webMethods Adapter for Enterprise JavaBeans Installation and User's Guide

Version 6.5 SP3

January 2013

WEBMETHODS

Document ID: ADAPTER-POJ-IUG-65 SP3-20221202

Table of Contents

About this Guide

This guide describes how to configure and use webMethods Adapter for Enterprise JavaBeans. It contains information for administrators who configure and manage webMethods Integration Server, and application developers who want to create services that exchange data with Enterprise JavaBeans (EJBs).

To use this guide effectively, you should be familiar with:

- Creating flow, Java, and/or C/C++ services
- The application server with which you will be using Adapter for Enterprise Javabeans
- Terminology and basic operations of your operating system
- The basic concepts and tasks for working with EJBs
- The setup and operation of webMethods Integration Server.
- How to perform basic tasks with Software AG Designer.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format *folder.subfolder.service*, APIs, Java classes, methods, properties.
Italic	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at https://documentation.softwareag.com.

In addition, you can also access the cloud product documentation via https://www.softwareag.cloud. Navigate to the desired product and then, depending on your solution, go to “Developer Center”, “User Center” or “Documentation”.

Product Training

You can find helpful product training material on our Learning Portal at https://knowledge.softwareag.com.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at https://techcommunity.softwareag.com. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at https://github.com/softwareag and https://hub.docker.com/u/softwareag and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at https://empower.softwareag.com. Many services on this portal require that you have an account. If you do not yet have one, you can request it at https://empower.softwareag.com/register. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

1 Overview of webMethods Adapter for Enterprise JavaBeans

About the Adapter

webMethods Adapter for Enterprise JavaBeans (Adapter for Enterprise Javabeans) is an add-on to the webMethods product suite that enables you to exchange data between webMethods Integration Server and Enterprise JavaBeans (EJBs) on an application server. The adapter provides seamless and real-time communication with the application server.

Adapter for Enterprise Javabeans provides support for the EJB 3.0 and the EJB 2.1 standards that enable the end user to efficiently configure and manage the adapter connections and services.

Adapter for Enterprise Javabeans provides user interfaces that enable you to configure and manage adapter connections and adapter services. The adapter services are linked to specific EJBs, and can be used on their own or included in flows built to implement business processes.

Using functionality provided by the adapter, a client can identify the EJBs deployed on the application server, create a remote instance of an EJB, and then invoke a selected method on the remote bean instance. For example, you can use Adapter for Enterprise Javabeans to create services that Integration Server clients use to create and update account information for a database connected to an application server that provides EJB capabilities.

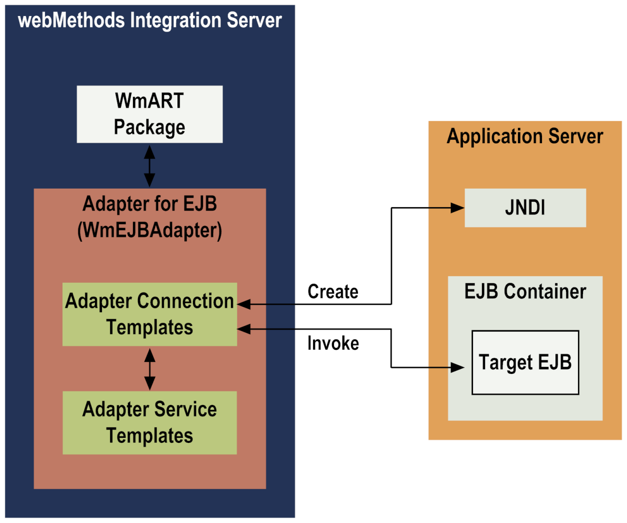
You can download a sample EJB application that illustrates how you might implement code that invokes services on Integration Server from the Software AG TECHcommunity website. The sample code is not part of Adapter for Enterprise Javabeans. For more information, see "Invoking webMethods Services From an EJB" on page 99.

Architecture and Components

Adapter for Enterprise Javabeans provides a set of user interfaces, built-in services, and templates that enable you to create integrations with EJBs on an application server. The adapter is provided as a single package that must be installed on Integration Server. For detailed installation instructions and software requirements, see "Installing, Upgrading, and Uninstalling" on page 45.

Adapter for Enterprise Javabeans exposes Enterprise JavaBeans to Integration Server by providing adapter connection templates and adapter service templates to obtain a reference to one or more bean instances, and to invoke a method on each instance (through the remote interface). Each of the service template classes can only operate against a single EJB.

The following diagram shows at a high level how the adapter components connect to an application server:


webMethods Integration Server
WmART Package
Adapter for EJB (WmEJBAdapter)
Adapter Connection Templates
Adapter Service Templates
Create
Invoke
Application Server
JNDI
EJB Container
Target EJB

- **Integration Server**. Adapter for Enterprise Javabeans is installed and runs on Integration Server.
- **WmART Package.** The WmART package provides a common framework for webMethods 6 and later adapters to use Integration Server's functionality, making Integration Server the run-time environment for Adapter for Enterprise Javabeans. The WmART package is installed with Integration Server and provides logging, transaction management, and error handling for the adapter and its connections and services.
- **Adapter for Enterprise Javabeans**. Adapter for Enterprise Javabeans is delivered as a single package called WmEJBAdapter. The adapter provides Integration Server Administrator user interfaces that enable you to configure and manage adapter connections, and Software AG Designer user interfaces that enable you to configure and manage adapter services. Adapter for Enterprise Javabeans installation includes templates from which all Adapter for Enterprise Javabeans connections and services can be created.
- **Adapter Connection Templates.** An adapter connection enables Integration Server to connect to application servers at run time. You must configure an adapter connection before you can create adapter services. Adapter for Enterprise Javabeans provides templates for adapter connections in Integration Server Administrator. For a detailed description of adapter connections and usage information, see “Adapter Connections” on page 17.
- **Adapter Service Templates.**An adapter service enables Integration Server to create and invoke a method on an EJB instance deployed on an application server. For example, an adapter service could enable Integration Server clients to write salary information to an employee database connected to an application server. Adapter for Enterprise Javabeans provides adapter service templates in Designer. For more information about adapter service templates and how the services interact, see “Adapter Services” on page 25.
- **Application Server.** Adapter for Enterprise Javabeans connections and services interact with EJBs on an application server. The application server provides support for persistence, transactions, and remote access, among other things. Within the application server, the EJB container is the component responsible for managing the EJBs deployed on that system. Individual beans provide application-specific functionality. Through support of the EJB

standards, the application server allows external applications to create and remove bean instances, and invokes methods defined by those beans.

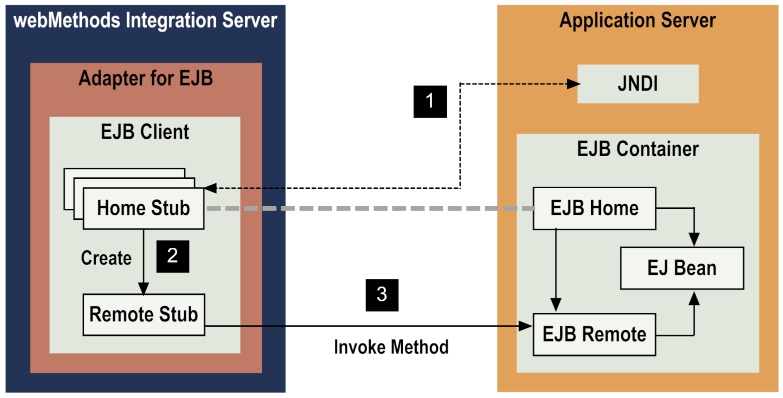
The application server also implements access to deployed beans through the home interface (only for 2.1 EJBs) and remote bean interfaces, which the bean developer defines. The home interface for 2.1 EJBs defines methods to gain access to individual instances for an EJB class, either looking up one or more instances or creating a new instance. The remote interface defines methods that may be invoked on one of those instances.

The application server also implements access to deployed beans through the remote interface for 3.0 EJBs which the bean developer defines. The remote interface for 3.0 EJBs defines methods to gain access to individual instances for an EJB class, either looking up one or more instances or creating a new instance. The remote interface defines methods that may be invoked on one of those instances.

- **JNDI.** Adapter for Enterprise Javabeans uses the Java Naming and Directory Interface (JNDI) to perform lookup operations that identify EJBs whose methods you want to invoke. Lookup operations get this information from the application server's JNDI implementation. After identifying the EJB whose public methods you want to invoke, the adapter gets public interface details about the EJB. Public interface details include whether the EJB is a session or entity bean, details about how to create an instance of the EJB, and what business methods it exposes. The adapter then enables you to configure service instances that execute the public EJB methods on the application server.

Services created with Adapter for Enterprise Javabeans are actually remote EJB client applications that interact with the deployed EJB through stub interfaces. These are lightweight, external, and distributable representations of the actual EJB. A remote client never interacts directly with the EJB itself.

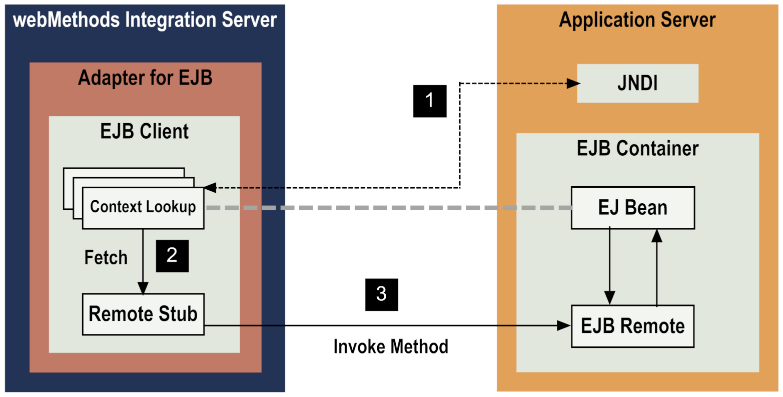
The basic data flow for any client-initiated 2.1 EJB interaction is as follows:



Step	Action
1	The client accesses the JNDI server to look up the EJB home interface of an EJB deployed on the application server.

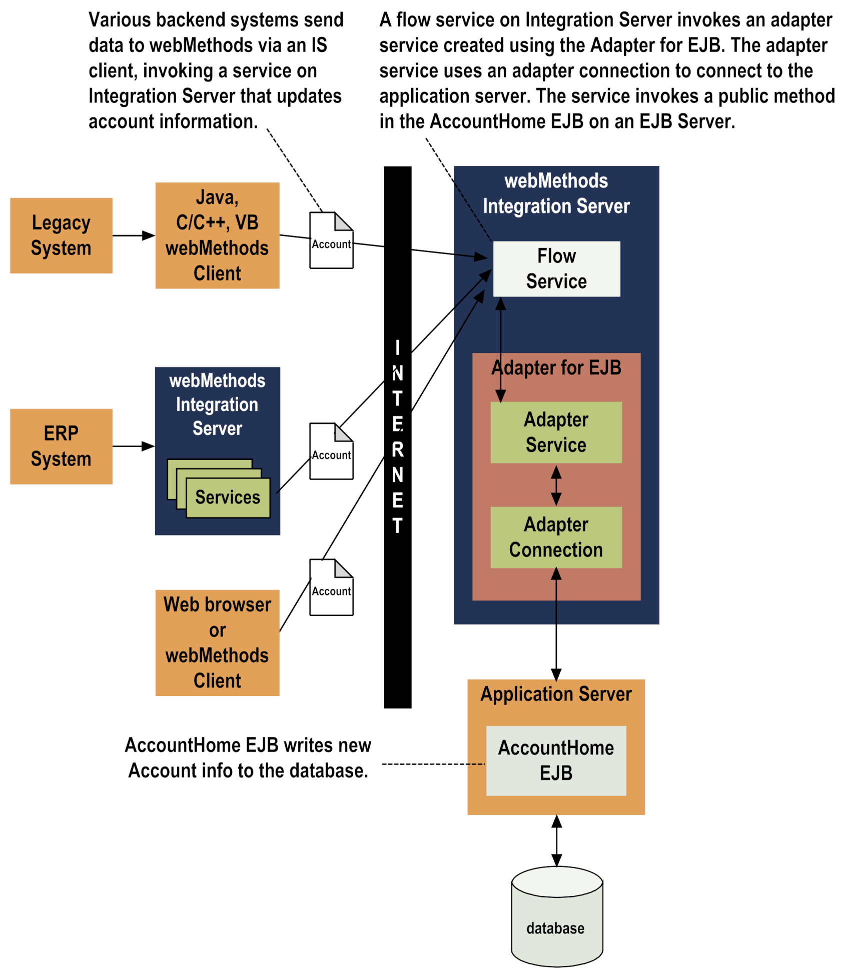
Step	Action
2	The client invokes a "creator/finder" method on the home interface, creating an EJB instance with which Adapter for Enterprise Javabeans can interact. The home interface returns a remote EJB interface to the client.
3	The client then invokes the desired public method on the remote EJB.

The basic data flow for any client-initiated 3.0 EJB interaction is as follows:



Step	Action
1	The client accesses the JNDI server to look up the EJBObject interface of an EJB deployed on the application server.
2	The client fetches a remote instance of a 3.0 EJB associated with the given JNDI Name with which Adapter for Enterprise Javabeans can interact. Adapter for Enterprise Javabeans returns a remote EJB interface to the client.
3	The client then invokes the desired public method on the remote EJB.

The following diagram illustrates the use of Adapter for Enterprise Javabeans and Integration Server in a typical business-process integration.


Various backend systems send data to webMethods via an IS client, invoking a service on Integration Server that updates account information.
A flow service on Integration Server invokes an adapter service created using the Adapter for EJB. The adapter service uses an adapter connection to connect to the application server. The service invokes a public method in the AccountHome EJB on an EJB Server.
Legacy System
Java, C/C++, VB webMethods Client
Account
ERP System
webMethods Integration Server
Services
Account
Web browser or webMethods Client
Account
INTERNET
webMethods Integration Server
Flow Service
Adapter for EJB
Adapter Service
Adapter Connection
Application Server
AccountHome EJB
AccountHome EJB writes new Account info to the database.
database

Adapter Package Management

Adapter for Enterprise Javabeans is provided as a package called WmEJBAdapter that you manage like any package on Integration Server.

There are several considerations regarding how you set up and effectively manage your packages on Integration Server, such as those described in the following list.

- Configure user-defined packages for your adapter connections and adapter services. See "Managing the Adapter Package" on page 58 for details.
- Understand how package dependencies work so you make the best decisions regarding how you manage your adapter services. See "Package Dependency Requirements and Guidelines" on page 59 for details.
- Control which development groups have access to which adapter services. See "Controlling Group Access" on page 61 for details.
- Understand how clustering, an advanced feature of Integration Server, works to effectively manage your adapter services. See "Using Adapter for Enterprise Javabeans in a Clustered Environment" on page 62 for details.

- Enable and disable packages. See "Enabling Packages" on page 60 and "Disabling Packages" on page 59 for details.
- Load, reload, and unload packages. See "Loading, Reloading, and Unloading Packages" on page 60 for details.

Adapter Connections

An adapter connection enables an Adapter for Enterprise Javabeans service to connect to a supported application server's JNDI service to discover and obtain remote references to EJBs deployed on that server.

The adapter supports three types of adapter connections: those that support local, single-phase commit transactions (EJB Local Connection); those that support distributed, two-phase commit (XA) transactions (EJB XA Connection); and those that are non-transacted (EJB Non-transactional Connection). Transacted connections are managed by Integration Server's built-in transaction manager. For more information about the transaction types, see "Transaction Management of Adapter for Enterprise Javabeans Connections" on page 19.

You configure one or more connections at design time to use in integrations. The number of connections you configure and the types of those connections depend on the nature of the specific EJBs you want to interact with and your integration needs. For example:

- Suppose you need to create an integration that will invoke a method on an EJB that returns the total number of hits on a Web site. Because this is essentially a read-only operation, you should configure a non-transacted connection to use with this EJB method. (It would not make sense to incur the additional overhead imposed by using either of the transacted connection types.)
- Or you may need to interact with an EJB that maintains session state information on the server. A shopping cart is a good example of such an EJB. In this case, you should use one of the transacted connection types so that if any single method call on the EJB fails, all previous method calls will be automatically rolled-back. (This example assumes that the EJB is configured to support client-initiated transactions.)

When you configure an adapter connection, you specify parameters (such as the connection type, the connection's name and location, the location of the JNDI properties file, and a JNDI username and password, and a caching level) that Integration Server uses to manage connections to the application server. You configure connections using Integration Server Administrator. You must have webMethods administrator privileges to access Adapter for Enterprise Javabeans's administrative screens.

For instructions for configuring, viewing, editing, enabling, and disabling Adapter for Enterprise Javabeans connections, see "Adapter Connections" on page 67. For information about setting user privileges, see the *webMethods Integration Server Administrator's Guide* for your release.

For a list of tasks that you must do before you can create your connections, see "Before Configuring or Managing Adapter Connections" on page 68.

Adapter Connection Templates

Adapter for Enterprise Javabeans provides the following adapter connection templates:

Adapter Connection Template	Description
EJB Non-transactional Connection	Creates connections that will not be transacted.
EJB Local Connection	Creates connections that can be used in non-distributed, local transactions.
EJB XA Connection	Creates connections that can be used in distributed, two-phase commit transactions.

For more information about the connections, see "Transaction Management of Adapter for Enterprise Javabeans Connections" on page 19. For more information about using the adapter connection templates, see "Configuring Adapter Connections" on page 69.

Connection Pools

Integration Server includes a connection management service that dynamically manages connections and connection pools based on configuration settings that you specify for the connection. By default, connection pooling is enabled for all adapter connections.

A connection pool is a collection of connections with the same set of attributes. Integration Server maintains connection pools in memory. Connection pools improve performance by enabling adapter services to reuse open connections instead of opening new connections.

Run-Time Behavior of Connection Pools

When you enable a connection, Integration Server initializes the connection pool, creating the number of connection instances you specified in the connection's **Minimum Pool Size** parameter. Whenever an adapter service needs a connection, Integration Server provides a connection from the pool. If no connections are available in the pool, and the maximum pool size has not been reached, the server creates one or more new connections (according to the number specified in **Pool Increment Size**) and adds them to the connection pool. If the pool is full (as specified in **Maximum Pool Size**), the requesting service will wait for Integration Server to obtain a connection, up to the length of time specified in the **Block Timeout** parameter, until a connection becomes available. Periodically, Integration Server inspects the pool and removes inactive connections that have exceeded the expiration period that you specified in **Expire Timeout**.

If the connection pool initialization fails because of a network connection failure or some other type of exception, you can enable the system to retry the initialization any number of times, at specified intervals.

For information about configuring connections, see "Adapter Connections" on page 67.

Built-In Services For Connections

Integration Server provides built-in services that enable you to programmatically control connections. You can use them to enable and disable a connection, and to return usage statistics and the current state (Enabled or Disabled) and error status for a connection. These services are located in the WmART package, in the pub.art.connection folder.

The built-in service setAdapterServiceNodeConnection enables you to change the connection associated with an adapter service. See "Changing the Connection Associated with an Adapter Service at Design Time" on page 20.

For details about the WmART services, see the *webMethods Integration Server Built-In Services Reference* for your release.

Transaction Management of Adapter for Enterprise Javabeans Connections

Adapter for Enterprise Javabeans connections support the following transaction types:

Transaction Type	Description
Non-transactional	Used by the EJB Non-transactional Connection connection template. The connection provides no transaction control over the operations being performed. Once completed, an operation cannot be reversed. That is, the connection automatically commits (Auto Commit) all operations.
Local	Used by the EJB Local Connection connection template. With this transaction type, all of the operations on the same connection in one transaction boundary will be committed or rolled back together. A transaction boundary means the scope of the transaction, from the beginning to the end of a transaction. It can be in one adapter service, one flow service, one Java service, or several steps in a flow service. For all supported versions of all supported application servers, Integration Server transaction manager can accommodate any number of adapter services using local transaction connections in a single flow.
XA	Used by the EJB XA Connection connection template.This transaction type allows the connection to participate in two-phase transactions executed across multiple distributed resources. In one transaction boundary, all of the operations on multiple connections will be committed or rolled back together. A transaction boundary means the scope of the transaction, from the beginning to the end of a transaction. It can be in one adapter service, one flow service, one Java service, or several steps in a flow service.

When you configure a connection, the connection template you choose determines the type of transaction management that the connection's operations use implicitly. Implicit transactions are managed by Integration Server's transaction manager. For more information about implicit transactions, "Implicit and Explicit Transactions" on page 140.

You can also explicitly manage transactions using built-in services. See "Overview" on page 148 for information about explicitly managing transactions.

Changing the Connection Associated with an Adapter Service at Design Time

Integration Server solves this limitation by providing a built-in service that you can use at design time to change the connection associated with an adapter service. This built-in service is named setAdapterServiceNodeConnection and is provided in the WmART package's pub.art.service folder. Using this function, you can change the specific connection associated with an adapter service at design time so that you do not need to create and maintain multiple adapter services.

Note:
This built-in service can be run at design time only; do not use it within Integration Server flow or Java service. You must run this service directly from Designer by selecting the service and running it.

For details about setAdapterServiceNodeConnection, see the *webMethods Integration Server Built-In Services Reference* for your release.

Other built-in services enable you to control connections; for more information, see "Built-In Services For Connections" on page 19.

Changing the Connection Associated with an Adapter Service at Run Time

Integration Server enables you to dynamically select the connection a service uses to interact with the adapter's resource. This feature enables one service to interact with multiple, similar backend resources.

For example, you can configure an adapter service to use a default connection that interacts with your company's production application server. However, at runtime you can override the default connection and instead use another connection to interact with the company's test application server.

For more information about overriding a service's default connection at runtime, see "Dynamically Changing a Service's Connection at Runtime" on page 73.

