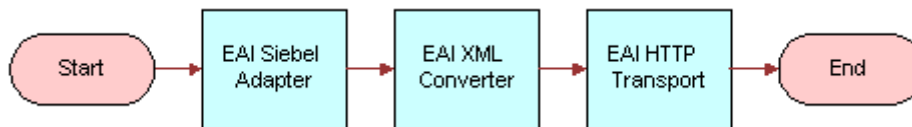
   NOTE: For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide*.

   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
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
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 will be 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 Web Engine (SWE) 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" on page 116

■   "Specifying HTTP Parameters for Inbound Integration" on page 117

■   "Using the EAI HTTP Transport in Session Mode" on page 118

■   "Using the EAI HTTP Transport in Sessionless Mode" on page 121

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 are not required when using the EAI HTTP Transport for outbound integration:

1   Install and configure the Siebel Web Server, Siebel Gateway Name Server, and Siebel Server.

2   Start the Siebel Web Server, Siebel Gateway Name Server, and Siebel Server.

3   Start the Siebel Web Engine (SWE) to be able to use the EAI HTTP Transport.

4   Configure SWE to run the EAI HTTP Transport for inbound integration. See "Specifying HTTP Parameters for Inbound Integration" on page 117.

5   Set certain configuration parameters for whatever Siebel Server you are using.

    The server component you are running must be a Client Application Manager component. Set this in the configuration file for the server component of your choice, or use named subsystems.

NOTE: You can type `http://Web_Server_Name/ObjectManager_lang/start.swe` in a Web browser on any computer that has connectivity to the Web server to check the connectivity between the URL-issuing (for the EAI HTTP Transport) machine and SWE. This URL brings up the login page of the Siebel application corresponding to the *ObjectManager_lang* parameter, confirming the connectivity between SWE and the URL-issuing machines.

Specifying HTTP Parameters for Inbound Integration

The EAI HTTP Transport is built into SWE. To use it, you need to set certain configuration parameters for the virtual directory on the Web server. Your Siebel application installation includes a configuration file called eapps.cfg in the \bin subdirectory of your installation directory. This file is on the Web server side of your configuration, as opposed to on the Siebel Server side of your installation. 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" on page 18.

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

1 Open your *eapps.cfg* file in a text editor.

2 Look for the section [*/eai_lang*]. Where *lang* is the three-letter language code for the language you are using, such as enu for U.S. English.

3 Add the EnableExtServiceOnly configuration parameter or set it as follows, if it already exists, to enable the HTTP inbound transport. This example shown is for use with UNIX in an English environment.

```
[/eai_enu]
ConnectString = Connect String
EnableExtServiceOnly = TRUE
```

For the virtual directory, you need to set the ConnectString parameter. The syntax for the ConnectString is:

```
ConnectString = siebel[.transport][.encryption][.compression]://SiebelServer
:SCBPort/EnterpriseName/XXXObjMgr_lang
```

where:

transport is TCPIP or http

encryption is none or mscrypto

compression is none or zlib

SiebelServer is the name of your Siebel Server

SCBPort is the listening port number for the SCBroker component (usually 2321)

EnterpriseName is the name of your Siebel Enterprise Server

XXXObjMgr_lang is the type of Application Object Manager for the Siebel Business Application you are installing and the language code used for this installation

The following example shows the connect string using TCP/IP, with no encryption, no compression, and the server name and default port. In addition, you need to point to the Application Object Manager specific to the Siebel Business Application you are installing.

In the following example connect string, the Siebel Business Application installed is Siebel Sales, and the Application Object Manager is EAIObjMgr:

```
ConnectString = siebel.TCPIP.None.None://server1:2321/siebelEnt/EAIObjMgr_enu
```

4 Save and close the configuration file.

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, 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 Step 1 in either the HTTP cookie header or the URL string as a parameter.

NOTE: By default the Siebel Web Server Extension (SWSE) accepts the presence of the session cookie in either the HTTP header or the URL as a parameter. However, this behavior can be changed by setting the SessionTracking parameter in the SWSE configuration (eapps.cfg) file, in the [defaults] section or specific Siebel URL section. For more information on the SessionTracking parameter, see the session cookie topic in *Siebel Security Guide*.

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 will not be set as it is expected that this cookie will be 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 sections explain in detail how each of the session mode calls is configured.

Table 20 presents each of the Login HTTP Request variables for session mode.

Table 20.   Session Mode Variables

Variable	Description
webserver	URL of the Web server that has Siebel Web Engine installed, such as www.*myserver*.com.
path	Default is eai. Virtual path on the server referring to the specific SWE configuration.

Table 20.   Session Mode Variables

Variable	Description
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.
password	Password for the login user name above.

Login HTTP Request Example 1

In this example, if the call completes successfully, it will return a session cookie:

■ Using HTTP GET:

```
URL  = http://webserver/path/
start.swe?SWEExtSource=source&SWEExtCmd=ExecuteLogin&UserName=username&Password
=password
```

■ Using HTTP POST:

```
URL  = http://webserver/path/start.swe

HTTP Body  =
SWEExtSource=source&SWEExtCmd=ExecuteLogin&UserName=username&Password=password
```

■ Example Login URL:

```
http://www.myserver.com/eai/
start.swe?SWEExtSource=SiebelQuery&SWEExtCmd=ExecuteLogin&UserName=user1&Passwo
rd=login123
```

Login HTTP Request Example 2

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/start.swe?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*/start.swe

    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.*myserver*.com/eai/start.swe?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*/start.swe?SWEExtCmd=Logoff

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

■   Example Logoff URL:

http://www.*myserver*.com/eai/start.swe?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.

Table 21 presents each of the variables for sessionless mode.

Table 21.   Sessionless Mode Variables

Variable	Description
webserver	URL of the Web server that has Siebel Web Engine installed, such as www.*myserver*.com.
path	Default is eai. Virtual path on the server referring to the specific SWE configuration.
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.
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 on 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, the URL describes the parameters for the HTTP Inbound Transport call over HTTP:

■ Using HTTP GET:

URL = http://*webserver*/*path*/
start.swe?SWEExtSource=*source*&SWEExtCmd=Execute&UserName=*username*&Password=
password&SWEExtData=*data text*

NOTE: Unlike session mode, the SWEExtCmd is Execute, not ExecuteLogin.

■ Using HTTP POST:

URL = http://*webserver*/*path*/start.swe

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

NOTE: When using the 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 Sessionless Mode URL

NOTE: This sample URL must be entered as a single line of text. The URL is presented here on separate lines for clarity.