EJB Transaction Management

When configuring adapter connections to be used by adapter services, you must also consider how or whether the EJB uses container demarcation to manage its transactions. With container demarcation, the EJB client may call an EJB with an external transaction context, or not. The value of the `<trans-attribute>` deployment descriptor determines how transactions are handled. A transaction attribute can have the following values:

Transaction Attribute Value	Description
NotSupported	Indicates an "unspecified transaction context." The client's transaction context is suspended until the bean method completes.
Supports	The client's transaction context is propagated to the bean. If there is no client context, the bean method is executed under "unspecified context."
Required	The client's transaction context is propagated to the bean. If no client context is specified, the container creates a new context for the transaction.
RequiresNew	The client's transaction context (if any) is suspended and a new context is always created.
Mandatory	The client's transaction context is propagated to the bean. If no client context is specified, the container throws a TransactionRequiredException.
Never	The client's transaction context is prohibited. If a client context is passed, it causes the container to throw a RemoteException. Otherwise, the container operates the transaction with "unspecified transaction context."

JNDI Properties File

Adapter for Enterprise Javabeans is essentially an EJB client application, and must first be able to access JNDI itself before it can gain access to deployed EJBs. This information is provided in a JNDI properties file. JNDI defines a number of standard Java properties that a client may use for the purpose of establishing a connection. Before configuring a connection, you (or an administrator) create a separate .properties file containing the JNDI property definitions.

When configuring a connection, you must provide the full path to the JNDI properties file used by the application server. This file contains definitions of the JNDI-specific Java properties required to establish a connection with the application server's JNDI, but can also be used to hold other Java properties as well. The format of this file follows the conventions for defining Java system properties as described in the java.utils.Property Javadoc.

Note:
Adapter for Enterprise Javabeans copies all properties defined in a properties file to both the javax.naming.InitialContext constructor and the system properties list. Because these properties are visible across the JVM and Integration Server, changing one or more of these may impact other, non-related system components after the connection has been enabled. We recommend that you limit the properties in this file to the properties that are documented in this guide for your application server vendor and version. For more information about JNDI properties, see "Application Server Configuration Notes" on page 163, and see the section for your application server.

Required Properties and Values

An adapter connection requires that values be set in the JNDI properties file for the following standard JNDI properties. The values for these properties are specific to each application server:

Required JNDI Property	Application Server and Value
java.naming.factory.initial	**For WebLogic, specify:** weblogic.jndi.WLInitialContextFactory **For WebSphere, specify:** com.ibm.websphere.naming.WsnInitialContextFactory **For JBoss, specify:** org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs	**For JBoss only, specify:** org.jboss.naming:org.jnp.interfaces
java.naming.provider.url	**For WebLogic:** ■ **If using T3 (non-SSL) protocol, specify:** t3://*WebLogic_server*:*port_number* ■ **If using T3S protocol, specify:** t3s://*WebLogic_server*:*port_number* ■ **If using HTTP protocol, specify:** http://*WebLogic_server*:*port_number* ■ **If using HTTPS protocol, specify:** https://*WebLogic_server*:*port_number* **Note:** For additional information about SSL and encryption with WebLogic, see “Encryption” on page 169. **For WebSphere, specify:** iiop://*WebSphere_server*:*port_number* **For JBoss, specify:** jnp://*JBoss_server*:*port_number*

Some supported application servers may require that additional properties be defined, or may allow other properties for optional use. For more information about JNDI properties, see

"Application Server Configuration Notes" on page 163, and see the section for your application server.

Specifying JNDI Credentials

If your JNDI server requires authentication, you can use the following properties to set a username and password for access to the JNDI server:

- java.naming.security.principal=*username*
- java.naming.security.credentials=*password*

You can either include the properties and values in the JNDI properties file or you can specify the values when you configure an adapter connection. Do not use a combination of these two methods: enter both properties in a JNDI properties file or specify them both when configuring an adapter connection.

To specify the credentials when configuring a connection, use the **JNDI Username** parameter and the **JNDI Password** parameter on the Connection Types screen, as follows:

- If you specify a username in the **JNDI Username** parameter, this value is assigned to the java.naming.security.principal property. If this property is also included in the JNDI properties file, the value in the **JNDI Username** parameter overrides the value in the properties file.
- If you specify a password in the **JNDI Password** parameter, this value is assigned to the java.naming.security.credentials property. If this property is also included in the JNDI properties file, the value in the **JNDI Password** parameter overrides the value in the properties file.

  Providing the JNDI password in the **JNDI Password** parameter is more secure than defining the password in the JNDI properties file. This is because Integration Server Administrator automatically encrypts the value in the **JNDI Password** parameter. The JNDI properties file, on the other hand, is never encrypted. The password will be in clear text and can be seen by any user having the appropriate access to that file within the local operating system.

For information about these parameters on the Connection Types screen, see the instructions in "Configuring Adapter Connections" on page 69. For additional information about JNDI security credentials, see "Application Server Configuration Notes" on page 163, and see the section for your application server.

Security Considerations

The system administrator should restrict access to the following files and directories:

- **JNDI Properties Files.** All properties defined in the JNDI properties file used to create an adapter connection will be copied into the JVM's system properties list. This feature provides the adapter with a certain degree of flexibility in how it may interact with the application server. It also provides a mechanism through which arbitrary string values can be introduced into a running JVM.

  Due to the nature of this feature, access should be restricted to all JNDI properties files used with Adapter for Enterprise Javabeans connections. Certain users and processes will need to read these files (the Integration Server process, for instance). Other users will need to create and modify these files. The system administrator should use the features of the local operating system to restrict access to these files to only those users.

- **Classes and Supported Directories.**The `WmEJBAdapter\code\classes` directory and, in particular, the `WmEJBAdapter\code\classes\com\wm\adapter\wmejb\connection\impl\supported` directory contain façade classes that determine which application server vendors and versions are supported. Adding or removing class files from the supported directory could have undesired consequences. The system administrator should use the features of the local operating system to restrict access to both of these directories.

EJB Information Caching

An adapter connection may be configured to cache information retrieved from the JNDI server. This information is available to all connection instances created by that adapter connection. It is retained in memory for a period of time determined by the level of caching selected. Information in the cache is used during design time when you configure adapter services and assign a connection. Caching has no impact on adapter services or connections at runtime.

Caching allows the adapter connection to minimize the number of round-trip accesses to the JNDI server for the purpose of retrieving information about the EJBs bound in that server. Typically, this information is static: an EJB that was available yesterday is likely still available today. For large EJB installations, the costs associated with retrieving this information repetitively from JNDI may be prohibitive.

Which caching level works best varies from site to site, and is influenced by a variety of external factors: for example, network overhead, capacity of the JNDI server, capacity of Integration Server, and size of the JNDI tree. In general, no caching (None) should be avoided on development systems and Hard caching should be avoided on production systems.

Adapter for Enterprise Javabeans uses the cache level specified for the connection when you configure an adapter service using Designer's adapter service editor. Adapter for Enterprise Javabeans supports the following caching levels:

Caching Level	Description
None	No caching occurs. When configuring the adapter service, the connection retrieves EJB details directly from the JNDI. This level of caching is suitable when working against a very small set of deployed EJBs with little network latency between Adapter for Enterprise Javabeans and the application server.
Weak	Based on the Java WeakReference implementation. Information in the cache is retained as long as there are outstanding weak references to the cache. When no such references remain, the JVM reclaims the memory consumed by the cache.
Soft	Based on the Java SoftReference implementation. Information in the cache is retained until the JVM is forced to throw an OutOfMemoryError

Caching Level	Description
	in response to a memory allocation request. At this point, it frees the memory in the cache.
Hard	Data is retrieved from the JNDI once and cached until no more connections remain enabled for the connection factory. Subsequent requests for EJB detail are retrieved from the cache. This level of caching results in the fewest round-trip lookups on JNDI at the expense of more virtual memory being tied up in Integration Server for longer periods of time.

Adapter Services

Adapter services allow you to connect to the adapter's resource (that is, an application server) and initiate an operation on the resource from Integration Server.

You call adapter services from a flow or Java service to interact with EJB instances on an application server. For information about using the services in a flow, see "Creating Flows for Adapter for Enterprise Javabeans Services" on page 147. To see how to use the services in different operations, see "Scenarios" on page 129.

At design time, the adapter obtains information about each EJB deployed on the application server. For each EJB, this information includes its home interface (for 2.1 adapter services only), the home methods it exposes and their signatures, and its corresponding remote interface methods. The adapter gets this information directly from JNDI or from the local cache. From this information, you configure adapter services. Integration Server then uses adapter connections that you defined earlier to execute the adapter services.

You configure adapter services using the templates provided with Adapter for Enterprise Javabeans. Each template represents a specific technique for doing work on a resource, such as creating an EJB on an application server. An adapter service template contains all the code necessary for interacting with the resource but without the data specifications. You provide these specifications when you configure a new adapter service. The adapter provides different versions of templates for creating adapter services for use with the EJB 3.0 or EJB 2.1 standards.

You use Designer to configure the adapter service. Some familiarity with using Designer is required. For more information, see the *webMethods Service Development Help* for your release.

Configuring a new service from an adapter service template is straightforward. Using Designer, you assign the service an adapter connection. After you select the connection for the adapter service, you select the adapter service template to use. At this point, depending on which template you have chosen, the adapter automatically populates EJB-specific fields and menus in the adapter service editor tabs. You then use the editor to manipulate these fields to configure the adapter service.

The input signature for an adapter service template is predefined and depends on the specific signature(s) of the EJB method(s) selected. You can re-name the method parameters, but you cannot re-define the types of any method parameters.

The output signature is predefined and is always the same for a given template class. You cannot change any of the parameter names or re-define the types.

Each adapter service you configure can be used as a standalone, executable entity. However, you can also incorporate your adapter services in flow services or Java services to implement some business workflow. Using the Flow Service Editor in Designer, you can create intelligent business applications wrapped around Adapter for Enterprise Javabeans services. For more information about creating flows with adapter services you configure using Adapter for Enterprise Javabeans, see “Creating Flows for Adapter for Enterprise Javabeans Services” on page 147.

Supported Bean Types

The adapter supports the following types of EJBs:

- Entity beans 2.0 and 2.1
- Stateless session beans 2.0, 2.1, and 3.0
- Stateful session beans 2.0, 2.1, and 3.0

There is no support for message-driven EJBs.

Services and Transaction Management

Adapter for Enterprise Javabeans is designed to interact with Integration Server's built-in transaction manager. When designing and implementing transactions, and especially distributed transactions, you must not only be familiar with the capabilities of the built-in transaction manager, but also with those of the application server in general, as well as the particular EJBs involved. For example, configuring an adapter service in the following scenarios will cause the adapter to generate an error at runtime:

- Configuring an adapter service to use an EJB that has the Mandatory transaction attribute value against a non-transactional connection.
- Configuring an adapter service to use an EJB that has the Never transaction attribute value against a local or XA connection.
- Calling the RemoveEJB service against a stateful session EJB deployed on WebSphere or WebLogic within a transaction. See “Removing EJBs” on page 93 for more information about the RemoveEJB service.

The EJB standard does not provide a way for a client to determine the transaction attributes of an EJB, so these situations cannot be detected when configuring an adapter service. For more information about transaction management, see “Transaction Management of Adapter for Enterprise Javabeans Connections” on page 19. For information about transaction attributes, see “EJB Transaction Management” on page 20.

Configuring Adapter for Enterprise Javabeans for Adapter Services

It is necessary to configure Adapter for Enterprise Javabeans for the adapter services. The file used for configuring the adapter services, config.xml, is available in the *Integration Server_directory* \ `instances\`*`instance_name`*`\packages\WmEJBAdapter\config` folder.

Specifying the config.xml File for Adapter Services

The config.xml file has two sets of elements:

- Application level elements comprising `ejbConfig`, `appServer`, `restrictedJNDINames`, `jndiTreeSearchOptions`, and `jndiContexts`.
- EJB level elements comprising `jndiNames` and `ejbClientJars`.

Important:
For 2.1 adapter services, both the application level and EJB level elements are optional. For 3.0 adapter services, the application level elements are optional, whereas the EJB level elements are mandatory. Adapter for Enterprise Javabeans cannot work with the 3.0 adapter services if neither of the EJB level elements are specified.

The elements that are available in config.xml are:

Element	Description	
ejbConfig	This element is the root element of the config.xml file and contains only the application server elements.	
appServer	This element is specific to every instance of the application server, and is used to customize the application server configuration for Adapter for Enterprise Javabeans. Specify a value of "*" for providerURL to use the default instance of the application server. To list multiple instances of application servers, specify the configuration of the sub-elements listed below within the `<appserver>... </appserver>` XML tag. Specify the JNDI provider URL of the application server as a value for providerURL for every element of `<appServer>`.	

	restrictedJNDINames	Contains the list of JNDI names that are to be avoided when traversing through the JNDI tree.
		
		Update this list with objects that are not accessible to the different application servers or different versions of the same application server.
		
		The structure of this element is:
		`<restrictedJNDINames>`
		`  <jboss>`

```
    <jndiName>jndiName</jndiName>
    .
    .
    .
  </jboss>
  <weblogic>
    <jndiName>jndiName</jndiName>
    .
    .
    .
  </weblogic>
  <websphere>
    <jndiName>jndiName</jndiName>
    .
    .
    .
  </websphere>
</restrictedJNDINames>
```

Element	Description	
	jndiTreeSearchOptions	Defines the options for searching the JNDI tree. The options available are <contextSearch> and <nameSearch>. If the <contextSearch> option is set to true, the application server searches the JNDI tree based on the contexts that are specified in the <jndiContexts> sub-element. If the <nameSearch> option is set to true, the application server searches the JNDI tree based on the jar files names specified in the `<ejbClientJars>` sub-element. Both the options, <contextSearch> and `<nameSearch>`, for any instance of the application server, can be set to true to search on both context and jar file name. If both the options are set to false, the application server traverses through the entire JNDI tree.
	jndiContexts	In the <jndiTreeSearchOptions>, if the `<contextSearch>` option is set to true, specify a list of JNDI context elements, based on the contexts to be searched. Each JNDI context is represented in the format of the JNDI tree.
	jndiNames	Required for 3.0 adapter services. In the <jndiTreeSearchOptions>, if the `<nameSearch>` option is set to true, specify a list of `<jndiName>` elements, based on the `<jndiNames>` to be searched.

Element	Description	
		The <jndiName> is represented in the format of the JNDI tree.
	ejbClientJars	Required for 3.0 adapter services. Specify the jar files to be executed by the adapter services. The value of each <jarFileName> element must be either the absolute path of the jar file or the name of the jar file, only if the jar file is available at \\WmEJBAdapter\code\jars.

The following is an example of the format entries in a config.xl file:

```
<ejbConfig>
<appServer providerURL="*">
<restrictedJNDINames>
  <jboss>
     <jndiName>jboss.logging.DomainLogHandler</jndiName>
     .
     .
     .
  </jboss>
  <weblogic>
      <jndiName>weblogic.logging.DomainLogHandler</jndiName>
     .
     .
     .
  </weblogic>
  <websphere>
      <jndiName>websphere.logging.DomainLogHandler</jndiName>
      .
      .
      .
  </websphere>
</restrictedJNDINames>

<jndiTreeSearchOptions>
           <contextSearch>true</contextSearch>
           <nameSearch>true</nameSearch>
</jndiTreeSearchOptions>

<jndiContexts>
          <jndiContext>thisNode/servers/server1/WSsamples</jndiContext>
          <jndiContext>thisNode/servers/server1/WSsamples1</jndiContext>
           .
           .
           .
</jndiContexts>

<jndiNames>
           <jndiName>WSsamples/BasicCalculator1</jndiName>
           <jndiName>WSsamples/BasicCalculator2</jndiName>
            .
            .
            .
</jndiNames>
```

```
<ejbClientJars>
           <jarFileName>BasicCalculatorEJBClient.jar</jarFileName>
            .
            .
            .
</ejbClientJars>
</appServer>
</ejbConfig>
```

Adapter Service Templates

There are two activities a client may perform against an EJB: creating a new EJB instance (or finding an existing one) and executing the remote methods exposed by that EJB. Accordingly, Adapter for Enterprise Javabeans provides the following adapter service templates:

Adapter Service Template	Description
CreateEJB 2.1	Creates one or more instances of a single remote 2.1 EJB class and returns the EJB handles to the caller. See "CreateEJB 2.1 Adapter Service" on page 30 for more information.
InvokeEJB 2.1	Invokes a single method on an existing 2.1 EJB. See "InvokeEJB 2.1 Adapter Service" on page 32 for more information.
CreateInvokeEJB 2.1	Creates one or more instances of a single 2.1 EJB class and invokes a single method on those instances. See "CreateInvokeEJB 2.1 Adapter Service" on page 34 for more information.
FetchEJB 3.0	Creates one or more instances of a single remote 3.0 EJB class and returns the EJB handles to the caller. See "FetchEJB 3.0 Adapter Service" on page 36 for more information.
InvokeEJB 3.0	Invokes a single method on an existing 3.0 EJB. See "InvokeEJB 3.0 Adapter Service" on page 37 for more information.
FetchInvokeEJB 3.0	Creates one or more instances of a single 3.0 EJB class and invokes a single method on those instances. See "FetchInvokeEJB 3.0 Adapter Service" on page 39 for more information.

CreateEJB 2.1 Adapter Service

Services configured with the CreateEJB 2.1 template create one or more instances of a single remote 2.1 EJB class and return the EJB handles to the caller. A CreateEJB 2.1 service does not invoke any methods on the EJBs it creates. You can create the following types of EJBs: entity beans, stateless session beans, and stateful session beans.

At design time, you provide the following inputs in the adapter service editor: the JNDI lookup name of the EJB to create and the identity of the EJBHome creator/finder method to be executed by the service. If the method has parameters, you can override the default names given to the parameters in Designer's adapter service editor. See "Configuring CreateEJB 2.1 Services" on page 81 for more information about the parameters.

In a CreateEJB 2.1 service's input signature, the document name used to contain EJB method arguments is pre-configured and localized. The input document name is EJBHome_Args and is only visible if the home method takes arguments.

The output of a CreateEJB 2.1 adapter service is the remote EJB(s), and typically serves as the input to an InvokeEJB 2.1 adapter service. For more information, see "InvokeEJB 2.1 Adapter Service" on page 32.

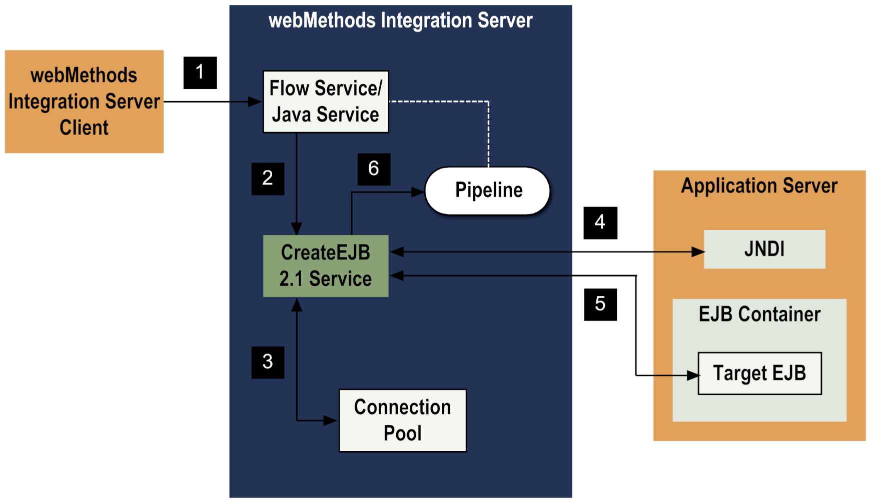
Run-Time Processing for a CreateEJB 2.1 Service

At run time, the service's only inputs are the parameter values required by the EJBHome method (if any).

If the operation is successful, the CreateEJB 2.1 service returns to the caller an array of javax.ejb.Handle instances that represent the remote EJBs found or created by the EJBHome method. These handles may subsequently be de-serialized by the caller to obtain the underlying EJBs or may be passed into a suitably configured InvokeEJB 2.1 adapter service.

Note:
In general, session EJBHome methods will always return a single EJBObject, entity EJBHome methods (for example, finder methods) may return multiple EJBObjects. To accommodate the possibility of multiple objects being returned, a CreateEJB 2.1 service always wraps the output of the create/find method in an array.

If the CreateEJB 2.1 service fails, it throws an adapter exception.



Step	Description
1	An Integration Server client runs a flow service or Java service on Integration Server.
2	The flow or Java service invokes a CreateEJB 2.1 adapter service on Integration Server. You configured the adapter service earlier using Designer.
3	The CreateEJB 2.1 adapter service gets a connection from the service's connection pool.

Step	Description
	You created and enabled the adapter connection earlier using Integration Server Administrator.
4	Through its connection, the CreateEJB 2.1 adapter service accesses the application server's JNDI to look up the EJBHome interface of the EJB for which the service was configured. JNDI will return the remote stub representing this EJBHome.
5	The CreateEJB 2.1 adapter service invokes the configured EJBHome method on the application server. If the method takes any parameters, the values are extracted from the pipeline and passed to the remote method. If the operation is successful, the adapter service returns an array of javax.ejb.Handle instances that represent the remote EJBs. If the operation is unsuccessful, the adapter service throws an AdapterException or AdapterConnectionException. For more information about how the adapter handles exceptions, see Adapter Logging and Exception Handling.
6	The CreateEJB 2.1 adapter service saves the resulting remote EJB handle(s) and status on pipeline. These handles may then be de-serialized by the caller to obtain the underlying EJBs or simply passed into a suitably configured InvokeEJB 2.1 adapter service.

InvokeEJB 2.1 Adapter Service

Services configured with the InvokeEJB 2.1 template invoke a single method on an existing 2.1 EJB.

At design time, an InvokeEJB 2.1 service requires two inputs: the JNDI lookup name of the EJB and the identity of the bean method it should invoke. If the method has parameters, you can override the default names given to the parameters in Designer's adapter service editor. For more information about the parameters, see “Configuring InvokeEJB 2.1 Services” on page 83. In an InvokeEJB 2.1 service's input signature, the document name used to contain EJB method arguments is pre-configured and localized. The input document name is EJBObject_Args and is only visible if the bean method takes arguments.

Note that the output of a CreateEJB 2.1 adapter service (the remote EJBs it created) can serve as the input to an InvokeEJB 2.1 adapter service. Because the output of CreateEJB 2.1 is an array of one or more Handle objects representing EJBs of a specific class, you should configure an InvokeEJB 2.1 adapter service to work with the same EJB class returned by the CreateEJB 2.1 service.

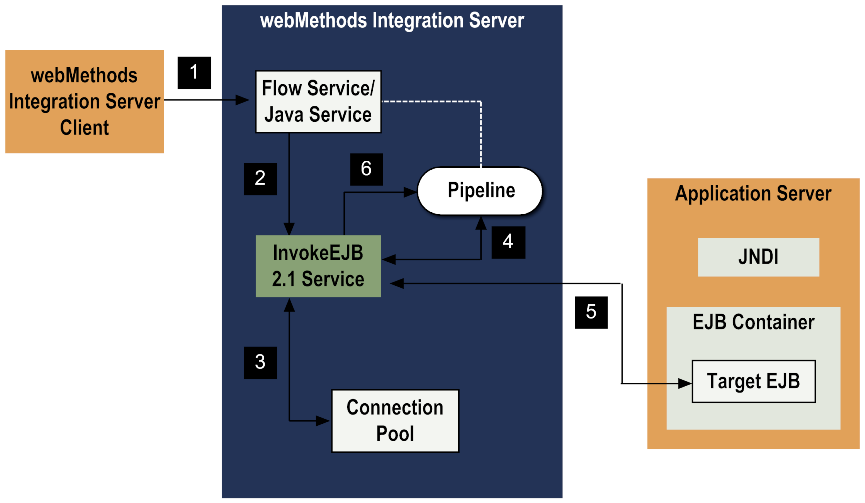
Important:
Be careful when naming instances of these CreateEJB 2.1 and InvokeEJB 2.1 adapter services to avoid mismatching them. If a pair of CreateEJB 2.1 and InvokeEJB 2.1 services are mismatched in a flow, this condition will not be detected until that flow is executed.

When adding an InvokeEJB 2.1 adapter service to a flow, note that:

- Any execution of an InvokeEJB 2.1 service must be preceded by an execution of a CreateEJB 2.1 service. Both services must be configured to operate against the same EJB class.
- Any execution of an InvokeEJB 2.1 service will always require at least one input: the output of a prior CreateEJB 2.1 execution. That is, before you can execute an EJB method, you must first obtain that EJB. For more information, see "Obtaining an EJB" on page 148.

Run-Time Processing for an InvokeEJB 2.1 Service

At run time, an InvokeEJB 2.1 service expects to receive a single instance of javax.ejb.Handle along with any parameter values needed by the bean method. The Handle object represents the EJB to invoke the remote method against. Its output depends on the output signature of the method it invokes. If the service fails, it throws an adapter exception.



Step	Description
1	An Integration Server client runs a flow service or Java service on Integration Server.
2	The flow or Java service invokes an InvokeEJB 2.1 adapter service on Integration Server. You configured the adapter service earlier using Designer.
3	The InvokeEJB 2.1 adapter service gets a connection from the service's connection pool. You created and enabled the adapter connection earlier using Integration Server Administrator.
4	The InvokeEJB 2.1 adapter service gets the EJB object handle from the pipeline.
5	Through its connection, the InvokeEJB 2.1 adapter service invokes the configured EJBObject method on the application server. If the method takes any parameters, the values are extracted from the pipeline and passed to the remote method. If the operation is successful, the adapter service returns the method's output, if any.

Step	Description
	If the operation is unsuccessful, the adapter service throws an AdapterException or AdapterConnectionException. For more information about how the adapter handles exceptions, see Adapter Logging and Exception Handling.
6	The InvokeEJB 2.1 adapter service saves the output of the method and status on pipeline.

CreateInvokeEJB 2.1 Adapter Service

A CreateInvokeEJB 2.1 service combines the functionality of the CreateEJB 2.1 and InvokeEJB 2.1 adapter services and the RemoveEJB Java service into one service. Services configured with this template create one or more instances of a single 2.1 EJB class and invoke a single method on those instances. Unlike CreateEJB 2.1, the remote EJB instances are not returned to the caller. CreateInvokeEJB 2.1 automatically calls RemoveEJB for each non-entity EJB instance; however, RemoveEJB is never called for an entity bean. See “Removing EJBs” on page 93 for more information about the RemoveEJB service.

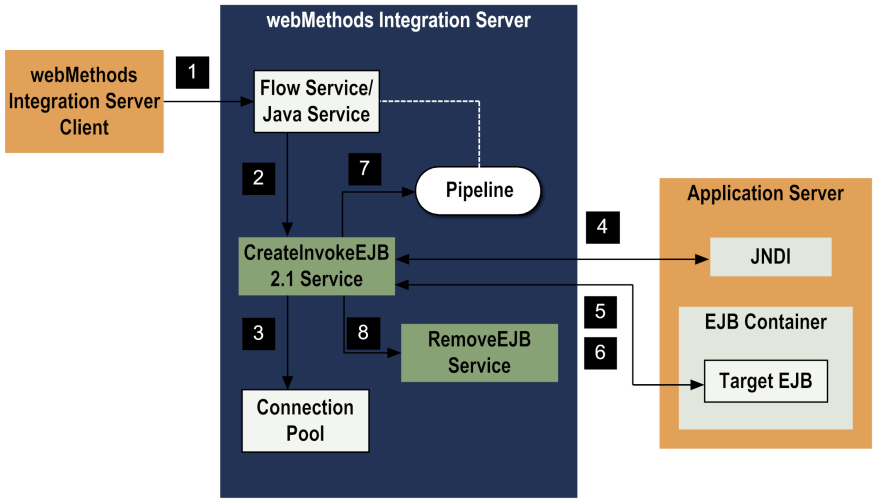
Use the CreateInvokeEJB 2.1 template when you want to create an adapter service that will retrieve an EJB, run a method on that bean, then release it. Because it does not return any Handle objects in its output, the lifecycle of the EJB is entirely contained within the bounds of the adapter service: the EJB is created, a single method is invoked and its output captured, then its remove() method is called. For this reason, a CreateInvokeEJB 2.1 service is not particularly useful with stateful session beans.

At design time, a CreateInvokeEJB 2.1 service requires the JNDI lookup name of the EJB to create, the identity of the create method to invoke, and the identity of the bean method to invoke. As with the other service templates, you may override the default names of any parameter used in a home or remote method. See “Configuring CreateInvokeEJB 2.1 Services” on page 85 for more information about the parameters.

In a CreateInvokeEJB 2.1 service's input signature, the document names used to contain EJB method arguments are pre-configured and localized. The input document name of the home method is EJBHome_Args and the input document name of the bean method is EJBObject_Args. The documents appear in the input signature only if the corresponding methods have any arguments.

Run-Time Processing for a CreateInvokeEJB 2.1 Service

At run time, a CreateInvokeEJB 2.1 service instance requires any parameter values the home and/or bean methods require for inputs. Its output is the output of the remote bean method itself, if any. The adapter service wraps the bean's output in an array in case multiple objects are returned.


webMethods Integration Server
webMethods Integration Server Client
1
Flow Service/ Java Service
2
7
Pipeline
CreateInvokeEJB 2.1 Service
3
8
RemoveEJB Service
Connection Pool
4
5
6
Application Server
JNDI
EJB Container
Target EJB

Step	Description
1	An Integration Server client runs a flow service or Java service on Integration Server.
2	The flow or Java service invokes a CreateInvokeEJB 2.1 adapter service on Integration Server. You configured the adapter service earlier using Designer.
3	The CreateInvokeEJB 2.1 adapter service gets a connection from the service's connection pool. You created and enabled the adapter connection earlier using Integration Server Administrator.
4	Through its connection, the CreateInvokeEJB 2.1 adapter service accesses the application server's JNDI to look up the EJBHome interface of the EJB for which the service was configured. JNDI will return the remote stub representing this EJBHome.
5	The CreateInvokeEJB 2.1 adapter service invokes the configured EJBHome method on the application server. If the method takes any parameters, the values are extracted from the pipeline and passed to the remote method. If the operation is successful, the adapter service obtains an array of javax.ejb.Handle instances that represent the remote EJBs. If the operation is unsuccessful, the adapter service returns an AdapterException or AdapterConnectionException. For more information about how the adapter handles exceptions, see Adapter Logging and Exception Handling.
6	Using the handles obtained in the previous step, the CreateInvokeEJB 2.1 adapter service invokes the configured EJBObject method on the application server. If the method takes any parameters, the values are extracted from the pipeline and passed to the remote method. If the operation is successful, the adapter service returns the method's output, if any.

Step	Description
	If the operation is unsuccessful, the adapter service returns an AdapterException or AdapterConnectionException. For more information about how the adapter handles exceptions, see Adapter Logging and Exception Handling.
7	The CreateInvokeEJB 2.1 adapter service puts the output of the remote method invocation and the status on the pipeline.
8	The CreateInvokeEJB 2.1 adapter service has completed its processing for session EJBs, it calls the EJB-standard remove() method on each EJB object obtained in step 5.

FetchEJB 3.0 Adapter Service

The services configured with the FetchEJB 3.0 template find one or more instances of a single remote 3.0 EJB class and return the EJB handles to the caller. A FetchEJB 3.0 service does not invoke any methods on the EJBs it finds. You can create the following types of EJBs: stateless session beans and stateful session beans.

At design time, you provide the JNDI lookup name of the EJB to fetch. See "Configuring FetchEJB 3.0 Services" on page 88 for more information about the parameters.

The output of a FetchEJB 3.0 adapter service is the remote EJB(s), and typically serves as the input to an InvokeEJB 3.0 adapter service. For more information, see "InvokeEJB 3.0 Adapter Service" on page 37.

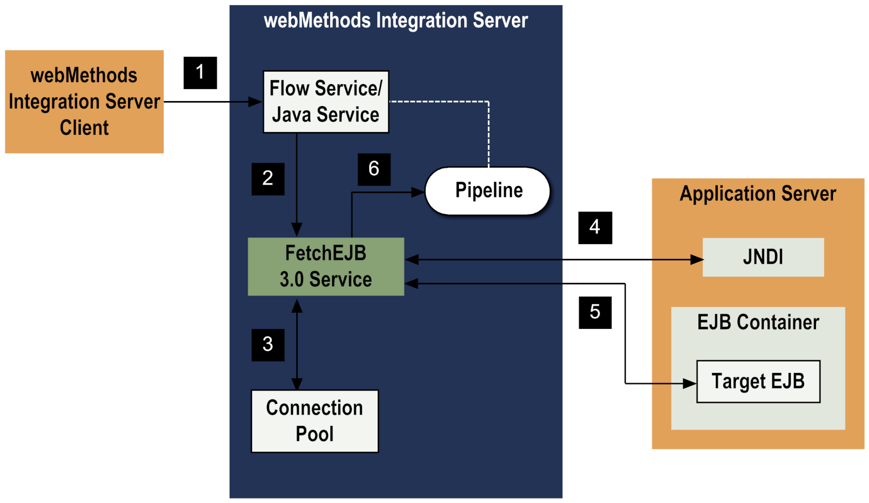
Run-Time Processing for a FetchEJB 3.0 Service

At run time, the service's only inputs are the parameter values required by the EJBObject method (if any).

If the operation is successful, the FetchEJB 3.0 service returns to the caller an array of javax.ejb.Handle instances that represent the remote EJBs found. These handles may subsequently be de-serialized by the caller to obtain the underlying EJBs or may be passed into a suitably configured InvokeEJB 3.0 adapter service.

Note:
In general, session EJBObject methods will always return a single EJBObject, entity EJBObject methods (for example, finder methods) may return multiple EJBObjects. To accommodate the possibility of multiple objects being returned, a FetchEJB 3.0 service always wraps the output of the service in an array.

If the FetchEJB 3.0 service fails, it throws an adapter exception.



Step	Description
1	An Integration Server client runs a flow service or Java service on Integration Server.
2	The flow or Java service invokes a FetchEJB 3.0 adapter service on Integration Server. You configured the adapter service earlier using Designer.
3	The FetchEJB 3.0 adapter service gets a connection from the service's connection pool. You created and enabled the adapter connection earlier using Integration Server Administrator.
4	Through its connection, the FetchEJB 3.0 adapter service accesses the application server's JNDI to look up the EJBRemote interface of the EJB for which the service was configured.
5	The FetchEJB 3.0 adapter service fetches the configured EJBObjects associated with the JNDI Name configured on the application server. If the operation is successful, the adapter service returns an array of javax.ejb.Handle instances that represent the remote EJBs. If the operation is unsuccessful, the adapter throws an AdapterException or AdapterConnectionException. For more information about how the adapter handles exceptions, see Adapter Logging and Exception Handling.
6	The FetchEJB 3.0 adapter service saves the resulting remote EJB handles and status on pipeline. These handles may then be de-serialized by the caller to obtain the underlying EJBs or simply passed into a suitably configured InvokeEJB 3.0 adapter service.

InvokeEJB 3.0 Adapter Service

Services configured with the InvokeEJB 3.0 template invoke a single method on an existing 3.0 EJB.

At design time, an InvokeEJB 3.0 service requires two inputs: the JNDI lookup name of the EJB and the identity of the bean method it should invoke. If the method has parameters, you can override the default names given to the parameters in Designer's adapter service editor. For more information about the parameters, see "Configuring InvokeEJB 3.0 Services" on page 89. In an InvokeEJB 3.0 service's input signature, the document name used to contain EJB method arguments is pre-configured and localized. The input document name is EJBObject_Args and is only visible if the bean method takes arguments.

Note that the output of a FetchEJB 3.0 adapter service (the remote EJBs it created) can serve as the input to an InvokeEJB 3.0 adapter service. Because the output of FetchEJB 3.0 is an array of one or more Handle objects representing EJBs of a specific class, you should configure an InvokeEJB 3.0 adapter service to work with the same EJB class returned by the FetchEJB 3.0 service.

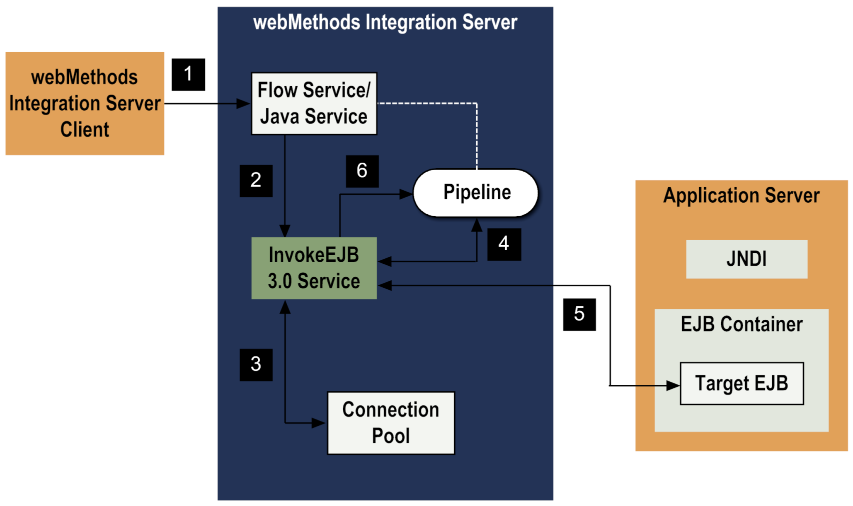
Important:
Be careful when naming instances of these FetchEJB 3.0 and InvokeEJB 3.0 adapter services to avoid mismatching them. If a pair of FetchEJB 3.0 and InvokeEJB 3.0 services are mismatched in a flow, this condition will not be detected until that flow is executed.

When adding an InvokeEJB 3.0 adapter service to a flow, note that:

- Any execution of an InvokeEJB 3.0 service must be preceded by an execution of a FetchEJB 3.0 service. Both services must be configured to operate against the same EJB class.
- Any execution of an InvokeEJB 3.0 service will always require at least one input: the output of a prior FetchEJB 3.0 execution. That is, before you can execute an EJB method, you must first obtain that EJB. For more information, see "Obtaining an EJB" on page 148.

Run-Time Processing for an InvokeEJB 3.0 Service

At run time, an InvokeEJB 3.0 service expects to receive a single instance of javax.ejb.Handle along with any parameter values needed by the bean method. The Handle object represents the EJB to invoke the remote method against. Its output depends on the output signature of the method it invokes. If the service fails, it throws an adapter exception.



Step	Description
1	An Integration Server client runs a flow service or Java service on Integration Server.
2	The flow or Java service invokes an InvokeEJB 3.0 adapter service on Integration Server. You configured the adapter service earlier using Designer.
3	The InvokeEJB 3.0 adapter service gets a connection from the service's connection pool. You created and enabled the adapter connection earlier using Integration Server Administrator.
4	The InvokeEJB 3.0 adapter service gets the EJB object handle from the pipeline.
5	Through its connection, the InvokeEJB 3.0 adapter service invokes the configured EJBObject method on the application server. If the method takes any parameters, the values are extracted from the pipeline and passed to the remote method. If the operation is successful, the adapter service returns the method's output, if any. If the operation is unsuccessful, the adapter service throws an AdapterException or AdapterConnectionException. For more information about how the adapter handles exceptions, see Adapter Logging and Exception Handling.
6	The InvokeEJB 3.0 adapter service saves the output of the method and status on pipeline.

FetchInvokeEJB 3.0 Adapter Service

A FetchInvokeEJB 3.0 service combines the functionality of the FetchEJB 3.0 and InvokeEJB 3.0 adapter services into one service. Services configured with this template fetch one or more instances of a single 3.0 EJB class and invoke a single method on those instances. Unlike FetchEJB 3.0, the remote EJB instances are not returned to the caller.

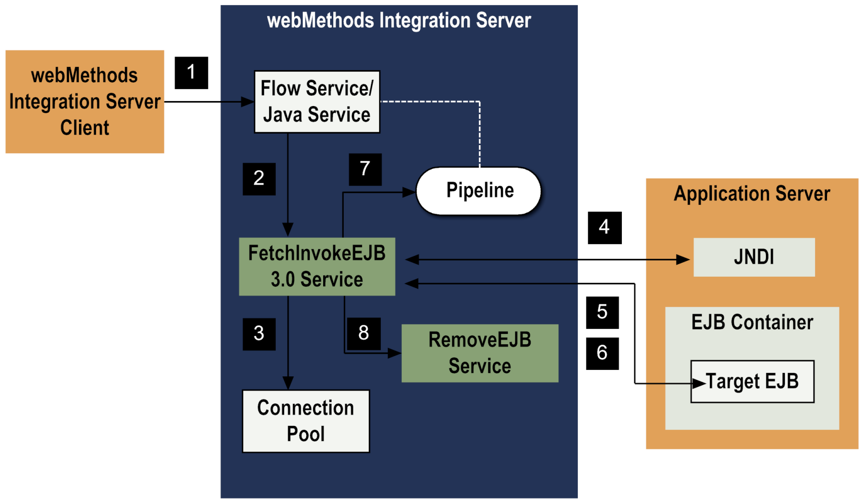
Use the FetchInvokeEJB 3.0 template when you want to create an adapter service that will retrieve an EJB, run a method on that bean, then release it. Because it does not return any Handle objects in its output, the lifecycle of the EJB is entirely contained within the bounds of the adapter service: the EJB is created, a single method is invoked and its output is captured. For this reason, a FetchInvokeEJB 3.0 service is not particularly useful with stateful session beans.

At design time, a FetchInvokeEJB 3.0 service requires the JNDI lookup name of the EJB to fetch and the identity the bean method to invoke. As with the other service templates, you may override the default names of any parameter used in remote method. See "Configuring FetchInvokeEJB 3.0 Services" on page 91 for more information about the parameters.

In a FetchInvokeEJB 3.0 service's input signature, the document names used to contain EJB method arguments are pre-configured and localized. The input document name of the bean method is EJBObject_Args. The documents appear in the input signature only if the corresponding methods have any arguments.

Run-Time Processing for a FetchInvokeEJB 3.0 Service

At run time, a FetchInvokeEJB 3.0 service instance requires any parameter values the bean methods require for inputs. Its output is the output of the remote bean method itself, if any. The adapter service wraps the bean's output in an array in case multiple objects are returned.


webMethods Integration Server
webMethods Integration Server Client
1
Flow Service/ Java Service
2
7
Pipeline
FetchInvokeEJB 3.0 Service
3
8
RemoveEJB Service
Connection Pool
4
5
6
Application Server
JNDI
EJB Container
Target EJB

Step	Description
1	An Integration Server client runs a flow service or Java service on Integration Server.
2	The flow or Java service invokes a FetchInvokeEJB 3.0 adapter service on Integration Server. You configured the adapter service earlier using Designer.
3	The FetchInvokeEJB 3.0 adapter service gets a connection from the service's connection pool. You created and enabled the adapter connection earlier using Integration Server Administrator.
4	Through its connection, the FetchInvokeEJB 3.0 adapter service accesses the application server's JNDI to look up the EJBRemote interface of the EJB for which the service was configured.
5	The FetchInvokeEJB 3.0 adapter service fetches the configured EJBObjects associated with the JNDI Name configure on the application server. If the operation is successful, the adapter service returns an array of javax.ejb.Handle instances that represent the remote EJBs. If the operation is unsuccessful, the adapter service returns an AdapterException or AdapterConnectionException. For more information about how the adapter handles exceptions, see Adapter Logging and Exception Handling.
6	Using the handles obtained in the previous step, the FetchInvokeEJB 3.0 adapter service invokes the configured EJBObject method on the application server. If the method takes

Step	Description
	any parameters, the values are extracted from the pipeline and passed to the remote method. If the operation is successful, the adapter service returns the method's output, if any. If the operation is unsuccessful, the adapter service returns an AdapterException or AdapterConnectionException. For more information about how the adapter handles exceptions, see Adapter Logging and Exception Handling.
7	The FetchInvokeEJB 3.0 adapter service puts the output of the remote method invocation and the status on the pipeline.
8	The FetchInvokeEJB 3.0 adapter service has completed its processing for session EJBs, it calls the EJB-standard remove() method on each EJB object obtained in step 5.