http://www.*myserver*.com/eai/start.swe?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>
```

To use this URL you do the following:

■ Change the Web server address, www.*myserver*.com, to your actual Web server URL.

■ Verify that the SWEExtSource argument has a corresponding section in the [HTTP Services] section of your eai.cfg file.

■ Change the Username and Password arguments to those of a valid system user, such as SADMIN/ SADMIN.

EAI HTTP Transport for Inbound Messages

To use the EAI HTTP Transport, you complete two steps:

■ Set up the business service for use in the workflow.

■ Create the workflow.

Both steps are explained in this section.

This scenario assumes incoming XML. Your business requirements dictate if and how you need to adapt these steps to fit your needs.

To set up the business service

1 Start Siebel Tools, connecting to the server.

2 Find the business service named Workflow Process Manager.

3 Copy this record and rename the copy EAITEST.

4 In the Business Service User Props list, add a new record:

a Enter ProcessName in the Name column.

b   Enter EAITEST in the Value column, as shown in the following illustration.

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		

5   Compile a new .srf file and copy it to the *SIEBEL_ROOT*\Objects*lang_code* directory.

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*.

6   Restart the Siebel Server.

7   Verify that the EAI Object Manager has started.

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

To create the new workflow to receive messages

1   In Siebel Tools, set up a new workflow as shown below and give it a unique name, such as EAITEST.
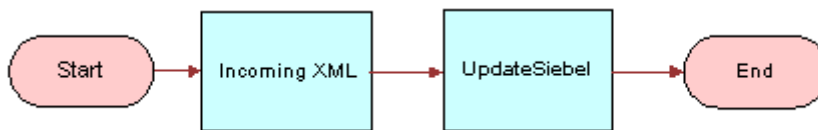
NOTE: For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide*.

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 will be placed in this variable. This allows you to then perform a given action on that data. If the POST method was used, the data sent in the Body will be stored in this property. If the GET method was used, the data sent in the URL will be 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), the values will 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.

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 shown below.

    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.

Processing and Sending Outbound XML Documents

This section 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.

Controlling the Behavior of EAI HTTP Transports

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

■ Specifying Parameters as Business Service User Properties on page 127.

■ Specifying Parameters as Subsystem Parameters on page 127.

■ About Parameters as Run-Time Properties on page 128.

■ About Parameters in Parameter Templates on page 128.

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 compiled the .srf file 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 .srf file or 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, the subsystem parameter or run-time property overrides any values you have defined in Siebel Tools and compiled into the .srf file or deployed to the run-time database.

For more information on 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 top 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
Subsystem Type	HTTPSubSys

5 In the Profile Parameters list applet (the bottom applet), specify the parameters required for the type of operations the subsystem will need to support:

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

Then, in the workflow on the Siebel Web Client, you will specify the Connection Subsystem input argument to the HTTP Transport, and the value will be the named subsystem that you created. For the case above, it will be 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.srvr.com/login?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 above 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 will 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://smthpa12.siebel.com:16007/eai_enu/
start.swe?SWEExtCmd=ExecuteLogin&SWEExtSource=MyEcho&UserName=SADMIN
&Password=db2"

HTTPLogoffMethod=GET

HTTPLogoffURLTemplate="http://smthpa12.siebel.com:16007/eai_enu/
start.swe?SWEExtCmd=Logoff"

HTTPRequestMethod=POST

HTTPRequestURLTemplate="http://smthpa12.siebel.com:16007/eai_enu/start.swe"
```

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 In Siebel Tools, set up a new workflow as follows:
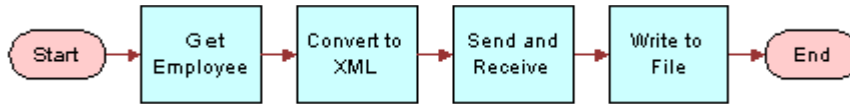
NOTE: For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide*.

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

■ "Controlling Login Sessions with Session Mode" on page 131

■ "Sending Requests in Sessionless Mode" on page 132

■ "Accessing a URL Protected by Basic Authentication" on page 133

■ "Providing Client Certificate Information for SSL Mutual Authentication" on page 133

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 Login, 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 logs in to a server with passed parameters for username and password:

> *HTTPLoginURLTemplate* = "http://$ServerPath$/
> start.swe?SWEExtSource=$Source$&SWEExtCmd=ExecuteLogin&User
> Name=$Username$&Password=$Password$"

◼ 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"

> *ServerPath* = "siebel1/eai"

> *Username* = "pdavis"

> *Password* = "1234abcd"

> *Source* = "testdoc"

In the preceding example, the ServerPath variable value of siebel1/eai is substituted for the token $ServerPath$. The Source variable value of testdoc is substituted for the $Source$ token, the Username variable value of pdavis for the token $Username$, and the Password variable value of 1234abcd for the $Password$ token.

Any XML document represented by the entry for SWEExtData can be put into the body. This would change the sample code so that the 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.*mypartner*.com/server/login.asp"

> *HTTPRequestBodyTemplate* = "Acct=ABCIntl&User=$Username$&pwd=$Password$"

> *Username* = "acctuser"

> *Password* = "123456789abcdefg"

Accessing a URL Protected by Basic Authentication

Siebel Business Applications support server, or basic, authentication. You can use basic authentication with the EAI HTTP Transport to send messages. 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@169.254.85.21:5555/mycompany.com/stuff

NOTE: For more information on authentication, see *Siebel Security Guide*.