Using Adapter for Enterprise Javabeans Services in a Flow

After configuring adapter services you can use them as either standalone services or you can construct flows that are built around these services to implement a business process. You create and edit flow services using Designer.

For the most part, Designer's Flow Service Editor provides all the tools needed to do this. However, there may be situations where you must provide additional "translation service objects" in the flow for EJBs that return or expect non-standard Java objects.

For example, you have an EJB that exposes a remote method to create an account, the createAccount method. This method takes as its input parameter a third-party com.foo.bar.Account object. Designer will interpret and present the standard classes provided in Java (for example, java.lang.String, java.lang.Double, long), but it cannot interpret com.foo.bar.Account and, therefore, cannot present this object as anything other than java.lang.Object.

This is acceptable if you can generate a suitable instance of com.foo.bar.Account by calling some other service in your flow prior to invoking createAccount on the EJB. However, if no such service is available, you need to create one. To do so, create a simple standalone translation service that exposes an input signature containing all parameters needed to create a com.foo.bar.Account object. At run time, the user enters values for these parameters. The translation service would then use these values to create an instance of com.foo.bar.Account, which it would then place on its outbound pipeline. In the flow you map this object to the input of the adapter service that invokes createAccount.

For more information about creating flow services that use Adapter for Enterprise Javabeans services, see "Creating Flows for Adapter for Enterprise Javabeans Services" on page 147.

Using Adapter Services

Note:

For 3.0 adapter services, you need to specify the jar file information or the JNDI name of the 3.0 EJBs in the config.xml file located at *Integration Server_directory* \instances*instance_name*\packages\WmEJBAdapter\config folder. For more details on configuring the config.xml file, see "Configuring Adapter for Enterprise Javabeans for Adapter Services" on page 27.

The following table lists the tasks required to use adapter services:

1. Create and enable an adapter connection using Integration Server Administrator. See "Adapter Connections" on page 67 for details.

2. Select the appropriate adapter service template and configure the necessary adapter services using Designer.

   Depending on the type of adapter service, you specify the:

   - Adapter connection
   - EJB class
   - EJB create method and/or bean method
   - Method parameter names, if any (optional)

     See "Adapter Services" on page 79 for more information about configuring each of the adapter services.

3. If you plan to use an Integration Server flow service or Java service to invoke the adapter service, design the flow service to use Adapter for Enterprise Javabeans services you configure using Designer. For information about using Designer, see the *webMethods Service Development Help* for your release.

   For information about using the adapter services in a flow, see "Creating Flows for Adapter for Enterprise Javabeans Services" on page 147. To see how to use the services in different operations, see "Scenarios" on page 129.

4. Manage the adapter services using Designer and Integration Server Administrator. See "Managing the Adapter Package" on page 58 and Adapter Logging and Exception Handling for details.

Using Version Control Systems to Manage Adapter Elements

The adapter supports the Version Control System (VCS) Integration feature provided by Designer. When you enable the feature in Integration Server, you can check adapter packages or elements into and out of your version control system from Designer. For more information about the VCS Integration feature, see the *Configuring the VCS Integration Feature*.

Beginning with Integration Server 8.2 SP3, the adapter supports the local service development feature in Designer. This feature extends the functionality of the VCS Integration feature to check package elements and their supporting files into and out of a VCS directly from Designer. For more information about local service development and how it compares to the VCS Integration feature, see the *webMethods Service Development Help*.

Optimize Infrastructure Data Collector Support for the Adapter

Optimize Infrastructure Data Collector monitors the system and operational data associated with webMethods runtime components such as Integration Server, Broker Servers, Brokers, and adapters, and reports the status of these components on Optimize for Infrastructure or other external tools. When you start monitoring Integration Server, Infrastructure Data Collector automatically starts monitoring all ART-based adapters that are installed on Integration Server.

For information about monitored key performance indicators (KPIs) collected for the monitored adapter components, see the *Administering webMethods Optimize* for your release.

Viewing the Adapter's Update Level

The list of updates that have been applied to Adapter for Enterprise Javabeans can be viewed in the **Updates** field on the adapter's About page in Integration Server Administrator.

Controlling Pagination

You can control the number of items that are displayed on the adapter Connections screen. By default, 10 items are displayed per page. Click **Next** and **Previous** to move through the pages, or click a page number to go directly to a page.

To change the number of items displayed per page, set the watt.art.page.size property and specify a different number of items.

To set the number of items per page

1. From Integration Server Administrator, click **Settings > Extended**.

2. Click **Edit Extended Settings**. In the Extended Settings editor, add or update the watt.art.page.size property to specify the preferred number of items to display per page. For example, to display 50 items per page, specify:

   ```
   watt.art.page.size=50
   ```

3. Click **Save Changes**. The property appears in the Extended Settings list.

   For more information about working with extended configuration settings, see the *webMethods Integration Server Administrator's Guide* for your release.

2 Installing, Upgrading, and Uninstalling

Overview

This chapter explains how to install, upgrade, and uninstall webMethods Adapter for Enterprise JavaBeans 6.5 SP3. The instructions use the Software AG Installer and the Software AG Uninstaller wizards. For complete information about the wizards or other installation methods, or to install other webMethods products, see the *Installing webMethods Products On Premises* for your release.

Requirements

For a list of the operating systems, third-party products, and webMethods products supported by the adapter, see the *webMethods Adapters System Requirements* .

Adapter for Enterprise Javabeans 6.5 SP3 has no hardware requirements beyond those of the host Integration Server.

The Integration Server Home Directory

Beginning with Integration Server 9.6, you can create and run multiple Integration Server instances under a single installation directory. Each Integration Server instance has a home directory under *Integration Server_directory* `\instances\`*`instance_name`* that contains the packages, configuration files, log files, and updates for the instance.

For more information about running multiple Integration Server instances, see the *webMethods Integration Server Administrator's Guide* for your release.

If you are using Integration Server 9.5 and lower, the Integration Server home directory is *Integration Server_directory* . For example, on Integration Server 9.5 the adapter package is installed in the *Integration Server_directory* `\packages` directory.

This guide uses the packages_directory as the home directory in Integration Server classpaths. For Integration Server 9.6 and above, the packages_directory is *Integration Server_directory* `\instances\`*`instance_name`*`\packages` directory. For Integration Server 9.5 and lower, the packages_directory is *Integration Server_directory* `\packages` directory.

Installing Adapter for Enterprise Javabeans 6.5 SP3

If you are installing the adapter in a clustered environment, you must install it on each Integration Server in the cluster, and each installation must be identical. For more information about working with the adapter in a clustered environment, see "Using Adapter for Enterprise Javabeans in a Clustered Environment" on page 62.

To install Adapter for Enterprise Javabeans 6.5 SP3

1. Download Software AG Installer from the Empower Product Support Web site.

2. If you are installing the adapter on an existing Integration Server, shut down the Integration Server.

3. Start the Installer wizard.

4. Choose the webMethods release that includes the Integration Server on which to install the adapter.

5. Specify the installation directory as follows:

   - If you are installing on an existing Integration Server, specify the webMethods installation directory that contains the host Integration Server.

   - If you are installing both the host Integration Server and the adapter, specify the installation directory to use.

6. In the product selection list, select **Adapters > webMethods Adapter for Enterprise JavaBeans 6.5 SP3**.

   If you are using Integration Server 9.6 and above, you can choose to install the package in the default instance. In this case, Software AG Installer installs the adapter in both locations, *Integration Server_directory* \packages and the default instance packages directory located in *Integration Server_directory* \instances\default\packages.

7. To download the documentation for the adapter, go to Software AG Documentation website.

8. After installation is complete, close the Installer.

9. Go to "Configure Integration Server to Work with the Application Server" on page 47 and perform those steps.

10. Start Integration Server.

Installing a Sample EJB Application

You can also download sample EJB source code from the Software AG TECHcommunity website. The code demonstrates the procedure for invoking services running on Integration Server from within an EJB. For more information about the sample EJB application, see "Sample EJB Application" on page 100.

Configure Integration Server to Work with the Application Server

After you finish with Installer, you need to perform several manual steps to set up webMethods Integration Server to work with your application server. This section describes some instances of class loading conflicts you should be aware of, and includes the configuration steps necessary for each application server and Integration Server combination.

Class Loading Conflicts

When possible, Adapter for Enterprise Javabeans takes full advantage of Integration Server's per-package class loading scheme. This feature enables you to install application server vendor jar files and deployed EJB jar files under `WmEJBAdapter\code\jars`. Installing the files to this location limits the visibility of the classes and resources in those jars to Adapter for Enterprise Javabeans and its dependents, thus reducing the potential for class loading collisions with other packages installed on Integration Server.

For some application servers, installing the files under `WmEJBAdapter\code\jars` may not be possible due to their own internal class loading requirements. In these cases, the third-party jars files must be installed on either the system classpath or the server's classpath.

If you put jar files in the `lib\system` directory on the Integration Server on which Adapter for Enterprise Javabeans is installed, that Integration Server cannot be managed by Optimize Infrastructure Data Collector due to class loading conflicts. For more information about Infrastructure Data Collector support in the adapter, see "Optimize Infrastructure Data Collector Support for the Adapter" on page 43.

Configuring WebLogic Application Servers

Before configuring the application server, you may want to see "Adapter for Enterprise Javabeans Support for Multiple Application Servers" on page 55.

Note:
You must copy all the jar files from the application server directory to the Integration Server directory as mentioned in the configuration steps of each version of the application server. In the absence of these jar files, the Adapter for Enterprise Javabeans link is not displayed on the Integration Server Administrator screen. Also, Adapter for Enterprise Javabeans may not start properly, may throw a class not found error, or may not enable some features without throwing an explicit error.

Important:
Do not delete the WmEJBConfigUtil.jar from the *Integration Server_directory* \ `instances\`*`instance_name`*`\packages\WmEJBAdapter\code\jars\static` directory. If you delete this jar file, Adapter for Enterprise Javabeans will not start.

WebLogic Server 10.3

To set up Integration Server with WebLogic Server 10.3

1. Copy the following jar files from the *`WebLogic_directory`*`\modules` directory to the *Integration Server_directory* `\instances\`*`instance_name`*`\packages\WmEJBAdapter\code\jars\static` directory on Integration Server:

   - com.bea.core.management.core_2.3.0.0.jar
   - com.bea.core.repackaged.asm_3.0.jar

- com.bea.core.store_1.4.0.0.jar
- com.bea.core.timers_1.4.0.0.jar
- com.bea.core.transaction_2.5.0.0.jar
- com.bea.core.utils.classloaders_1.4.0.0.jar
- com.bea.core.utils.full_1.4.0.0.jar
- com.bea.core.utils.wrapper_1.3.0.0.jar
- com.bea.core.weblogic.lifecycle_1.1.0.0.jar
- com.bea.core.weblogic.rmi.client_1.4.0.0.jar
- com.bea.core.weblogic.security.digest_1.0.0.0.jar
- com.bea.core.weblogic.security.identity_1.1.0.0.jar
- com.bea.core.weblogic.security.wls_1.0.0.0_5-0-2-0.jar
- com.bea.core.weblogic.security_1.0.0.0_5-0-2-0.jar
- com.bea.core.weblogic.socket.api_1.0.0.0.jar
- com.bea.core.weblogic.workmanager_1.4.0.0.jar
- javax.jdo_2.0.1.jar
- javax.persistence_1.0.0.0_1-0.jar
- org.apache.openjpa_2.2.0.0_1-1-0.jar
- com.bea.core.descriptor_1.4.0.0.jar
- com.bea.core.kodo_1.0.0.0_4-2-0.jar
- com.bea.core.logging_1.4.0.0.jar

2. Copy the following jar files from the `*WebLogic_directory*\server\lib` directory to the *Integration Server_directory* `\instances\*instance_name*\packages\WmEJBAdapter\code\jars\static` directory on Integration Server:

   - weblogic.jar
   - wlclient.jar
   - wls-api.jar

WebLogic Server 12c

To set up Integration Server with WebLogic Server 12c

1. In a command prompt window, navigate to the directory location, `WebLogic_directory\server\lib`.

2. Create a wlfullclient.jar file by typing the command, `java -jar wljarbuilder.jar`.

3. Copy the wlfullclient.jar file from the `WebLogic_directory\server\lib` directory to the *Integration Server_directory* `\instances\instance_name\packages\WmEJBAdapter\code\jars\static` directory on Integration Server.

Configuring WebSphere Application Servers

Use the set of instructions that correspond to the version of Integration Server and WebSphere Application Server you are using.

Before configuring the application server, you may want to see "Adapter for Enterprise Javabeans Support for Multiple Application Servers" on page 55.

Note:
You must copy all the jar files from the application server directory to the Integration Server directory as mentioned in the configuration steps of each version of the application server. In the absence of these jar files, the Adapter for Enterprise Javabeans link is not displayed on the IS Administrator screen. Also, Adapter for Enterprise Javabeans may not start properly, may throw a class not found error, or may not enable some features without throwing an explicit error.

Important:
Do not delete the WmEJBConfigUtil.jar from the *Integration Server_directory* \ `instances\instance_name\packages\WmEJBAdapter\code\jars\static` directory. If you delete this jar file, Adapter for Enterprise Javabeans will not start.

WebSphere Application Server

The procedure for setting up Integration Server with WebSphere Server is based on the JVM you are using.

WebSphere Application Server 7.0 with Oracle JVM 1.6

To set up Integration Server with WebSphere Server 7.0 when using Oracle JVM 1.6

1. Copy the orb.properties file from the `WebSphere_directory\jre\lib` directory into the *Integration Server_directory* \jvm\win160\jre\lib directory.

2. Copy the following jar files from the `WebSphere_directory\plugins` directory into the *Integration Server_directory* `\instances\instance_name\packages\WmEJBAdapter\code\jars\static` directory:

   - com.ibm.ws.jdt.core.jar

- com.ibm.ws.runtime.jar
- org.eclipse.jdt.core.jar
- org.eclipse.osgi_.jar
- com.ibm.ws.ejbportable.jar
- com.ibm.ws.emf.jar

3. Copy the following jar files from the `*WebSphere_directory*\java\jre\lib` directory into the *Integration Server_directory* `\instances\*instance_name*\packages\WmEJBAdapter\code\jars\static` directory:

    - ibmcfw.jar
    - ibmorb.jar
    - ibmorbapi.jar
    - iwsorbutil.jar

4. Copy the following jar files from the `*WebSphere_directory*\runtimes` directory into the *Integration Server_directory* `\instances\*instance_name*\packages\WmEJBAdapter\code\jars\static` directory:

    - com.ibm.ws.admin.client_7.0.0.jar
    - com.ibm.ws.ejb.thinclient_7.0.0.jar

WebSphere Application Server 8.5 with Oracle JVM 1.7

To set up Integration Server with WebSphere Server 8.5 when using Oracle JVM 1.7

1. Copy the orb.properties file from the `*WebSphere_directory*\jre\lib` directory into the *Integration Server_directory* `\jvm\win170\jre\lib` directory.

2. Copy the following jar files from the `*WebSphere_directory*\plugins` directory into the *Integration Server_directory* `\instances\*instance_name*\packages\WmEJBAdapter\code\jars\static` directory:

    - com.ibm.ws.jdt.core.jar
    - com.ibm.ws.runtime.jar
    - org.eclipse.jdt.core.jar
    - org.eclipse.osgi_.jar
    - com.ibm.ws.emf.jar

3. Copy the following jar files from the `WebSphere_directory\java\jre\lib` directory into the *Integration Server_directory* `\instances\instance_name\packages\WmEJBAdapter\code\jars\static` directory:

   - ibmcfw.jar
   - ibmorb.jar
   - ibmorbapi.jar
   - iwsorbutil.jar

4. Copy the following jar files from the `WebSphere_directory\runtimes` directory into the *Integration Server_directory* `\instances\instance_name\packages\WmEJBAdapter\code\jars\static` directory:

   - com.ibm.ws.admin.client_8.5.0.jar
   - com.ibm.ws.ejb.thinclient_8.5.0.jar

WebSphere Application Server 7.0 with IBM JVM 1.6

To set up Integration Server with WebSphere Server 7.0 when using IBM JVM 1.6

1. Install IBM JVM 1.6 at `\installation_directory\jvm` directory.

2. Copy the following jar files from the `WebSphere_directory\plugins` directory into the *Integration Server_directory* `\instances\instance_name\packages\WmEJBAdapter\code\jars\static` directory:

   - com.ibm.ws.jdt.core.jar
   - com.ibm.ws.runtime.jar
   - org.eclipse.jdt.core.jar
   - org.eclipse.osgi_.jar
   - com.ibm.ws.ejbportable.jar
   - com.ibm.ws.emf.jar

3. Copy the following jar files from the `WebSphere_directory\runtimes` directory into the *Integration Server_directory* `\instances\instance_name\packages\WmEJBAdapter\code\jars\static` directory:

   - com.ibm.ws.admin.client_7.0.0.jar
   - com.ibm.ws.ejb.thinclient_7.0.0.jar

4. Copy the EJB deployment jar files for any EJBs that will be used in the adapter to the *Integration Server_directory* `\instances\`*`instance_name`*`\packages\WmEJBAdapter\code\jars` directory on Integration Server.

5. To set the JAVA_DIR property to the installed JVM 1.6, open the *Integration Server_directory* `\server.bat` file and set the following property to:

   `SET JAVA_DIR=`*installation_directory*`\jvm\`*IBM JDK 1.6*`\jre`

Configuring JBoss Application Servers

Important:
j2ee.jar must be specified before implfactory.properties in the classpath.

Before configuring the application server, you may want to see "Adapter for Enterprise Javabeans Support for Multiple Application Servers" on page 55.

Note:
You must copy all the jar files from the application server directory to the Integration Server directory as mentioned in the configuration steps of each version of the application server. In the absence of these jar files, the Adapter for Enterprise Javabeans link is not displayed on the IS Administrator screen. Also, Adapter for Enterprise Javabeans may not start properly, may throw a class not found error, or may not enable some features without throwing an explicit error.

Important:
Do not delete the WmEJBConfigUtil.jar from the *Integration Server_directory* \instances\instance_name\packages\WmEJBAdapter\code\jars\static directory. If you delete this jar file, Adapter for Enterprise Javabeans will not start.

JBoss Application Server 5.1.0

To set up Integration Server with JBoss Server 5.1.0

1. Copy the following jar files from the JBoss Server directory (typically *`JBoss_directory`*`\client`) to the *Integration Server_directory* `\instances\`*`instance_name`*`\packages\WmEJBAdapter\code\jars\static` directory on Integration Server:

   - jboss-ejb3-common-client.jar
   - jboss-ejb3-core-client.jar
   - jboss-ejb3-ext-api.jar
   - jboss-ejb3-proxy-clustered-client.jar
   - jboss-ejb3-proxy-impl-client.jar

- jboss-ejb3-proxy-spi-client.jar
- jboss-ejb3-security-client.jar
- jboss-integration.jar
- jboss-javaee.jar
- jboss-managed.jar
- jboss-mdr.jar
- jboss-messaging.jar
- jboss-remoting.jar
- jboss-remoting-aspects.jar
- jboss-security-aspects.jar
- jboss-security-spi.jar
- jboss-serialization.jar
- jbosssx.jar
- jboss-system-jmx-client.jar
- jmx-adaptor-plugin.jar
- jmx-invoker-adaptor-client.jar
- jnp-client.jar
- trove.jar
- applet.jar
- concurrent.jar
- ejb3-persistence.jar
- javassist.jar
- jboss-aop.jar
- jboss-aspect-jdk50-client.jar
- jboss-client.jar
- jboss-common-core.jar

2. Copy the EJB deployment jar files for any EJBs that will be used in the adapter to the *Integration Server_directory* `\instances\`*`instance_name`*`\packages\WmEJBAdapter\code\jars` directory on Integration Server.

Adapter for Enterprise Javabeans Support for Multiple Application Servers

The details about Adapter for Enterprise Javabeans support for multiple application servers are as follows:

- Adapter for Enterprise Javabeans supports multiple instances of the same version of the application server.

  Note:
  For multiple instances of the same version of WebLogic application server, you must use JAAS authentication. For more information about using JAAS authentication for WebLogic servers, see "JAAS Authentication" on page 164.

- Adapter for Enterprise Javabeans supports two different versions of the same application server provided the jar files of the latest version are backward compatible and support the connections of the earlier version, but there can always be a risk of conflict.

- Adapter for Enterprise Javabeans does not support two different application servers at a time. This is because Adapter for Enterprise Javabeans requires you to copy jar files from the application server directory to the Integration Server directory. If you copy the jar files from different application servers to the Integration Server directory, there can be a conflict of jar files of one application server with those of other application servers.

Upgrading to Adapter for Enterprise Javabeans 6.5 SP3

To upgrade to Adapter for Enterprise Javabeans 6.5 SP3

1. Back up the WmEJBAdapter package and any existing custom adapter packages.

2. Uninstall the older version of the adapter. For information about how to uninstall Adapter for Enterprise Javabeans, see the installation guide for the older adapter version.

3. Delete the WmEJBAdapter package from the *Integration Server_directory* `\instances\`*`instance_name`*`\packages` directory.

   You should not delete any other packages, such as packages that contain connections and services, even when the packages have a dependency on the WmEJBAdapter package.

4. Install Adapter for Enterprise Javabeans as described in "Installing Adapter for Enterprise Javabeans 6.5 SP3" on page 46.

5. Copy the application server client libraries from the old adapter installation to *Integration Server_directory* `\instances\`*`instance_name`*`\packages\WmEJBAdapter\code\jars` and *Integration Server_directory* `\instances\`*`instance_name`*`\packages\WmEJBAdapter\code\jars\static`.

6. Restart Integration Server.

Uninstalling Adapter for Enterprise Javabeans 6.5 SP3

To uninstall Adapter for Enterprise Javabeans 6.5 SP3

1. Shut down the host Integration Server. You do not need to shut down any other webMethods products or applications that are running on your machine.

2. If you want to keep the Adapter for Enterprise Javabeans sample files, copy the *Integration Server_directory* `\packages\WmEJBAdapter\templates` directory to another location on your system.

3. Start Software AG Uninstaller, selecting the webMethods installation directory that contains the host Integration Server. In the product selection list, select **Adapters > webMethods Adapter for Enterprise JavaBeans 6.5 SP3**. You can also choose to uninstall documentation.

4. Restart the host Integration Server.

5. Uninstaller removes all Adapter for Enterprise Javabeans 6.5 SP3-related files that were installed. However, Uninstaller does not delete files created after you installed the adapter (for example, user-created or configuration files), nor does it delete the adapter directory structure. You can go to the *Integration Server_directory* `\packages` directory and *Integration Server_directory* `\instances\default\packages` directory. Delete the WmEJBAdapter directory.

3 Adapter Package Management

Overview

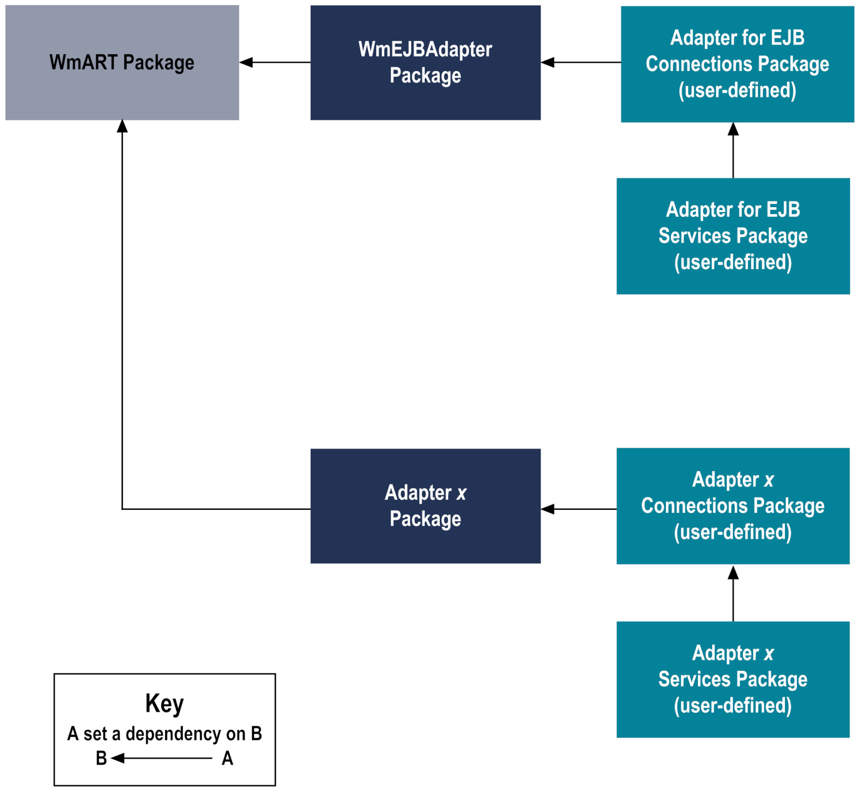
The following sections describe how to set up and manage your Adapter for Enterprise Javabeans packages, set up Access Control Lists (ACL), and use the adapter in a clustered environment.

Managing the Adapter Package

Adapter for Enterprise Javabeans is provided as a package called WmEJBAdapter. You manage the WmEJBAdapter package as you would manage any package on Integration Server.

When you create connections and adapter services, define them in user-defined packages rather than in the WmEJBAdapter package. Doing so will allow you to manage the package more easily.

As you create user-defined packages in which to store connections and adapter services, use the package management functionality provided in Designer and set the user-defined packages to have a dependency on the WmEJBAdapter package. That way, when the WmEJBAdapter package loads or reloads, the user-defined packages load automatically. See the following diagram:


WmART Package
WmEJBAdapter Package
Adapter for EJB Connections Package (user-defined)
Adapter for EJB Services Package (user-defined)
Adapter *x* Package
Adapter *x* Connections Package (user-defined)
Adapter *x* Services Package (user-defined)
Key
A set a dependency on B
B ← A

Package management tasks include:

- Setting package dependencies (see "Package Dependency Requirements and Guidelines" on page 59).
- "Enabling Packages" on page 60.
- "Disabling Packages" on page 59.

- "Controlling Group Access" on page 61.

Package Dependency Requirements and Guidelines

This section contains a list of dependency requirements and guidelines for user-defined packages. For instructions for setting package dependencies, see the *webMethods Service Development Help* for your release.

- A user-defined package must have a dependency on its associated adapter package, WmEJBAdapter. (The WmEJBAdapter package has a dependency on the WmART package.)
- Package dependencies ensure that at startup Integration Server automatically loads or reloads all packages in the proper order: the WmART package first, the adapter package next, and the user-defined package(s) last. The WmART package is automatically installed when you install Integration Server. You should not need to manually reload the WmART package.
- If the connections and adapter services of an adapter are defined in different packages, then:
  - A package that contains the connection(s) must have a dependency on the adapter package.
  - Packages that contain adapter services must have a dependency on their associated connection package.
- Keep connections for different adapters in separate packages so that you do not create interdependencies between adapters. If a package contains connections for two different adapters, and you reload one of the adapter packages, the connections for both adapters will reload automatically.
- Integration Server will not allow you to enable a package if it has a dependency on another package that is disabled. That is, before you can enable your package, you must enable all packages on which your package depends. For information about enabling packages, see "Enabling Packages" on page 60.
- Integration Server will allow you to disable a package even if another package that is enabled has a dependency on it. Therefore, you must manually disable any user-defined packages that have a dependency on the adapter package before you disable the adapter package. For information about disabling packages, see "Disabling Packages" on page 59.
- You can name connections and adapter services the same name provided that they are in different folders and packages.

Disabling Packages

All packages are automatically enabled by default. When you want to temporarily prohibit access to the elements in a package, disable the package. When you disable a package, the server unloads all of its elements from memory. Disabling a package prevents Integration Server from loading that package at startup. A disabled package will remain disabled until you explicitly enable it using Integration Server Administrator.

To disable a package

1. Open Integration Server Administrator if it is not already open.

2. In the **Packages** menu of the navigation area, click **Management**.

3. Click **Yes** in the **Enabled** column for the package that you want to disable. The server issues a prompt to verify that you want to disable the package. Click **OK** to disable the package. When the package is disabled, the server displays **No** in the **Enabled** column.

   A disabled adapter will:

   - Remain disabled until you explicitly enable it using Integration Server Administrator.
   - Not be listed in the Software AG Designer Package Navigator view.

Enabling Packages

A disabled package will remain disabled until you explicitly enable it using Integration Server Administrator.

To enable a package

1. Open Integration Server Administrator if it is not already open.

2. In the **Packages** menu of the navigation area, click **Management**.

3. Click **No** in the **Enabled** column. The server displays a ✓ and **Yes** in the **Enabled** column.

   Note:
   Enabling an adapter package will *not* cause its associated user-defined package(s) to be reloaded. For information about reloading packages, see "Loading, Reloading, and Unloading Packages" on page 60.

   Important:
   Before you manually enable a user-defined package, you must first enable its associated adapter package (WmEJBAdapter). Similarly, if your adapter has multiple user-defined packages, and you want to disable some of them, disable the adapter package first. Otherwise, errors will be issued when you try to access the remaining enabled user-defined packages.

Loading, Reloading, and Unloading Packages

Recall that if user-defined packages are properly configured with a dependency on the adapter package (as described in "Package Dependency Requirements and Guidelines" on page 59), at startup Integration Server automatically loads or reloads all packages in the proper order: the WmART package first, the adapter package next, and the node package(s) last. You should not need to manually reload the WmART package.

Reloading Packages Manually

Reloading a user-defined package will *not* cause its associated adapter package to be reloaded. You can reload adapter packages and user-defined packages from either Integration Server Administrator (by clicking the icon on the Management window) or from Software AG Designer (by right-clicking the package and selecting the **Reload Package** option from the menu).

Unloading Packages

At shutdown, Integration Server unloads packages in the reverse order in which it loaded them: it unloads the node package(s) first, the adapter package next, and the WmART package last (assuming the dependencies are correct).

Setting Package Dependencies

You set package dependencies if a given package needs services in another package to load before it can load. For example, any packages you create for Adapter for Enterprise Javabeans services should identify the webMethods Adapter for Enterprise JavaBeans package (WmEJBAdapter) as a package dependency because they require services in the WmEJBAdapter to load first. Use the following guidelines:

- Set package dependencies from the adapter service package to the package containing the connection if you configure a connection in one package and the adapter services in another package. That is, the package that contains the connection should load before the adapter service package.

  When you set this package dependency, it ensures that if someone disables the connection package and then re-enables it, the adapter services will reload correctly.

- If both the connection and adapter services are in the same package, set this package to have a dependency on the WmEJBAdapter package.

- In general, packages containing connections should have a dependency set to the adapter package itself. That is, the adapter service package should depend on the adapter connection package, which should depend on the adapter package. Similarly, if the adapter services are in the same package as the connections, the only dependency that you need to set is between the adapter connection package and the adapter package.

For more information about setting package dependencies, see the *webMethods Service Development Help* for your release.

Controlling Group Access

To control which development group has access to which adapter services, use access control lists (ACLs). You can use ACLs to prevent one development group from inadvertently updating the work of another group, or to allow or deny access to services that are restricted to one group but not to others.

For general information about assigning and managing ACLs, see the *webMethods Service Development Help* for your release.

Using Adapter for Enterprise Javabeans in a Clustered Environment

Clustering is an advanced feature of the webMethods product suite that substantially extends the reliability, availability, and scalability of webMethods Integration Server. Clustering accomplishes this by providing the infrastructure and tools to deploy multiple Integration Servers as if they were a single virtual server and to deliver applications that leverage that architecture. Because this activity is transparent to the client, clustering makes multiple servers look and behave as one.

For details on Integration Server clustering, see the *webMethods Integration Server Clustering Guide* for your release.

Integration Server 8.2 SP2 and higher supports the caching and clustering functionality provided by Terracotta. Caching and clustering are configured at the Integration Server level and Adapter for Enterprise Javabeans uses the caching mechanism that is enabled on Integration Server.

With clustering, you get the following benefits:

- **Load balancing.** This feature, provided automatically when you set up a clustered environment, allows you to spread the workload over several servers, thus improving performance and scalability.
- **Failover support.** Clustering enables you to avoid a single point of failure. If a server cannot handle a request, or becomes unavailable, the request is automatically redirected to another server in the cluster.

  Note:
  Integration Server clustering redirects HTTP and HTTPS requests, but does not redirect FTP or SMTP requests.
- **Scalability.** You can increase your capacity even further by adding new machines running Integration Server to the cluster.

Configuring the Adapter in a Clustered Environment

When you configure Adapter for Enterprise Javabeans to create adapter services, you must:

- Ensure that each Integration Server in the cluster contains an identical set of packages (see "Replicating Packages to webMethods Integration Servers" on page 63).
- Disable the redirection capability for certain predefined administrative services (see "Disabling the Redirection of Administrative Services" on page 63).

Replicating Packages to webMethods Integration Servers

Every Integration Server in the cluster should contain an identical set of packages that you define using Adapter for Enterprise Javabeans; that is, you should replicate Adapter for Enterprise Javabeans services and the connections they use.

To ensure consistency, we recommend that you create all packages on one server, and replicate them to the other servers. If you allow different servers to contain different services, you might not derive the full benefits of clustering. For example, if a client requests a service that resides in only one server, and that server is unavailable, the request cannot be successfully redirected to another server.

For information about replicating packages, see the chapter on managing packages in the *webMethods Integration Server Administrator's Guide* for your release.

Disabling the Redirection of Administrative Services

A server that cannot handle a client's service request can automatically redirect the request to another server in the cluster. However, Adapter for Enterprise Javabeans uses certain predefined administrative services that you should not allow to be redirected. These services are used internally when you configure the adapter. If you allow these services to be redirected, your configuration specifications might be saved on multiple servers, which is an undesirable result. For example, if you create two Adapter for Enterprise Javabeans services, one might be stored on one server, while the other one might be stored on another server. Remember that all adapter services must reside on all Integration Servers in the cluster.

To disable the redirection of administrative services

1. Shut down Integration Server Administrator. For more information on how to shut down Integration Server Administrator, see the *webMethods Integration Server Administrator's Guide* for your release.

2. Edit the following file:

   Integration Server_directory `\config\redir.cnf`

3. Add the following line to the file:

   ```
   <value name="wm.art">false</value>
   ```

4. Save the file and restart Integration Server.

Clustering Considerations and Requirements

Note:

The following sections assume that you have already configured the Integration Server cluster. For details about webMethods clustering, see the *webMethods Integration Server Clustering Guide* for your release.

The following considerations and requirements apply to Adapter for Enterprise Javabeans in a clustered environment.

Requirements for Each Integration Server in a Cluster

The following table describes the requirements of each Integration Server in a given cluster:

All Integration Servers in a given cluster must have identical...	For Example...
Integration Server versions	All Integration Servers in the cluster must be the same version, with the same service packs and fixes (updates) applied.
Adapter packages	All adapter packages on one Integration Server should be replicated to all other Integration Servers in the cluster.
Adapter versions	All Adapter for Enterprise Javabeanss must be the same version, with the same fixes (updates) applied.
Adapter connections	If you configure a connection to the application server, this connection must appear on all servers in the cluster so that any Integration Server in the cluster can handle a given request identically. If you plan to use connection pools in a clustered environment, see "Considerations When Configuring Connections with Connection Pooling Enabled" on page 65.
Adapter services	If you configure a specific adapter service, this same adapter service must appear on all servers in the cluster so that any Integration Server in the cluster can handle the request identically. If you allow different Integration Servers to contain different services, you might not derive the full benefits of clustering. For example, if a client requests a service that resides on only one server, and that server is unavailable, the request cannot be successfully redirected to another server.

See “Replicating Packages to webMethods Integration Servers” on page 63 for information about replicating adapter packages, connections, and adapter services across multiple Integration Servers in a cluster.

Considerations When Installing Adapter for Enterprise Javabeans Packages

For each Integration Server in the cluster, use the standard Adapter for Enterprise Javabeans installation procedures for each machine, as described in “Installing, Upgrading, and Uninstalling” on page 45.

Considerations When Configuring Connections with Connection Pooling Enabled

When you configure an adapter connection that uses connection pools in a clustered environment, be sure that you do not exceed the total number of connections that can be opened simultaneously for that application server.

For example, if you have a cluster of two Integration Servers with a connection configured to an application server that supports a maximum of 100 connections opened simultaneously, the total number of connections possible at one time must not exceed 100. This means that you cannot configure a connection with an initial pool size of 100 and replicate the connection to both servers, because there could be possibly a total of 200 connections opened simultaneously to this application server.

In another example, consider a connection configured with an initial pool size of 10 and a maximum pool size of 100. If you replicate this connection across a cluster with two Integration Servers, it is possible for the connection pool size on both servers to exceed the maximum number of connections that can be open at one time.

For information about configuring connections for Adapter for Enterprise Javabeans, see “Configuring Adapter Connections” on page 69.

For more general information about connection pools, see the *webMethods Integration Server Administrator’s Guide* for your release.

4 Adapter Connections

Overview

This chapter describes how to configure and manage Adapter for Enterprise Javabeans connections. For more information about how adapter connections work, see "Adapter Connections" on page 17.

Note:
You must have webMethods administrator privileges to access Adapter for Enterprise Javabeans's administrative screens. For information about setting user privileges, see the *webMethods Integration Server Administrator's Guide* for your release.

Before Configuring or Managing Adapter Connections

To prepare for creating an Adapter for Enterprise Javabeans connection

1. Ensure your application server is set up to work with Adapter for Enterprise Javabeans. These tasks are application-server specific and are listed below:

   a. Set up the JNDI properties file used by the application server. See "JNDI Properties File" on page 21 for the specific properties and the required values.

   b. Determine the level of security needed to access the JNDI server and how credentials are provided. See "Specifying JNDI Credentials" on page 23 and "Security Considerations" on page 23 for the required properties and values.

   c. Ensure that the JNDI properties files, the classes directory, and the supported connections directory are properly secured and that only privileged users can access them. See "Security Considerations" on page 23 for more information.

   d. Address any other application server configuration issues. This information is provided for each supported application server and can be found in "Application Server Configuration Notes" on page 163.

   e. Ensure that the necessary jar files have been copied to Integration Server:

      - Application-server specific jar files
      - EJB deployment jar files for any EJBs that will be used in the adapter

      Typically this step is performed as part of the Adapter for Enterprise Javabeans installation procedure. For more information about the jar files and configuring the application server to work with Integration Server, see "Installing, Upgrading, and Uninstalling" on page 45.

   f. Determine the number of connections you need to configure, and the types of those connections. This depends on the nature of the specific EJBs you want to interact with, your integration needs, and the level of transaction support necessary for the EJBs. Check that your application server supports the transactions you want to use. This information is provided for each supported application server and can be found in "Application Server Configuration Notes" on page 163.

2. Install webMethods Integration Server and Adapter for Enterprise Javabeans on the same machine. See "Installing, Upgrading, and Uninstalling" on page 45 for details.

3. Make sure you have webMethods administrator privileges so that you can access Adapter for Enterprise Javabeans's administrative screens. For more information about setting user privileges, see the *webMethods Integration Server Administrator's Guide* for your release.

4. Start your Integration Server and Integration Server Administrator, if they are not already running.

5. Using Integration Server Administrator, make sure the WmEJBAdapter package is enabled. See "Enabling Adapter Connections" on page 76 for instructions.

6. Using Software AG Designer, create a user-defined package to contain the connection, if you have not already done so. See "Managing the Adapter Package" on page 58 for details.

7. Create your connections, as described in "Configuring Adapter Connections" on page 69.

Configuring Adapter Connections

When you configure Adapter for Enterprise Javabeans connections, you specify information that the adapter uses to connect to a supported application server's JNDI server.

You configure Adapter for Enterprise Javabeans connections using Integration Server Administrator.

To configure an adapter connection

1. In Integration Server Administrator select **Adapters > Adapter for EJB**.

2. On the Connections screen, click **Configure New Connection**.

3. On the **Connection Types** screen, select one of the following adapter connection types:

Connection Type	Description
EJB Non-transactional Connection	Creates a connection that will not be transacted.
EJB Local Connection	Creates a connection that will be employed in non-distributed, local transactions.
EJB XA Connection	Creates a connection that will be employed in distributed, two-phase commit transactions.

4. For more information about the transaction types, see "Transaction Management of Adapter for Enterprise Javabeans Connections" on page 19.

5. In the **Adapter for EJB** section, provide values for the following parameters:

Parameter	Description/Action
Package	The package in which to create the connection. You must create the package using Designer before you can specify it using this parameter. For general information about creating packages, see the *webMethods Service Development Help* for your release. **Note:** Create the connection in a user-defined package rather than in the adapter's package. See "Adapter Package Management" on page 57 for other important considerations when creating packages for Adapter for Enterprise Javabeans.
Folder Name	The folder in which to create the connection.
Connection Name	The name you want to give the connection. Connection names cannot have spaces or use special characters reserved by Integration Server or Designer. For more information about the use of special characters in package, folder, and element names, see the *webMethods Service Development Help* for your release.

6. In the **Connection Properties** section, provide values for the following parameters:

Parameter	Description/Action
EJB Server Type	Select the name of the application server vendor/version implementation class for the connection. **Note:** WebLogic Server 10.3 supports connections for XA transactions.
XAResource Source	(Used only with connections for XA transactions.) Typically a vendor-specific string that specifies the application server instance you are running against. This parameter is used by the implementation class to obtain the XAResource object from the application server. **Note:**

Parameter	Description/Action
	WebLogic 10.3 support XA transactions. The server instance is one of the names shown under the Servers link on the WebLogic administration screen.
Properties File Name	Enter the full path to a text file on a file system accessible to Integration Server that contains the JNDI-specific Java properties. For information about the properties that need to be specified in this file, see "JNDI Properties File" on page 21.
JNDI Username	Optional. If a username and password is required to access the server's JNDI, enter the user name. Specifying a JNDI username in this parameter overrides the value specified in the standard java.naming.security.principal property if this property is also defined in the JNDI properties file. For more information about security credentials, see "JNDI Properties File" on page 21.
JNDI Password	Optional. If a username and password is required to access the server's JNDI, enter the password. Specifying a JNDI password in this parameter overrides the value specified in the standard java.naming.security.credentials property if this property is also defined in the JNDI properties file. For more information about security credentials, see "JNDI Properties File" on page 21.
Retype JNDI Password	Required if you entered a JNDI password. Verifies the password entered.
EJB Caching Level	Defines the level of EJB caching the connection uses when configuring adapter services. For more information and EJB caching levels, see "EJB Information Caching" on page 24. ■ `None`. No caching occurs. When configuring the adapter service, the connection will retrieve EJB details directly from the JNDI. This is the default. ■ `Weak`. Information in the cache is retained as long as there are outstanding weak references to the cache. ■ `Soft`.Information in the cache is retained until the JVM would be forced to throw an OutOfMemoryError in response to a memory allocation request. At this point, it frees the memory in the cache. ■ `Hard`. Data is retrieved from the JNDI once and cached until no more connections remain enabled for the connection factory. Subsequent requests for EJB detail are retrieved from the cache.