Providing Client Certificate Information for SSL Mutual Authentication

In certain releases of versions 7.7, 7.8, 8.0, and 8.1, Siebel Business Applications support client authentication for SSL-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 on the specific releases where mutual authentication is supported, see Article ID 560965.1 on My Oracle Support.

If client authentication is enabled, 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, 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=somecertmachine,
OU=ca,O=oracle,L=boston,C=usa");
```

```
// Invoke EAI HTTP Transport

oService.InvokeMethod("SendReceive", oInputs, oOutputs);
```

For more information on configuring SSL 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 "." notation used in creating property sets.

About Sending and Receiving Messages Through HTTP

To send and receive messages through HTTP, you need to 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 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 SWE. 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 Object Manager or on the server.

About Transport Headers and HTTP Response Headers

This section 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 will be able to 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.

Transport headers have the following features:

■ Every connection has its own transport header.

■ The transport header will separately store 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 will overwrite 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 above.

■ 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 Business Applications with Java Applications

This chapter discusses the integration of Java applications with Siebel Business Applications. It includes the following topics:

About Siebel Business Applications 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 Business Applications support 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, 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://mymachine: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, 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 may 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 section. The Siebel Resource Adapter is part of the Java EE Connector Architecture, which is described in "About the Siebel Resource Adapter" on page 153.

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://mymachine: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

This section 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.
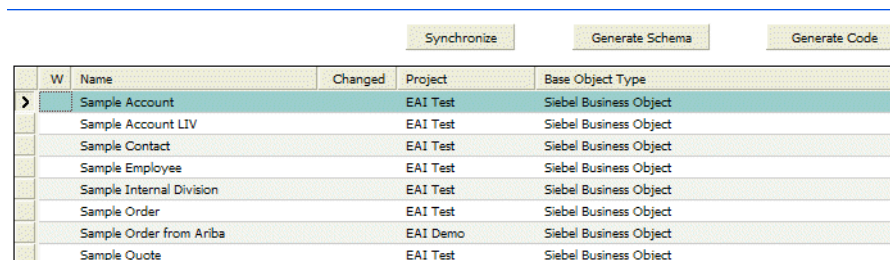
NOTE: For information on how to use Siebel Tools, consult *Using Siebel Tools*.

2 Select Business Service (or Integration Object) from the Types tab on the Object Explorer.

NOTE: If Integration Object is not present, add it by checking Integration Object on the Object Explorer tab of the Development Tools Options window opened by selecting View >Options.

3 Select the desired business service (or integration object), as shown in the following illustration for an integration object.

On the right top corner of the Integration Object list, there is a set of three buttons. The following illustration shows the Sample Account integration object highlighted.

	W	Name	Changed	Project	Base Object Type
>		Sample Account		EAI Test	Siebel Business Object
		Sample Account LIV		EAI Test	Siebel Business Object
		Sample Contact		EAI Test	Siebel Business Object
		Sample Employee		EAI Test	Siebel Business Object
		Sample Internal Division		EAI Test	Siebel Business Object
		Sample Order		EAI Test	Siebel Business Object
		Sample Order from Ariba		EAI Demo	Siebel Business Object
		Sample Quote		EAI Test	Siebel Business Object

(Buttons shown: Synchronize, Generate Schema, 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 section.

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 Figure 3, a subfolder named *com* is created in the folder specified during the generation
process. Below the com folder is a folder named *siebel* containing a folder named *service*, containing
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.

A folder is created under jdb (or jca)
for every business service generated.
The folder holds several Java files.
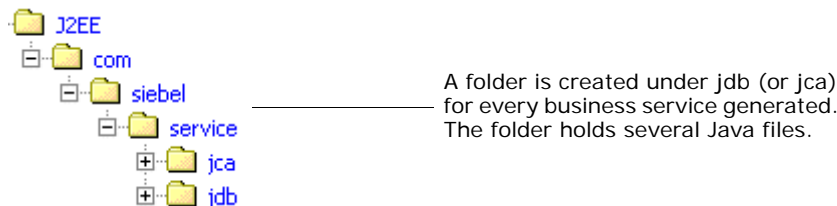
Figure 3.   Directory Structure Created to Contain Java Code for Business Services

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* prepended. 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*
prepended. 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

Table 22, Table 23, and Table 24 describe 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.

Table 22 lists methods of the Java class com.siebel.service.jdb.GenericServiceBusServAdapter
generated for an example business service, GenericService, having the business service method
GenericMethod.

Table 22.   Java Class com.siebel.service.jdb.GenericServiceBusServAdapter Methods Generated

Method	Description
GenericServiceBusServAdapter()	Constructor that uses the default properties file, siebel.properties.
GenericServiceBusServAdapter(SiebelDataBean)	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.

Table 22.   Java Class com.siebel.service.jdb.GenericServiceBusServAdapter Methods Generated

Method	Description
GenericServiceBusServAdapter(String, String, String, String)	Constructor taking the username, password, connect string, and language.
GenericMethod(GenericMethodInput)	Invokes the specified business service method.

Table 23 lists methods of the Java class com.siebel.service.jdb.GenericMethodInput generated for an example business service method, GenericMethod.