7. In the **Connection Management Properties** section, provide values for the following parameters:

Parameter	Description/Action
Enable Connection Pooling	Enables the adapter to use connection pooling. Default: true. See "Connection Pools" on page 18 for more information about connection pooling in the adapter. If you plan to enable connection pooling in a clustered environment, consider the connection pool size. For details, see "Considerations When Configuring Connections with Connection Pooling Enabled" on page 65.
Minimum Pool Size	The minimum number of connection objects that remain in the connection pool at all times. When the adapter creates the pool, it creates this number of connections. Default: 1.
Maximum Pool Size	The maximum number of connection objects that can exist in the connection pool. When the connection pool has reached its maximum number of connections, the adapter will reuse any inactive connections in the pool or, if all connections are active, it will wait for a connection to become available. Default: 10.
Pool Increment Size	If connection pooling is enabled, this parameter specifies the number of connections by which the pool will be incremented if connections are needed, up to the maximum pool size. Default: 1.
Block Timeout	If connection pooling is enabled, this parameter specifies the number of milliseconds that Integration Server will wait to obtain a connection with the database before it times out and returns an error. Default: 1000.
Expire Timeout	If connection pooling is enabled, this parameter specifies the number of milliseconds that an inactive connection can remain in the pool before it is closed and removed from the pool. For example, to specify 10 seconds, specify 10000. Enter 0 to specify no timeout. Default: 1000. **Note:** The adapter will never violate the **Minimum Pool Size** parameter. These connections remain in the pool regardless of how long they are inactive.
Startup Retry Count	If connection pooling is enabled, this parameter specifies the number of times that the system should attempt to initialize the connection pool at startup if the initial attempt fails, before issuing an AdapterConnectionException. Default: 0.

Parameter	Description/Action
Startup Backoff Timeout	If connection pooling is enabled, this parameter specifies the number of seconds to wait between each attempt to initialize the connection pool. Default: 10.

8. Click **Save Connection**.

   The connection you created appears on the adapter's Connections screen and in Designer's Package Navigator.

   Be sure to enable the connection before you create adapter services that use it. See "Enabling Adapter Connections" on page 76 for instructions.

Dynamically Changing a Service's Connection at Runtime

You can run an adapter service using a connection other than the default connection that was associated with the service when the service was created. To override the default, you must code your flow to pass a value through the pipeline into a service's $connectionName field.

For example, you have a flow whose primary purpose is to create an entity EJB on the production application server. However, you want the flow to have the capability to create the entity on the test server, with the decision of which application server to update to be made programmatically at runtime. The output signature of the flow's first service contains a field called Target. The flow could branch based on the value in Target:

- If Target contains the value Production, the second service in the flow, a CreateEJB 2.1 adapter service, would ignore $connectionName - thus using its default connection to create the EJB on the production server.
- However, if Target contains the value Test, the second service in the flow would use the value in the $connectionName from the pipeline and connect to (and then update) the test server.

Keep in mind these restrictions when using dynamic connections:

- The EJB invoked by the CreateEJB 2.1 or FetchEJB 3.0 adapter service must be deployed on both application servers
- The default and override connections must be of the same type: EJB Non-Transactional, EJB Local, or EJB XA Connection
- The $connectionName field is present only in services created with Designer

For more information, see "Changing the Connection Associated with an Adapter Service at Run Time" on page 20.

Viewing Adapter Connection Parameters from Integration Server Administrator

You can view a connection's parameters from Integration Server Administrator.

To view the parameters for an adapter connection from Integration Server Administrator

1. Start Integration Server Administrator if it is not already running.

2. Make sure the connection is enabled. See "Enabling Adapter Connections" on page 76 for details.

3. In the **Adapters** menu in the navigation area of Integration Server Administrator, click **Adapter for EJB**.

   You can sort and filter the list of connections that appears on the Connections screen.

   - To sort information on the Connections screen, click the **Up** and **Down** arrows.

   - To filter the list of connections:

     1. On the Connections screen, click **Filter Connections**.

     2. Type the criterion by which you want to filter into the **Filter criteria** box. Filtering is based on the node name, not the connection alias. To locate all connections containing specific alphanumeric characters, use asterisks (*) as wildcards. For example, if you want to display all connections containing the string "abc", type `*abc*` in the **Filter criteria** box.

     3. Click **Submit**. The Connections screen displays the connections that match the filter criteria.

     4. To re-display all connections, click **Show All Connections**.

   - The Connections screen appears, listing all the current connections. You can control the number of connections that are displayed on this screen. For more information, see "Controlling Pagination" on page 43.

   - On the Connections screen, click the icon for the connection you want to see.

     The View Connection screen displays the parameters for the connection. For descriptions of the connection parameters, see the table of parameters in "Configuring Adapter Connections" on page 69.

4. Click **Return to Adapter for EJB Connections**to return to the Connections screen.

Viewing Adapter Connection Parameters from Designer

You can view a connection's parameters from Designer.

To view the parameters for a connection using Designer

1. Start Designer if it is not already running.

2. From the Designer Package Navigator view, open the package and folder in which the connection is located.

3. Double-click the connection you want to view.

4. The parameters for the connection appear on the **Connection Information** tab. For descriptions of the connection parameters, see "Configuring Adapter Connections" on page 69.

Editing Adapter Connections

If you want to redefine parameters that a connection uses when connecting to an application server, you can update a connection's parameters using Integration Server Administrator.

To edit an adapter connection

1. Start Integration Server Administrator if it is not already running.

2. Make sure the connection is disabled. See "Disabling Adapter Connections" on page 77 for instructions.

3. In the **Adapters** menu in the navigation area of Integration Server Administrator, click **Adapter for EJB**.

4. On the Connections screen, click the 📝 icon for the connection you want to edit.

   The Edit Connection screen displays the current parameters for the connection. Update the connection's parameters by typing or selecting the values you want to specify.

   For descriptions of the connection parameters, see the table of parameters in "Configuring Adapter Connections" on page 69.

5. Click **Save Changes** to save the connection and return to the Connections screen.

6. Enable the connection when you are ready to use it. See "Enabling Adapter Connections" on page 76 for instructions.

Copying Adapter Connections

You can copy an existing Adapter for Enterprise Javabeans connection to create a new connection with the same or similar connection properties without retyping all properties for the new connection.

To copy an adapter connection

1. Start Integration Server Administrator if it is not already running.

2. Make sure the connection is enabled. See "Enabling Adapter Connections" on page 76 for details.

3. In the **Adapters** menu in the navigation area of Integration Server Administrator, click **Adapter for EJB**.

4. On the Connections screen, click the 📝 icon for the connection you want to copy.

   The Copy Connection screen displays the current parameters for the connection you want to copy. Name the new connection and edit any connection parameters as needed by typing or selecting the values you want to specify.

   For descriptions of the connection parameters, see the table of parameters in "Configuring Adapter Connections" on page 69.

5. Click **Save Connection Copy** to save the connection and return to the Connections screen.

Deleting Adapter Connections

If you no longer want to use an Adapter for Enterprise Javabeans connection, use the following instructions to delete the connection.

If you delete an Adapter for Enterprise Javabeans connection, the adapter services that are defined to use the connection will no longer work. However, because you can change which connection an adapter service uses, if you delete an Adapter for Enterprise Javabeans connection, you can assign a different connection to an adapter service and re-use the service. To do this, you use the built-in Integration Server function setAdapterServiceNodeConnection. For more information, see "Changing the Connection Associated with an Adapter Service at Design Time" on page 20.

To delete an adapter connection

1. Start Integration Server Administrator if it is not already running.

2. Disable the connection. See "Disabling Adapter Connections" on page 77 for details.

3. In the **Adapters** menu in the navigation area of Integration Server Administrator, click **Adapter for EJB**.

4. On the Connections screen, click the ✗ icon for the connection you want to delete.

   Integration Server deletes the adapter connection.

Enabling Adapter Connections

Adapter connections must be enabled before you can create adapter services for those connections.

Note:

When you reload a package that contains enabled connections, the connections will automatically be enabled when the package reloads. If the package contains connections that are disabled, they will remain disabled when the package reloads.

To enable an adapter connection

1. Start Integration Server Administrator if it is not already running.
2. Make sure the WmEJBAdapter package is enabled. See "Enabling Packages" on page 60 for details.
3. In the **Adapters** menu in the navigation area of Integration Server Administrator, click **Adapter for EJB**.
4. On the Connections screen, click **No** in the **Enabled** column for the connection you want to enable.

   Integration Server Administrator enables the adapter connection and displays ✓ and **Yes** in the **Enabled** column.

Disabling Adapter Connections

Adapter connections must be disabled before you can edit or delete the connections.

To disable an adapter connection

1. Start Integration Server Administrator if it is not already running.
2. Make sure the WmEJBAdapter package is enabled. See "Enabling Packages" on page 60 for details.
3. In the **Adapters** menu in the navigation area of Integration Server Administrator, click **Adapter for EJB**.
4. On the Connections screen, click **Yes** in the **Enabled** column for the connection you want to disable.

   Integration Server Administrator disables the adapter connection and displays **No** in the **Enabled** column.

5 Adapter Services

Overview

The following sections describe how to configure adapter services that you use to access the business methods exposed on the EJBs.

Note:
You cannot configure Adapter for Enterprise Javabeans services that invoke more than a single bean method.

You can configure the following types of services for use with Adapter for Enterprise Javabeans:

- **CreateEJB 2.1.** See "Configuring CreateEJB 2.1 Services" on page 81.
- **InvokeEJB 2.1.** See "Configuring InvokeEJB 2.1 Services" on page 83.
- **CreateInvokeEJB 2.1.** See "Configuring CreateInvokeEJB 2.1 Services" on page 85.
- **FetchEJB 3.0.** See "Configuring FetchEJB 3.0 Services" on page 88.
- **InvokeEJB 3.0.** See "Configuring InvokeEJB 3.0 Services" on page 89.
- **FetchInvokeEJB 3.0.** See "Configuring FetchInvokeEJB 3.0 Services" on page 91.

  Note:
  The 2.1 services, namely, CreateEJB 2.1, InvokeEJB 2.1, and CreateInvokeEJB 2.1 are used with EJB 2.1 or earlier versions. All the 3.0 services, namely, FetchEJB 3.0, InvokeEJB 3.0, and FetchInvokeEJB 3.0 are used only with EJB 3.0.

For a description of the adapter services, see "Adapter Services" on page 79. For information about using adapter services you create in a flow service, see "Creating Flows for Adapter for Enterprise Javabeans Services" on page 147.

Before Configuring or Managing Adapter Services

To prepare to configure or manage an Adapter for Enterprise Javabeans service

1. Start your Integration Server and Integration Server Administrator, if they are not already running.

2. If your adapter services work with the EJB 3.0 standard, ensure that you have specified the 3.0 EJBs that will be accessed by your 3.0 adapter services in the config.xml file. For information about how to specify this information, see "Configuring Adapter for Enterprise Javabeans for Adapter Services" on page 27.

3. Make sure you have webMethods administrator privileges so that you can access Adapter for Enterprise Javabeans administrative screens. For information about setting user privileges, see the *webMethods Integration Server Administrator's Guide* for your release.

4. Using Integration Server Administrator, make sure the WmEJBAdapter package is enabled. For instructions, see “Enabling Packages” on page 60.

5. Using Integration Server Administrator:

   a. Configure the adapter connection you plan to use with the adapter service. For instructions, see “Configuring Adapter Connections” on page 69.

   b. Make sure the connection you plan to use with the adapter service is enabled. For instructions, see “Enabling Adapter Connections” on page 76.

6. Using Designer, create a user-defined package to contain the service, if you have not already done so. When you configure adapter services, you should always define them in user-defined packages rather than in the WmEJBAdapter package. For more information about managing packages for the adapter, see “Adapter Package Management” on page 57.

Configuring CreateEJB 2.1 Services

A CreateEJB 2.1 adapter service invokes a specific EJB creator or finder method on the EJB's remote home interface. For more information about the template used to create these services, see “CreateEJB 2.1 Adapter Service” on page 30. For more information about adapter services, see .

To configure a CreateEJB 2.1 adapter service

1. Review the steps in “Before Configuring or Managing Adapter Services” on page 80.

2. Start Designer.

   Note:
   Make sure the server with which you want to use Designer is running.

3. Right-click the package in which the service should be contained and select **New > Adapter Service**.

4. Select the parent namespace and type a name for the adapter service. Click **Next**.

5. Select **Adapter for EJB** from the list of available adapter types. Click **Next**.

6. Select the appropriate **Adapter Connection Name** and click **Next**.

7. From the list of available templates, select the **CreateEJB 2.1** template and click **Finish**.

   The service is created and its parameters and controls are displayed in the adapter service editor.

8. In the editor, select the **CreateEJB 2.1** tab and specify the following values:

Parameter	Description/Action
EJB Lookup Name	Lists the JNDI lookup names of all available EJBs on the application server. The name you select here determines the values that will appear in the **EJB Create Method** and **Return Type** parameters. At runtime, the service uses this name to look up the EJBHome object. The EJBs are listed alphabetically. By default, the first EJB in the list is initially selected.
EJB Create Method	Lists the available creator/finder methods for the selected EJB lookup name. The method selected will be executed by the service. The methods exposed by the EJB are listed alphabetically. By default, the first method in this list is initially selected. ■ If the method has parameters, they appear in the parameter list. For each parameter, the default parameter name and the corresponding Java class type are shown in the first two columns of the parameter list. You may override a parameter's default name by entering a new name in the **Override Parameter Name** column. The value that appears in this column for a parameter is the value that appears in the configured method's input signature at runtime. ■ If the method takes no parameters, the parameter list is empty.
Return Type	Lists the remote EJB method's Java return type for the selected EJB lookup name. This value is read-only.

Note:
The EJB lookup name and EJB create method are displayed by default. If the lookup name and method have input parameters, the parameters are not displayed in the service's input signature until you reload the adapter values. Save the adapter service if you want to use the default values. If you change the lookup name or the create method, the values are automatically refreshed and the parameters appear in the adapter service editor. For more information about reloading adapter values, see “Reloading Adapter Values” on page 97.

9. You can select the **Adapter Settings** tab at any time to confirm adapter properties such as adapter type, connection name, and service template, as needed.

10. The **Input/Output** tab lists the input and output parameters, if any, for the method. If the create method has parameters, their names and types appear under the EJBHome_Args document. If this service is later used in a flow, its input and output signatures will be visible in the flow editor.

    For additional information about using the **Input/Output** tab, see the *webMethods Service Development Help* for your release.

11. Select **File > Save**.

12. To test the service directly from Designer, see "Testing Adapter Services" on page 94.

Configuring InvokeEJB 2.1 Services

An InvokeEJB 2.1 adapter service invokes one or more methods on a single remote EJB reference. For more information about the template used to create these services, see "InvokeEJB 2.1 Adapter Service" on page 32. For more information about adapter services, see .

To configure an InvokeEJB 2.1 service

1. Review the steps in "Before Configuring or Managing Adapter Services" on page 80.
2. Start Designer.

   Note:
   Make sure the server with which you want to use Designer is running.

3. Right-click the package in which the service should be contained and select **New > Adapter Service**.
4. Select the parent namespace and type a name for the adapter service. Click **Next**.
5. Select **Adapter for EJB** from the list of available adapter types. Click **Next**.
6. Select the appropriate **Adapter Connection Name** and click **Next**.
7. From the list of available templates, select the **InvokeEJB 2.1** template and click **Finish**.

   The service is created and its parameters and controls are displayed in the adapter service editor.

8. In the editor, select the **InvokeEJB 2.1** tab and specify the following values:

Parameter	Description/Action
EJB Lookup Name	Lists the JNDI lookup names of all available EJBs on the application server. The name you select here determines the values that will appear in the **EJB Remote Method** and **Return Type** parameters. At runtime, the service uses this name to match against a given EJB Handle object. The EJBs are listed alphabetically. By default, the first EJB in the list is initially selected.
EJB Remote Method	Lists the available remote methods for the selected EJB. The method selected will be executed by the service. The methods

Parameter	Description/Action
	exposed by the EJB are listed alphabetically. By default, the first method in this list is initially selected. ■ If the method has parameters, they appear in the parameter list. For each parameter, the default parameter name and the corresponding Java class type are shown in the first two columns of the parameter list. You may override a parameter's default name by entering a new name in the **Override Parameter Name** column. The value that appears in this column for a parameter is the value that appears in the configured method's input signature at runtime. ■ If the method takes no parameters, the parameter list is empty.
Return Type	Lists the remote EJB method's Java return type for the selected EJB lookup name. This value is read-only.
Expand Collections?	Allows you to specify the format of the output of the remote method. When selected, only the elements in the collection are returned as output. When cleared, the collection itself is returned as output. By default, this option is selected.
Allow Null Return Value?	Allows Adapter for Enterprise Javabeans to return a null value when the EJB method has a non-void return type. When selected, Adapter for Enterprise Javabeans does not throw any exception if the EJB method returns a null value, and returns an object array with no elements in it. When cleared, Adapter for Enterprise Javabeans throws an exception if the EJB method returns a null value. By default, this option is cleared.

Note:
The EJB lookup name and EJB remote method are displayed by default. If the lookup name and method have input parameters, the parameters are not displayed in the service's input signature until you reload the adapter values. Save the adapter service if you want to use the default values. If you change the lookup name or the remote method, the values are automatically refreshed and the parameters appear in the adapter service editor. For more information about reloading adapter values, see "Reloading Adapter Values" on page 97.

9. You can select the **Adapter Settings** tab at any time to confirm adapter properties such as adapter type, connection name, and service template, as needed.

10. The **Input/Output** tab lists the input and output parameters, if any, for the method. If the remote method has parameters, their names and types will appear in the input signature under the EJBObject_Args document. If the service is subsequently used in a flow, its input and output signatures will be visible in the flow editor.

    For additional information about using the **Input/Output** tab, see the *webMethods Service Development Help* for your release.

11. Select **File > Save**.

12. To test the service directly from Designer, you must first include the InvokeEJB 2.1 service in a flow service. In the flow service you must map the output of a CreateEJB 2.1 adapter service (the remote EJBs it created) to the input of the InvokeEJB 2.1 adapter service. You cannot run an InvokeEJB 2.1 adapter service as a standalone service. For more information about testing an adapter service, see "Testing Adapter Services" on page 94.

Configuring CreateInvokeEJB 2.1 Services

A CreateInvokeEJB 2.1 adapter service combines the functionality of a CreateEJB 2.1 service and an InvokeEJB 2.1 service, enabling you to create one or more instances of a single EJB class and then invoke a single method on those instances. Additionally, a CreateInvokeEJB 2.1 service automatically calls the EJB-standard remove() method to release each session EJB instance. For more information about the template used to create these services, see "CreateInvokeEJB 2.1 Adapter Service" on page 34. For more information about adapter services, see . For more information about the RemoveEJB service, see "Removing EJBs" on page 93.

To configure a CreateInvokeEJB 2.1 service

1. Review the section "Before Configuring or Managing Adapter Services" on page 80.

2. Start Designer.

   Note:
   Make sure the server with which you want to use Designer is running.

3. Right-click the package in which the service should be contained and select **New > Adapter Service**.

4. Select the parent namespace and type a name for the adapter service. Click **Next**.

5. Select **Adapter for EJB** from the list of available adapter types. Click **Next**.

6. Select the appropriate **Adapter Connection Name** and click **Next**.

7. From the list of available templates, select the **CreateInvokeEJB 2.1** template and click **Finish**.

The service is created and its parameters and controls are displayed in the adapter service editor.

8. In the editor, select the **CreateInvokeEJB 2.1** tab and specify the following values:

Parameter	Description/Action
EJB Lookup Name	Lists the JNDI lookup names of all available EJBs on the application server. The name you select here determines the values that will appear in the **EJB Create Method** and **Return Type** parameters, and also in the **EJB Remote Method** value on the **Method to Invoke** tab. The EJBs are listed alphabetically. By default, the first EJB in the list is initially selected.
EJB Create Method	Lists the available creator/finder methods for the selected EJB lookup name. The method selected will be executed by the service. The methods exposed by the EJB are listed alphabetically. By default, the first method in this list is initially selected. ■ If the method has parameters, they appear in the parameter list. For each parameter, the default parameter name and the corresponding Java class type are shown in the first two columns of the parameter list. You may override a parameter's default name by entering a new name in the **Override Parameter Name** column. The value that appears in this column for a parameter is the value that appears in the configured method's input signature at runtime. ■ If the method takes no parameters, the parameter list is empty.
Return Type	Lists the remote EJB method's Java return type for the selected EJB lookup name. This value is read-only.

Note:
The EJB lookup name and EJB create method are displayed by default. If the lookup name and method have input parameters, the parameters are not displayed in the service's input signature until you reload the adapter values. Save the adapter service if you want to use the default values. If you change the lookup name or the create method, the values are automatically refreshed and the parameters appear in the adapter service editor. For more information about reloading adapter values, see "Reloading Adapter Values" on page 97.

9. Select the **Method to Invoke** tab and specify the following values:

Parameter	Description/Action
EJB Remote Method	Lists the available remote methods for the selected EJB. The method selected will be executed by the service. The methods exposed by the EJB are listed alphabetically. By default, the first method in this list is initially selected.

Parameter	Description/Action
	■ If the method has parameters, they appear in the parameter list. For each parameter, the default parameter name and the corresponding Java class type are shown in the first two columns of the parameter list. You may override a parameter's default name by entering a new name in the **Override Parameter Name** column. The value that appears in this column for a parameter is the value that appears in the configured method's input signature at runtime. ■ If the method takes no parameters, the parameter list is empty.
Return Type	Lists the remote EJB method's Java return type for the selected EJB lookup name. This value is read-only.
Expand Collections?	Allows you to specify the format of the output of the remote method. When selected, only the elements in the collection are returned as output. When cleared, the collection itself is returned as output. By default, this option is selected.
Allow Null Return Value?	Allows Adapter for Enterprise Javabeans to return a null value when the EJB method has a non-void return type. When selected, Adapter for Enterprise Javabeans does not throw any exception if the EJB method returns a null value, and returns an object array with no elements in it. When cleared, Adapter for Enterprise Javabeans throws an exception if the EJB method returns a null value. By default, this option is cleared.