Table 23.   Java Class com.siebel.service.jdb.GenericMethodInput Methods Generated

Method	Description
GenericMethodInput()	Constructor.
GenericMethodInput(SiebelPropertySet)	Constructor that sets its fields from the given property set.
fromPropertySet(SiebelPropertySet)	Copies fields 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(String)	Sets the value of a business service method argument.

Table 24 lists methods of the Java class com.siebel.service.jdb.GenericMethodOutput generated for an example business service method, GenericMethod.

Table 24.   Java class com.siebel.service.jdb.GenericMethodOutput Methods Generated

Method	Description
GenericMethodOutput()	Constructor.
GenericMethodOutput(SiebelPropertySet)	Constructor that sets its fields from the given property set.
fromPropertySet(SiebelPropertySet)	Copies fields values from the given property set.
toPropertySet()	Returns a SiebelPropertySet with the properties and values corresponding to the fields of this object.

Table 24.   Java class com.siebel.service.jdb.GenericMethodOutput Methods Generated

Method	Description
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 may 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 Figure 4.
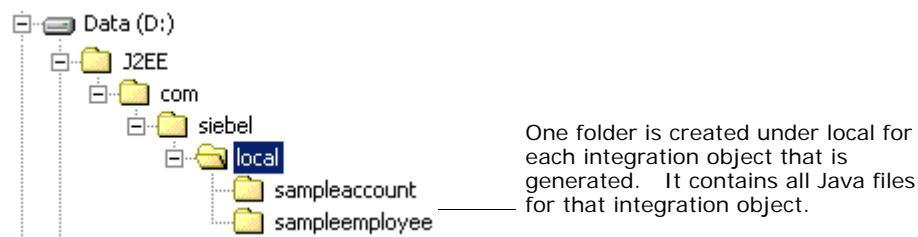
Figure 4.   Directory Structure Created of Java Code Generated for Integration Objects

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

Table 25 describes methods that are present in the generated Java code for every integration object, using the example integration object GenericIntObj.

Table 25.   Java Class com.siebel.local.GenericIntObjIO Methods Generated

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.
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.

Table 25.   Java Class com.siebel.local.GenericIntObjIO Methods Generated

Object	Description
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

Table 26 describes methods that are present in the generated Java code for every integration object component, using an example integration object component, GenericIntComp, having the child component GenericIntCompChild and field GenericField.

Table 26.   Java Class com.siebel.local.GenericIntCompIC Methods Generated

Object Component	Description
addfGenericIntCompChildIC(GenericIntCompChildIC)	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.
GenericIntCompIC()	Default constructor.
GenericIntCompIC(SiebelPropertySet)	Creates an integration object component from a property set.

Table 26.   Java Class com.siebel.local.GenericIntCompIC Methods Generated

Object Component	Description
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 SiebelDataBean, 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:

si ebel : // *Si ebel ServerName*: *SCBPort*/ *Enterpri seName*/ *XXX*Obj Mgr_*l ang*

For example:

si ebel : //mymachi ne: 2321/mysi ebel enterpri se/SCCObj Mgr_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). 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 may 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 may 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).

Table 27 gives the details of these parameters.

Table 27.   Properties in the siebel.properties File

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 will be 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 will wait 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 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.
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 will try servers from the list in a round-robin fashion. An example of such a list follows: $Virtual Server1$=sid1:host:port,sid2:host:port...; $Virtual Server2$=... where: $Virtual\ Servers$ = an assigned list of real Siebel Servers with host names and port numbers (of the local SCB).
siebel.connection.string	The Siebel connect string. For information on the syntax of the connect string, see *Siebel Object Interfaces Reference*.

Table 27.   Properties in the siebel.properties File

Property	Description
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 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://172.20.94.55:2321/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" on page 139, 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://mymachine: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://mymachine: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 on JCA, see:

http://java.sun.com/j2ee/connector

The following Siebel Resource Adapter topics are also discussed:

- "Using the Resource Adapter" on page 153
- "About the Connect String and Credentials for the Java Connector" on page 154
- "About JCA Logging" on page 156

Using the Resource Adapter

When deploying the Siebel Resource Adapter to a Java EE application server (for example, Oracle Application Server, Oracle WebLogic, or IBM WebSphere MQ), you must make sure that the necessary Siebel JAR files are included. The Siebel JAR files that need to 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, may also be required for deployment. Refer to the documentation of the Java EE application server for more information on deploying a JCA adapter on the server.

The following sections 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 username and password are present in the descriptor, they will be 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" on page 147, 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 on generating code, see "About the Siebel Code Generator" on page 139.)

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 be specified explicitly when creating the business service instance.

```
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 on generating
code, see "About the Siebel Code Generator" on page 139.)

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 be specified explicitly when
creating the business service instance.

```
    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://mymachine: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 Business Applications
version 8.0 and higher:

■ 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.mycompany.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.mycompany.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.mycompany.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

This chapter discusses the EAI DLL Transport and EAI File Transport business services. It includes the following topics:

- About the EAI DLL Transport on page 161
- About the EAI File Transport on page 164

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" on page 161
- "EAI DLL Transport Parameters" on page 162
- "Creating a DLL to Call a Function in an External DLL" on page 162

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 Chapter 2, "EAI Transports and Interfaces Overview," the EAI DLL Transport takes the parameters presented in Table 28.

Table 28.   EAI DLL Transport Parameters

Argument	Description
DLLName	Name of the (request/response) DLL.
ExternalFunction	Function in the DLL to invoke.
Return Value	The return value from the function called. This value is an output property.

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, as presented in Table 28 on page 162.

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.

Starting in release 7.5.3, it is not necessary for the DLL to release the memory either on Microsoft Windows or UNIX. The DLL transport business service will release the memory. If the DLL does a memory deallocation, it will most likely crash. The basic assumption is that the DLL must do the memory allocation with a C-style malloc only. Any other type of allocation will not be handled properly and may even lead to crashes.

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 the name of the project.

3 In the next dialog box, select the option Simple dll project.

Following files are created by default:

- Project.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:

```
StdAfx.h
struct XMLDataBuf
{
    int        nLength;
    void*      pData;
};
extern "C" int __declspec(dllexport) TestEAI(const XMLDataBuf* Value, XMLDataBuf*
pReply);
Main.cpp
#include "stdafx.h"
#include <string.h>
#include <stdio.h>
#include <io.h>


BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                                         )
{
    return TRUE;
}
extern "C" int __declspec(dllexport) TestEAI(const XMLDataBuf* Value, XMLDataBuf*
pReply)
{
    FILE *p;
    p = fopen("c:\\test.txt","w");
    fprintf(p,"before test");
    fprintf(p,"%s After Test",Value->pData);
//strcpy(s,"Hello World");
    fclose(p);
    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, it uses a string output property instead.