Note:
The EJB lookup name and EJB remote method are displayed by default. If the lookup name and method have input parameters, the parameters are not displayed in the service's input signature until you reload the adapter values. Save the adapter service if you want to use the default values. If you change the lookup name or the remote method, the values are automatically refreshed and the parameters appear in the adapter service editor. For more information about reloading adapter values, see “Reloading Adapter Values” on page 97.

10. You can select the **Adapter Settings** tab at any time to confirm adapter properties such as adapter type, connection name, and service template, as needed.

11. The **Input/Output** tab lists the input and output parameters for methods. If the create method has parameters, their names and types appear under the EJBHome_Args document. Similarly, if the bean method has parameters, their names and types appear under the EJBObject_Args

document. If the service is subsequently used in a flow, its input and output signature will be visible in the flow editor.

For additional information about using the **Input/Output** tab, see the *webMethods Service Development Help* for your release.

12. Select **File > Save**.

13. To test the service directly from Designer, see "Testing Adapter Services" on page 94.

Configuring FetchEJB 3.0 Services

A FetchEJB 3.0 adapter service invokes a specific EJB finder method on the EJB's remote interface. For more information about the template used to create these services, see "FetchEJB 3.0 Adapter Service" on page 36. For more information about adapter services, see . For more information about the RemoveEJB service, see "Removing EJBs" on page 93.

To configure a FetchEJB 3.0 adapter service

1. Review the section "Before Configuring or Managing Adapter Services" on page 80.

2. Start Designer.

   Note:
   Make sure the server with which you want to use Designer is running.

3. Right-click the package in which the service should be contained and select **New > Adapter Service**.

4. Select the parent namespace and type a name for the adapter service. Click **Next**.

5. Select **Adapter for EJB** as the adapter type and click **Next**.

6. Select the appropriate **Adapter Connection Name** and click **Next**.

7. From the list of available templates, select the **FetchEJB 3.0** template and click **Finish**.

   The service is created and its parameters and controls are displayed in the adapter service editor.

8. In the editor, select the **FetchEJB 3.0** tab and specify the following values:

Parameter	Description/Action
EJB Lookup Name	Lists the JNDI lookup names of all available 3.0 EJBs on the application server. The selected JNDI name determines the EJB to

Parameter	Description/Action
	be fetched. At runtime, the service uses this name to look up the EJBRemote object. The EJBs are listed alphabetically. By default, the first EJB in the list is initially selected.

9. You can select the **Adapter Settings** tab at any time to confirm adapter properties such as adapter type, connection name, and service template, as needed.

10. The **Input/Output** tab lists the input and output parameters, if any, for the method. If the create method has parameters, their names and types appear under the EJBObject_Args document. If this service is later used in a flow, its input and output signatures will be visible in the flow editor.

    For additional information about using the **Input/Output** tab, see the *webMethods Service Development Help* for your release.

11. Click **File > Save**.

12. To test the service directly from Designer, see "Testing Adapter Services" on page 94.

Configuring InvokeEJB 3.0 Services

An InvokeEJB 3.0 adapter service invokes one or more methods on a single remote EJB reference. For more information about the template used to create these services, see "InvokeEJB 3.0 Adapter Service" on page 37. For more information about adapter services, see . For more information about the RemoveEJB service, see "Removing EJBs" on page 93.

To configure an InvokeEJB 3.0 service

1. Review the steps in "Before Configuring or Managing Adapter Services" on page 80.

2. Start Designer.

    Note:
    Make sure the server with which you want to use Designer is running.

3. Right-click the package in which the service should be contained and select **New > Adapter Service**.

4. Select the parent namespace and type a name for the adapter service. Click **Next**.

5. Select **Adapter for EJB** as the adapter type and click **Next**.

6. Select the appropriate **Adapter Connection Name** and click **Next**.

7. From the list of available templates, select the **InvokeEJB 3.0** template and click **Finish**.

8. In the adapter service editor, select the **InvokeEJB 3.0** tab and specify the following values:

Parameter	Description/Action
EJB Lookup Name	Lists the JNDI lookup names of all available EJBs on the application server. The name you select here determines the values that will appear in the EJB Remote Method and Return Type parameters. At runtime, the service uses this name to match against a given EJB Handle object. The EJBs are listed alphabetically. By default, the first EJB in the list is initially selected.
EJB Remote Method	Lists the available remote methods for the selected EJB. The method selected will be executed by the service. The methods exposed by the EJB are listed alphabetically. By default, the first method in this list is initially selected. ■ If the method has parameters, they appear in the parameter list. For each parameter, the default parameter name and the corresponding Java class type are shown in the first two columns of the parameter list. You may override a parameter's default name by entering a new name in the Override Parameter Name column. The value that appears in this column for a parameter is the value that appears in the configured method's input signature at runtime. ■ If the method takes no parameters, the parameter list is empty.
Return Type	Lists the remote EJB method's Java return type for the selected EJB lookup name. This value is read-only.
Expand Collections?	Allows you to specify the format of the output of the remote method. When selected, only the elements in the collection are returned as output. When cleared, the collection itself is returned as output. By default, this option is selected.
Allow Null Return Value?	Allows Adapter for Enterprise Javabeans to return a null value when the EJB method has a non-void return type. When selected, Adapter for Enterprise Javabeans does not throw any exception if the EJB method returns a null value, and returns an object array with no elements in it. When cleared, Adapter for Enterprise Javabeans throws an exception if the EJB method returns a null value. By default, this option is cleared.

> **Note:**
> The EJB lookup name and EJB remote method are displayed by default. If the lookup name and method have input parameters, the parameters are not displayed in the service's input signature until you reload the adapter values. Save the adapter service if you want to use the default values. If you change the lookup name or the remote method, the values are automatically refreshed and the parameters appear in the adapter service editor. For more information about reloading adapter values, see "Reloading Adapter Values" on page 97.

9. You can select the **Adapter Settings** tab at any time to confirm adapter properties such as adapter type, connection name, and service template, as needed.

10. The **Input/Output** tab lists the input and output parameters, if any, for the method. If the remote method has parameters, their names and types will appear in the input signature under the EJBObject_Args document. If the service is subsequently used in a flow, its input and output signatures will be visible in the flow editor.

    For additional information about using the Input/Output tab, see the *webMethods Service Development Help* for your release.

11. Select **File > Save**.

12. To test the service directly from Designer, you must first include the InvokeEJB 3.0 service in a flow service. In the flow service you must map the output of a FetchEJB 3.0 adapter service (the remote EJBs it created) to the input of the InvokeEJB 3.0 adapter service. You cannot run an InvokeEJB 3.0 adapter service as a standalone service. For more information about testing an adapter service, see "Testing Adapter Services" on page 94.

Configuring FetchInvokeEJB 3.0 Services

A FetchInvokeEJB 3.0 adapter service combines the functionality of a FetchEJB 3.0 service and an InvokeEJB 3.0 service, enabling you to create one or more instances of a single EJB class and then invoke a single method on those instances. Additionally, a FetchEJB 3.0 service automatically calls the EJB-standard remove() method to release each session EJB instance. For more information about the template used to create these services, see "FetchInvokeEJB 3.0 Adapter Service" on page 39. For more information about adapter services, see . For more information about the RemoveEJB service, see "Removing EJBs" on page 93.

To configure a FetchInvokeEJB 3.0 service

1. Review the steps in "Before Configuring or Managing Adapter Services" on page 80.

2. Start Designer.

> **Note:**
> Make sure the server with which you want to use Designer is running.

3. Right-click the package in which the service should be contained and select **New > Adapter Service**.

4. Select the parent namespace and type a name for the adapter service. Click **Next**.

5. Select **Adapter for EJB** as the adapter type and click **Next**.

6. Select the appropriate **Adapter Connection Name** and click **Next**.

7. From the list of available templates, select the **FetchInvokeEJB 3.0** template and click **Finish**.

8. In the adapter service editor, select the **FetchInvokeEJB 3.0** tab and specify the following values:

Parameter	Description/Action
EJB Lookup Name	Lists the JNDI lookup names of all available 3.0 EJBs on the application server. The selected JNDI name determines the EJB to be fetched. At runtime, the service uses this name to look up the EJBRemote object. The EJBs are listed alphabetically. By default, the first EJB in the list is initially selected.
EJB Remote Method	Lists the available finder method for the selected EJB lookup name. The method selected will be executed by the service. The methods exposed by the EJB are listed alphabetically. By default, the first method in this list is initially selected. ■ If the method has parameters, they appear in the parameter list. For each parameter, the default parameter name and the corresponding Java class type are shown in the first two columns of the parameter list. You may override a parameter's default name by entering a new name in the Override Parameter Name column. The value that appears in this column for a parameter is the value that appears in the configured method's input signature at runtime. ■ If the method takes no parameters, the parameter list is empty.
Return Type	Lists the remote EJB method's Java return type for the selected EJB lookup name. This value is read-only.
Expand Collections?	Allows you to specify the format of the output of the remote method. When selected, only the elements in the collection are returned as output. When cleared, the collection itself is returned as output. By default, this option is selected.
Allow Null Return Value?	Allows Adapter for Enterprise Javabeans to return a null value when the EJB method has a non-void return type.

Parameter	Description/Action
	When selected, Adapter for Enterprise Javabeans does not throw any exception if the EJB method returns a null value, and returns an object array with no elements in it. When cleared, Adapter for Enterprise Javabeans throws an exception if the EJB method returns a null value. By default, this option is cleared.

9. You can select the **Adapter Settings** tab at any time to confirm adapter properties such as adapter type, connection name, and service template, as needed.

10. The **Input/Output** tab lists the input and output parameters for methods. If the fetch method has parameters, their names and types appear under the EJBObject_Args document. Similarly, if the bean method has parameters, their names and types appear under the EJBObject_Args document. If the service is subsequently used in a flow, its input and output signature will be visible in the flow editor.

    For additional information about using the **Input/Output** tab, see the *webMethods Service Development Help* for your release.

11. Select **File > Save**.

12. To test the service directly from Designer, see "Testing Adapter Services" on page 94.

Removing EJBs

Adapter for Enterprise Javabeans also provides a non-configurable built-in service, RemoveEJB. The RemoveEJB service is packaged with Adapter for Enterprise Javabeans and is available in the adapter's public namespace at pub.ejbadapter.removeEJB.

Note:
The RemoveEJB service is used for EJB 2.1 or earlier services only, that is for CreateEJB 2.1, InvokeEJB 2.1, and CreateInvokeEJB 2.1. For 3.0 services, as there is no home interface, the bean is automatically released by the application server.

RemoveEJB is a generic non-configurable service that takes an EJB Handle as its only parameter and invokes the EJB-standard remove() method on the EJB object represented by that handle. If successful, RemoveEJB produces no output whatsoever. If it fails, it throws an AdapterServiceException.

To control the remote EJB's life-cycle, call the RemoveEJB service from a flow service to inform the application server that a particular EJB is no longer being used by the client. This is most important when dealing with stateful or entity EJBs.

Calling the RemoveEJB service on an EJB has slightly different consequences depending upon the type of that EJB:

- For stateful session EJBs, its effect is to signal the end of the session to the application server.

- For entity EJBs, the RemoveEJB service causes the underlying entity to be removed from the EJB container. For example, CreateEJB 2.1 is used to create a new entity EJB, which may result in the application server creating a new row in a table to hold that EJB's state. Subsequently passing the Handle of that EJB to the RemoveEJB service causes the application server to delete that row from the table.
- For stateless session EJBs, it has no effect in the client. RemoveEJB simply notifies the application server that it is no longer using the EJB so that the server may, at its discretion, perform any housekeeping tasks.

The application server determines how and when the tasks above occur. If the client invokes the RemoveEJB service against an entity EJB, the client should consider that EJB to be deleted even if the application server does not actually delete the EJB at that point in time.

Testing Adapter Services

You use Designer to test adapter services. For information about testing and debugging services, see the *webMethods Service Development Help* for your release.

To test an adapter service

1. Review the steps in "Before Configuring or Managing Adapter Services" on page 80.
2. In Designer, expand the package and folder that contain the service you want to test.
3. Double-click the service you want to test.

   Designer displays the configured service in the service template's Adapter Service Editor.
4. Select **Run > Run As > Run Service**.
5. For every service input field, you will be prompted to enter an input value. Enter a value for each input field and then click **OK**.
6. Click the **Results** tab to view the output from this service.

   Note:
   Credentials you provide in the *username* and *password* fields override any connection.

   Note:
   Specifying the *$connectionName* input parameter changes the connection for this execution of the service. To reconfigure the service to use a different connection, use the setAdapterServiceNodeConnection, which is located in the WmART package's pub.art.service folder. For more information about this service, see the *webMethods Integration Server Built-In Services Reference* for your release.

Viewing Adapter Services

You use Designer to view adapter services.

To view an adapter service

1. Review the steps in "Before Configuring or Managing Adapter Services" on page 80.
2. In Designer, expand the package and folder that contain the service you want to view.
3. Double click the service you want to view.

   Designer displays the configured service in the service template's Adapter Service Editor.

Editing Adapter Services

You use Designer to edit adapter services.

To edit an adapter service

1. In Designer, browse to and open the adapter service that you want to edit.
2. Double-click the service that you want to edit.

   Designer displays the adapter service in the service template's Adapter Service Editor.
3. Do one of the following:
   - If you have the VCS Integration feature enabled, right-click the service and select **Check Out**.
   - If you do not have the VCS Integration feature enabled, right-click the service and select **Lock for Edit**.
   - If you are using the local service development feature, from the **Team** menu in Designer, select the appropriate option to check out the service. The options available in the **Team** menu depend on the VCS client that you use.
4. Modify the values for the adapter service's parameters as needed. For detailed descriptions of the service's parameters, see the section on configuring a service for the specific type of service you want to edit.
5. After you complete your modifications, save the service and do one of the following:
   - If you have the VCS Integration feature enabled, right-click the service and select **Check In**. Enter a check-in comment and click **OK**.

- If you do not have the VCS Integration feature enabled, right-click the service and select **Unlock**.
- If you are using the local service development feature, from the **Team** menu in Designer, select the appropriate option to check in the service. The options available in the **Team** menu depend on the VCS client that you use.

6. Save the service.

Deleting Adapter Services

You use Designer to delete adapter services.

To delete an adapter service

1. Review the steps in "Before Configuring or Managing Adapter Services" on page 80.
2. In Designer, expand the package and folder that contain the service you want to delete.
3. Right-click the service and click **Delete**.

Validating Adapter Service Values

Designer enables Adapter for Enterprise Javabeans to validate user-defined data for adapter services at design time. You can validate the values for a single adapter service or you can configure Designer to always validate the values for adapter services. Both options could potentially slow your design-time operations.

When you enable data validation for a single adapter service, Designer compares the service values against the resource data that has already been fetched from the selected adapter.

If you select the option to always validate values for adapter services, it will do so for all webMethods 6.x adapters installed on Integration Server.

For more information about the **Adapter Service/Notification Editor**and other Designer menu options and toolbar icons, see the *webMethods Service Development Help* for your release.

Enabling Automatic Data Validation for a Single Adapter Service

To enable automatic data validation for a single adapter service

1. Review the steps in "Before Configuring or Managing Adapter Services" on page 80.
2. In Designer, expand the package and folder that contain the service for which you want to enable automatic validation.

3. Double-click the service for which you want to validate the data.

   Designer displays the configured adapter service in the service template's Adapter Service Editor.

4. Click the ▶ icon.

Validating Adapter Service Values for all Adapter Services

To validate adapter service values for all adapter services

1. Review the steps in "Before Configuring or Managing Adapter Services" on page 80.

2. Start Designer.

3. Select the **Window > Preferences >Software AG> Service Development > Adapter Service/Notification Editor** item.

4. Enable the **Automatic data validation** option.

5. Click **OK**.

Reloading Adapter Values

Designer enables Adapter for Enterprise Javabeans to reload and validate user-defined data for adapter services at design time. You can reload values for a single adapter service or you can configure Designer so it automatically reloads the values for adapter services. Both options could potentially slow your design-time operations.

When you reload adapter values for a single adapter service, Designer compares the service values against the resource data that has already been fetched from the selected adapter.

If you select the option to always reload values for adapter services, it will do so for all webMethods 6.x adapters installed on Integration Server.

For more information about the **Adapter Service/Notification Editor**, other menu options, and toolbar icons, see the *webMethods Service Development Help* for your release.

Reloading Adapter Values for a Single Adapter Service

To reload the adapter values for a single adapter service

1. Review the steps in "Before Configuring or Managing Adapter Services" on page 80.

2. In Designer, expand the package and folder that contain the service for which you want to enable automatic validation.

3. Double-click the service for which you want to validate the data.

   Designer displays the configured adapter service in the service template's Adapter Service Editor.

4. Click the icon.

Reloading Adapter Values for all Adapter Services

To reload the adapter values for all adapter services

1. Review the steps in "Before Configuring or Managing Adapter Services" on page 80.

2. Start Designer.

3. Select the **Window > Preferences >Software AG> Service Development > Adapter Service/Notification Editor** item.

4. Enable the **Automatic polling of adapter metadata** option.

5. Click **OK**.

6 Invoking webMethods Services From an EJB

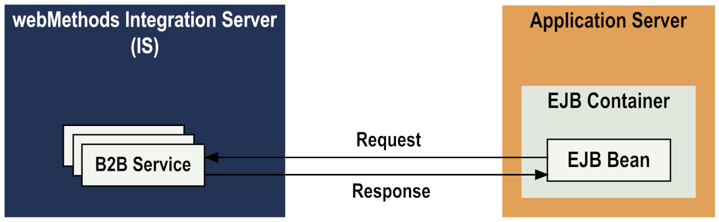
Overview

In addition to creating services with Adapter for Enterprise Javabeans that enable you to invoke methods on EJBs deployed on an application server, you can also invoke services on webMethods Integration Server (IS) from an EJB. The IS services invoked can be any service available on Integration Server. This model does not interact with Adapter for Enterprise Javabeans, although an EJB developer could use this functionality to invoke an adapter service created by Adapter for Enterprise Javabeans.

To demonstrate this functionality, you should work with the following components:

- Sample EJB application (available for download on the Software AG TECHcommunity website)
- Sample webMethods Java service (provided with the adapter)
- webMethods APIs (packaged with Integration Server)

Through the APIs, an EJB may establish an HTTP connection to Integration Server and instruct it to execute a particular service. Upon completion, the service results are returned to the EJB. Upon receiving the results, you can have the EJB, process this information, to suit its needs. Inputs to and outputs from the service are passed using webMethods proprietary IData objects. Included in the API used by the EJB are tools for creating and manipulating IData instances. This interaction is logically depicted in the following diagram.



The following sections describe each of these components and explain how an EJB developer might implement code that invokes services on Integration Server.

Sample EJB Application

To use the sample EJB application, download it from the Software AG TECHcommunity website and extract the downloaded zip file to the *Integration Server_directory* \packages\WmEJBAdapter\templates\EJBToWebmSamples directory. The sample EJB application contains EJB source code that demonstrates the procedure for invoking services running on Integration Server from within an EJB. The sample EJB application must be installed on your application server in order to demonstrate this facility.

The sample EJB application includes the following files:

- Source code for a simple stateless session EJB (HelloBean) that talks to a simple stand-alone Java service that is part of the adapter (see "Sample webMethods Java Service" on page 101).
- EJB deployment descriptor files for the HelloBean EJB, one set for each supported application server.

- Source code for a simple EJB client (HelloEJBClient) designed to run on the application server and used to invoke HelloBean.
- JNDI properties file needed by HelloEJBClient, one for each supported application server.
- Ant build scripts used to compile HelloBean and HelloEJBClient, deploy HelloBean, and run HelloEJBClient. There is one build script for each supported application server.
- Readme file describing how to build the sample EJB and the EJB client, deploy the sample EJB, and run the EJB client. There is one readme file for each supported application server (ReadMe*appserver*.txt).

Sample webMethods Java Service

The sample testEJBToWebm service is a standalone Java service and built in to Adapter for Enterprise Javabeans. The service appears in the IS namespace as pub.ejbadapter.testEJBToWebm.

The HelloBean sample EJB will attempt to invoke testEJBToWebm. The testEJBToWebmservice simply echoes the string sent by the sample EJB back to the pipeline.

webMethods APIs

The sample EJB uses the webMethods APIs to establish communications with Integration Server and subsequently interact with the testEJBToWebm Java service. The webMethods APIs are available in the packages com.wm.app.b2b.client and com.wm.data, which can be found in the following file:

Integration Server_directory \lib\client.jar

- The com.wm.app.b2b.client package contains classes that you use to build clients for Integration Server (including clients that use the guaranteed-delivery facility).
- The com.wm.data package contains classes that you use create and manipulate IData objects.

Documentation for these classes is available in the online help in the *Integration Server_directory* \doc\api\Java directory.

Running the Sample EJB

Use the sample EJB application to see how to implement the logic in your code to invoke IS services from an EJB.

To run the sample EJB, follow the instructions in the ReadMeappserver.txt file in the EJBToWebmSamples folder that tells you how to find the client.jar and how to build and deploy the sample EJB code on your application server.

Basic Flow of Events

Note:

The EJB must have access to the client jar file containing classes needed to engage in a request-response dialogue with Integration Server.

At a high level, the steps to invoking an IS service from an EJB are as follows:

1. On the application server, create and deploy an EJB that uses the webMethods APIs in client.jar for the purpose of executing a service on Integration Server.

2. The EJB creates a context (connection) with Integration Server, providing a URL, user name, and password.

3. The EJB constructs an IData pipeline, if necessary, inserting appropriate service input values.

4. The EJB posts to Integration Server a request containing a valid service name and the pipeline constructed in step 3.

5. The EJB receives a response from Integration Server and uses the provided APIs to access and process the values contained in the returned IData pipeline.