The following topics are discussed here:

- "EAI File Transport Methods" on page 164
- "Using the EAI File Transport Methods" on page 164
- "Generating Unique Filenames" on page 165
- "EAI File Transport Parameters" on page 165
- "EAI File Transport Named Subsystem" on page 166

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 Table 29 on page 165.

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, 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" on page 19, the EAI File Transport takes the parameters presented in Table 29. These parameters can be specified as service method arguments, subsystem parameters, or user properties.

Table 29.   EAI File Transport Parameters

Display Name	Argument	Description
Append To File	AppendToFile	Default is False. A value of True means that if the file exists, 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, no error is given, but the information is traced.

Table 29.   EAI File Transport Parameters

Display Name	Argument	Description
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, a random name is used for the output file. You must specify an explicit path for file name. You can also use $$ as the wildcard symbol in the file name. For example, if you specify a file name of "file$$.xml," then Siebel 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 set to True. If set to False (the default), 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.

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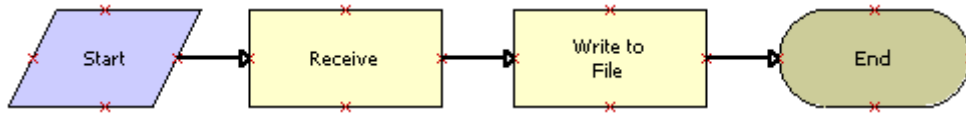
NOTE: For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide*.

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

This chapter discusses the Transcode Service business service. It includes the following topics:

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 ("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 information about supported character sets and languages in Siebel Business Applications, see *Siebel Global Deployment Guide* and *Siebel System Requirements and Supported Platforms* on Oracle Technology Network.

Transcode Service Business Service Methods

The Transcode Service business service has two methods:

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 Business Applications 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 Table 30.

Table 30.   Convert Method Arguments

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, 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 .

Table 31. Validate Method Arguments

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 will be 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 will be 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, 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, 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" on page 172

■ "Using the Convert Method" on page 175

Using the Validate Method

The following examples demonstrate the use of the Validate method of the Transcode Service business service:

■ "XML Hierarchy Example" on page 172

■ "Siebel Message Example" on page 173

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

1 Create a workflow as follows:
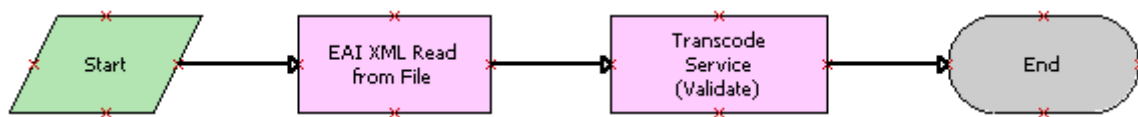
NOTE: For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide*.

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

1 Create a workflow as follows:
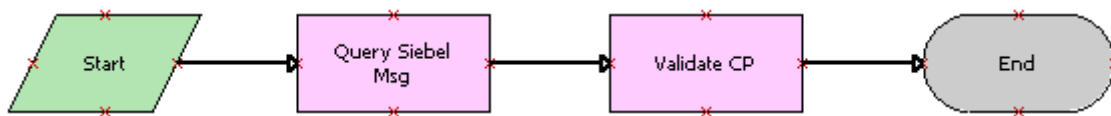
NOTE: For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide*.

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

Name	Data Type	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
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

1 Create a workflow as follows:
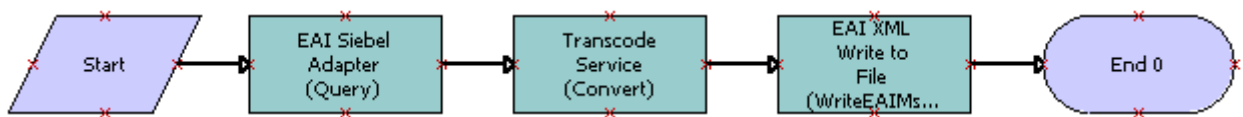
> **NOTE:** For details on the Business Process Designer, see *Siebel Business Process Framework: Workflow Guide*.

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
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

Index