6. The EJB successfully receives and processes the desired results from the service it executed.

7 Predefined Health Indicator

Predefined Health Indicator

Microservices Runtime includes predefined health indicators for some of its basic components. The health indicator captures the connection details for all the WmART based adapters at runtime. For more information, see *webMethods Adapter Runtime User's Guide*.

8 Administrator APIs

Administrator APIs

The Administrator APIs are available for Adapter for Enterprise JavaBeans. For more information about Administrator APIs and samples, see *webMethods Adapter Runtime User's Guide*.

9 Configuration Variables Templates for Adapter Assets in Microservices Runtime

Configuration Variables Templates for Adapter Assets in Microservices Runtime

The webMethods Adapter Runtime (ART) asset properties that can be configured from Integration Server Administrator are available in the configuration variables template (`application.properties` file) generated by Microservices Runtime. For more information, see *webMethods Adapter Runtime User's Guide* and *Developing Microservices with webMethods Microservices Runtime*.

10 Adapter Logging and Exception Handling

Overview

The following sections describe message logging and Adapter for Enterprise Javabeans exception handling. A list of error codes and supporting information appears at the end of this chapter.

Adapter Logging Levels

Adapter for Enterprise Javabeans uses Integration Server's logging mechanism to log messages. You can configure and view Integration Server's logs to monitor and troubleshoot Adapter for Enterprise Javabeans. For detailed information about logging into Integration Server, including instructions for configuring and viewing the different kinds of logs supported by the server, see the *webMethods Integration Server Administrator's Guide* for your release.

You can configure different logging levels for Adapter for Enterprise Javabeans.

Accessing the Adapter's Logging Information

To access the Adapter's logging information

1. From the Integration Server Administrator screen, select **Settings > Logging**.

   On the Logging Settings screen, the **Loggers** section has **Adapters** included in the **Facility** section.

2. Expand the **Adapters** tree to see a list of all installed adapters with their code number and adapter description, along with the logging level.

Changing Logging Settings

To change the logging settings for Adapter for Enterprise Javabeans

1. Click **Edit Logging Settings**. Select the required **Level of logging** for Adapter for Enterprise Javabeans.

2. After making your changes, click **Save Changes**.

3. For complete information about specifying the amount and type of information to include in the log, see the *webMethods Audit Logging Guide* for your release.

Adapter Message Logging

Integration Server maintains several types of logs; however, Adapter for Enterprise Javabeans only logs messages to the Audit, Error and Server logs, as described in the table below:

Log	Description
Audit Log	You can monitor individual adapter services using the audit log as you would audit any service in Integration Server. The audit properties for an adapter service are available in each Adapter for Enterprise Javabeans service template on the **Audit** tab.
Error Log	Adapter for Enterprise Javabeans automatically posts critical-level and error-level log messages to the server's Error log. These log messages will appear as Adapter Runtime messages.
Server Log	Adapter for Enterprise Javabeans posts messages to the Server log, depending on how the server log is configured. Critical-level through debug-level log messages appear as Adapter Runtime log messages. V1-Verbose1 or V4-Verbose4 log messages appear as Adapter for Enterprise Javabeans log messages.

Note that Adapter for Enterprise Javabeans does not log debug messages at the Debug level. Rather, it logs all of its debug output at the Verbose3 or Verbose4 level. Verbose3 is used for general debug logging and Verbose4 is used to log problems encountered while traversing entries on the JNDI server.

Adapter for Enterprise Javabeans log messages appear in the following format: ADA.0640.*nnnnc*, where:

- ADA is the facility code that indicates the message is from an adapter.
- 0640 (or 640) is the major error code for Adapter for Enterprise Javabeans, which indicates that the message is generated by Adapter for Enterprise Javabeans.
- *nnnn* represents the error's minor code. For detailed descriptions of the minor codes for Adapter for Enterprise Javabeans, see "Adapter for Enterprise Javabeans Error Messages" on page 113.
- *c* represents the message's severity level (optional).

Because Adapter for Enterprise Javabeans works in conjunction with the WmART package, exceptions generated within the adapter frequently will appear within log messages for the WmART package.

To monitor Adapter for Enterprise Javabeans log messages in the Server log, ensure that your server log's logging settings are configured to monitor the following facilities:

- 0113 Adapter Runtime (Managed Object)
- 0114 Adapter Runtime
- 0117 Adapter Runtime (Adapter Service)
- 0118 Adapter Runtime (Connection)
- 0121 Adapter Runtime (SCC Transaction Manager)
- 0126 Adapter Runtime (SCC Connection Manager)

Adapter Exception Handling

Adapter for Enterprise Javabeans throws three exception classes that you should be aware of as you build integrations using the adapter: AdapterException, AdapterConnectionException, and AdapterServiceException. In all cases, the adapter passes the underlying exception on to the Adapter Runtime, which wraps it in a container exception that it then passes on to Integration Server. Integration Server then serializes the exception and returns it to the client service. Typically, that client (for example, a flow or Java service that calls an adapter service) will include logic that traps these exceptions and branches accordingly. For information about how to trap the exception in a flow, see the *webMethods Service Development Help* for your release.

With Adapter for Enterprise Javabeans, errors typically originate from one of two sources: JNDI or EJB:

- JNDI exceptions generally are manifested as instances of javax.naming.NamingException. This is simply a generalized wrapper for a list of low-level causes; some fatal, some not. Adapter for Enterprise Javabeans examines the NamingException object to determine the underlying cause, and then wraps it in either an AdapterException or an AdapterConnectionException. If the exception occurs during lookup or introspection of a specific EJB, the adapter throws the wrapped exception to the Adapter Runtime. If the exception occurs while the adapter is while navigating the JNDI tree, the adapter logs a message in the Server log at level 8 (Verbose4) and continues.
- EJB exceptions are typically wrapped as java.rmi.RemoteException. As with NamingExceptions, Adapter for Enterprise Javabeans examines it for the root cause, wraps it in an AdapterException or AdapterConnectionException, and throws it to the Adapter Runtime.

AdapterException

Adapter for Enterprise Javabeans throws an AdapterException to report an error related to the back-end resource (the application server) that does not involve the connection to that resource. An example of this type of error might be that an EJB lookup failed.

AdapterConnectionException

Adapter for Enterprise Javabeans throws an AdapterConnectionException to report a non-recoverable error in the connection to the back-end resource (the application server). In this case, WmART drops the connection from the connection pool and tries to create a new connection.

It then wraps the exception in com.wm.pkg.art.error.DetailedSystemException and throws it to Integration Server.

AdapterServiceException

Adapter for Enterprise Javabeans throws an AdapterServiceException for any error that occurs:

- While starting the adapter
- While shutting down the adapter

- When invoking the standalone Java service RemoveEJB (pub.ejbadapter.removeEJB)

The first two situations indicate some fundamental problem that prevents the adapter from being started or terminated. These are administrative errors that do not typically involve client services using the adapter to access EJBs.

The third situation indicates a runtime error that occurs when the Java service pub.ejbadapter.removeEJB attempts to invoke a remote EJB's remove method. In this case, the client service may be involved and will likely need to trap the error.

Reporting Non-Fatal Connection Errors

To report non-fatal connection errors correctly, you must convert them to fatal.

To convert non-fatal connection errors to fatal, you must set the watt.ejbadapter.fatalErrors watt parameter to list the errors that are to be reported as fatal.

To set the watt.ejbadapter.fatalErrors parameter

1. In Integration Server Administrator, click **Settings > Extended > Edit Extended Settings**.

2. In the Extended Settings editor, type the following:

   ```
   watt.ejbadapter.fatalErrors=<error_1,error_2,...
   error_n>
   ```

   For example:

   ```
   watt.ejbadapter.fatalErrors=
   +COMM_FAILURE,NoRouteToHostException,InvocationTargetException
   ,ConnectException
   ```

   Note:
   The parameter name is case sensitive.

3. Click **Save Changes**. A key for the new property appears in the Extended Settings list.

4. Click **Show and Hide Keys**. You will see watt.ejbadapter.fatalErrors listed as a Key and as visible.

5. Restart Integration Server.

Adapter for Enterprise Javabeans Error Messages

Adapter for Enterprise Javabeans categorizes its minor code numbers as follows:

- **1000-1999.** Adapter-specific errors, warnings, and informational messages
- **3000-3999.** Connection related errors, warnings, and informational messages

- **5000-5999.** Adapter service related errors, warnings, and informational messages

Adapter Error Codes

Error Code	Description
1001	**In service {0}: Pipeline is empty**
	Explanation: The content of the IData pipeline sent to the testEJBToWebm sample service is null.
	Action: Ensure that the client from which you are invoking testEJBToWebm is sending an IData instance that conforms to this service's input signature. Specifically, it should contain a single String field named "inVal".
1002	**In service {0}: Failed to cast object in pipeline to expected type**
	Explanation: The value sent to the testEJBToWebm sample service is not a java.lang.String.
	Action: Ensure that the client from which you are invoking testEJBToWebm is sending an IData instance that conforms to this service's input signature. Specifically, it should contain a single String field named "inVal".
3000	**Caught exception attempting to create WmManagedConnection for server type: {0}**
	Explanation: An unexpected exception occurred attempting to create an instance of a connection of the indicated type.
	Action: Contact Software AG Global Support.
3001	**JNDI properties file is null**
	Explanation: User must enter a path to a text file containing the relevant JNDI properties.
	Action: Ensure that the Properties File Name parameter on the Connection Types screen contains the path to a valid JNDI properties file.
3002	**Failed to locate JNDI properties file: {0}**
	Explanation: While attempting to enable a connection, the adapter could not read the specified JNDI properties file.
	Action: Ensure that the Properties File Name parameter on the Connection Types screen contains the path to a valid JNDI properties file. Also ensure that the file can be read by the adapter at that path.
3003	**Failed to load JNDI properties file: {0}**

Error Code	Description
	Explanation: The contents of the specified JNDI properties file do not support the java.util.Properties standard.
	Action: Ensure that the contents of the specified file support the java.util.Properties standard.
3005	**Failed to instantiate XAResource from {0}**
	Explanation: The adapter failed to obtain an instance of XAResource from the application server using the specified string.
	Action: Ensure that the string in the XAResource Source parameter on the Connection Types screen conforms to the application server vendor's requirements for obtaining an XAResource instance.
3006	**Failed to get InitialContext**
	Explanation: The adapter failed to create an InitialContext for the configured application server.
	Action: Ensure that the application server and its JNDI implementation are running and can be reached over the network. Ensure that the necessary client-side jar files for this back-end are installed under the WmEJBAdapter/code/jars directory. Also ensure that the contents of the specified JNDI properties file are correct.
3007	**Failed to lookup UserTransaction object: {0}**
	Explanation: The adapter failed to obtain the specified UserTransaction instance from the back-end.
	Action: Ensure that the indicated UserTransaction lookup name is exposed in the application server's JNDI implementation and that it is accessible to the adapter.
3008	**Failed to locate any supported connection classes for this factory: {0}**
	Explanation: The indicated connection factory class could not find any implementor classes for this type of connection.
	Action: Contact Software AG Global Support.
3009	**Failed to release InitialContext**
	Explanation: An error occurred closing the InitialContext.
	Action: Contact Software AG Global Support.
3010	**Failed to create EJBDescriptor for: {0} reason: {1}**

Error Code	Description
	Explanation: An unexpected problem occurred introspecting an individual EJB. This error might occur if the EJB does not expose public methods, its home or remote interface class is null, or it is in some other way corrupt.
	Action: Ensure that the specified EJB is configured and deployed correctly on the application server.
3011	**Failed to lookup bean: {0}**
	Explanation: The adapter failed to look up the specified EJB on the application server.
	Action: Ensure that the specified EJB is configured and deployed correctly on the application server.
3012	**Failed to cast {0} to EJBHome**
	Explanation: The adapter could not type cast the specified EJB remote home to a javax.ejb.EJBHome instance.
	Action: Contact Software AG Global Support.
3013	**Failed to get NameParser for this context**
	Explanation: The adapter failed to obtain the javax.naming.NameParser instance from the current InitialContext.
	Action: Contact Software AG Global Support.
3014	**Failed to parse context name: {0}**
	Explanation: The connection failed to parse the specified root JNDI context.
	Action: Ensure that the adapter has access to JNDI and is authorized to read the specified context.
3015	**Failed to list contents of context: {0}**
	Explanation: The connection failed to list the JNDI bindings at the specified context.
	Action: Ensure that the adapter has access to JNDI and is authorized to read the specified context.
3016	**Failed to enumerate item in context: {0}, reason: {1}**
	Explanation: The connection could not isolate a binding in the specified JNDI context.
	Action: Ensure that the adapter has access to JNDI and is authorized to read the specified context.

Error Code	Description
3019	**Nested exception is {0}- {1}**
	Explanation: An unexpected error has occurred while traversing the bindings in the application server's JNDI implementation.
	Action: Contact Software AG Global Support.
3021	**Failed to lookup object bound at {0}: {1}**
	Explanation: The connection was unable to look up the specified object in the application server's JNDI implementation.
	Action: For certain types of objects deployed in some JNDI servers, this error may not be unexpected and can be safely ignored. If the specified object is an EJB, ensure that it is configured/deployed properly on the application server and is accessible to the adapter.
3022	**Object bound at {0} is null**
	Explanation: The object bound at the specified location in JNDI is null.
	Action: For certain types of objects deployed in some JNDI servers, this error may not be unexpected and can be safely ignored. If the specified object is an EJB, ensure that it is configured/deployed properly on the application server and is accessible to the adapter.
3024	**Failed to retrieve metadata for EJB bound at: {0}**
	Explanation: The adapter failed to obtain the EJB metadata for the specified EJB. This error could occur if the application server implements EJB 1.0 only. Adapter for EJB is not backwardly compatible with EJB 1.0 beans.
	Action: Upgrade to EJB 1.1 or greater.
3025	**Call to EJBHome.getEJBMetaData() failed for {0}: {1}**
	Explanation: An exception occurred while invoking the indicated method on the specified EJB.
	Action: Ensure that the specified EJB is configured and deployed correctly on the application server.
3026	**Failed to get EJBHome interface class object for {0}: {1}**
	Explanation: An exception occurred while attempting to get the remote home class object for the specified EJB.
	Action: Ensure that the specified EJB is configured and deployed correctly on the application server.
3027	**Failed to get remote interface class object for {0}: {1}**

Error Code	Description
	Explanation: An exception occurred while attempting to get the remote bean class object for the specified EJB.
	Action: Ensure that the specified EJB is configured and deployed correctly on the application server.
3028	**Failed to determine bean type for {0}: {1}**
	Explanation: An exception occurred while attempting to determine whether the specified EJB is a session bean or an entity bean.
	Action: Ensure that the specified EJB is configured and deployed correctly on the application server.
3029	**Failed to determine session bean type for {0}: {1}**
	Explanation: An exception occurred while attempting to determine whether the specified session EJB is stateless or stateful. This error could occur if the application server implements EJB 1.0 only. Adapter for EJB is not backwardly-compatible with EJB 1.0 beans.
	Action: Ensure that the specified EJB is configured and deployed correctly on the application server. Upgrade to EJB 1.1 or greater.
3030	**Failed to downcast EJB {0} of type {1} to javax.ejb.EJBHome, reason: {2}**
	Explanation: An attempt to type cast an EJB from the specified class to javax.ejb.EJBHome has failed.
	Action: Ensure that the specified EJB is configured and deployed correctly on the application server.
3031	**Unexpected exception registering resource domains in: {0}**
	Explanation: An unexpected exception occurred registering resource domain names for the given connection class.
	Action: Contact Software AG Global Support.
3032	**Unexpected exception looking-up resource domain: {0}**
	Explanation: An unexpected exception occurred performing a resource lookup on the specified resource domain.
	Action: Contact Software AG Global Support.
3033	**Unexpected exception checking values in resource domain: {0}**
	Explanation: An unexpected error occurred while validating the value(s) assigned to the specified resource domain.
	Action: Contact Software AG Global Support.

Error Code	Description
3034	**Failed to obtain WmEJBConnectionFactory object from parent**
	Explanation: The WmEJBConnection instance could not determine the factory object that created it.
	Action: Contact Software AG Global Support.
3035	**Failed to lookup EJBHome: {0}**
	Explanation: The adapter service failed to look up the remote home object for the specified EJB.
	Action: Ensure that the specified EJB is configured and deployed correctly on the application server.
3036	**Failed to begin local transaction**
	Explanation: An unexpected error occurred while trying to begin a local transaction. Note that certain EJB configurations (for example, those with the Never attribute set) cannot participate in a transaction.
	Action: Ensure that the specified EJB is configured to run in a transaction. For more information, see the sections “EJB Transaction Management” and “Services and Transaction Management” in the Overview chapter of the Adapter for EJB installation and user's guide.
3037	**Failed to commit local transaction**
	Explanation: An unexpected error occurred while trying to commit a local transaction. Note that certain EJB configurations (for example, those with the Never attribute set) cannot participate in a transaction.
	Action: Ensure that the specified EJB is configured to run in a transaction. For more information, see the sections “EJB Transaction Management” and “Services and Transaction Management” in the Overview chapter of the Adapter for EJB installation and user's guide.
3038	**Failed to rollback local transaction**
	Explanation: An unexpected error occurred while trying to roll back a local transaction. Note: Certain EJB configurations (for example, those with the Never attribute set) cannot participate in a transaction.
	Action: Ensure that the specified EJB is configured to run in a transaction. For more information, see the sections “EJB Transaction Management” and “Services and Transaction Management” in the Overview chapter of the Adapter for EJB installation and user's guide.
3039	**Connection factory is {0}, but connection does not implement {1}**

Error Code	Description
	Explanation: The configured connection factory and connection implementation classes are not compatible. This situation can only occur when programmatically creating adapter connections using the WmART Extended Utilities services.
	Action: Contact Software AG Global Support.
3998	**javax.naming.NamingException explanation: {0}**
	Explanation: This is a supplementary message that is logged when a NamingException has occurred. This message logs the output of NamingException.getExplanation().
	Action: This is a generic log message that may be caused by any of several JNDI-related events. Ensure that the problem is not due to the application server JNDI configuration.
3999	**javax.naming.NamingException root cause: {0}**
	Explanation: This is a supplementary message that is logged when a NamingException has occurred. This message logs the output of NamingException.getRootCause().getMessage().
	Action: This is a generic log message that may be caused by any of several JNDI-related events. Ensure that the problem is not due to the application server JNDI configuration.
5000	**WmManagedConnection instance does not implement EJBClient**
	Explanation: The Adapter Runtime has passed a WmManagedConnection object to one of the adapter's services that is not a subclass of com.wm.adapter.wmejb.connection.EJBClient.
	Action: Contact Software AG Global Support.
5001	**Unexpected exception executing adapter service: {0}**
	Explanation: An exception other than javax.naming.NamingException or com.wm.adk.error.AdapterException was thrown during execution of the specified adapter service.
	Action: Contact Software AG Global Support.
5002	**Invocation of home method {0}.{1} returned null**
	Explanation: The specified remote home method returned a null object to the invoking adapter service.
	Action: Ensure that the target EJB method is implemented, configured, and deployed properly on the application server.

Error Code	Description
5003	**InvocationTargetException occurred on method: {0} - {1}**
	Explanation: This type of exception can occur when the arguments provided to the specified remote method cause that method to fail. This situation typically occurs with entity EJB finder methods when the input values yield an empty result set in the target bean method.
	Action: Ensure that the target EJB method is implemented correctly and that the input values provided are valid.
5004	**Actual input signature does not match configured input signature**
	Explanation: The actual IData input signature provided to the adapter service is inconsistent with the configured input signature for that service.
	Action: Ensure that the client invoking the configured adapter service instance passes arguments to that service in compliance with the service's input signature.
5005	**For method: {0} on remote class: {1}, failed to introspect method parameters**
	Explanation: The adapter service failed to locate the class file for an object in the remote method's configured parameters list. This error occurs when the class file is not visible to the adapter's class loader.
	Action: Ensure that the jar file containing the class has been copied into the packages/WmEJBAdapter/code/jars directory.
5006	**Failed to locate method: {0} on remote class: {1}**
	Explanation: The adapter service failed to find the specified method in the specified EJB class. This error can occur if an InvokeEJB 2.1 service is passed an EJB handle for a different EJB type than what it was configured for, or the signature of the EJB class itself has been modified since the adapter service was configured.
	Action: Ensure that the EJB handle you are passing into an InvokeEJB 2.1 service instance matches the EJB class that the service was configured for. If the EJB has changed, re-configure the service instance.
5007	**Failed to invoke remote method: {0}, reason: {1}**
	Explanation: An unexpected exception was thrown while calling a remote method on an EJB's home or bean interface. This error could be due to a number of causes, including a defect in the EJB itself.
	Action: Ensure that the EJB is implemented, configured, and deployed properly and that it is accessible to Adapter for EJB.
5008	**Empty Collection/Enumeration returned by creator method: {0}**

Error Code	Description
	Explanation: The specified method on the EJB home interface was successfully invoked, but returned an empty collection object as its result. This error is due to an implementation defect in the EJB home method itself.
	Action: Correct the logic problem in the indicated method. Re-deploy the EJB.
5009	**Failed to lookup default EJB in resource domain: {0}**
	Explanation: This error indicates a logic defect in the adapter configuration code.
	Action: Contact Software AG Global Support.
5010	**Failed to lookup default method in resource domain: {0}**
	Explanation: While configuring the indicated resource domain, the adapter failed to detect any public methods in an EJBHome or EJBObject interface implementation class. This error could indicate that the EJB itself is corrupt or configured incorrectly. It may also indicate an internal logic defect in the adapter code.
	Action: Ensure that the EJB is implemented, configured, and deployed properly.
5011	**Failed to get EJBObject from Handle**
	Explanation: A call to javax.ejb.Handle.getEJBObject() has failed. The most likely cause is a configuration problem in Adapter for EJB.
	Action: Contact Software AG Global Support.
5012	**Remote method {0}.{1} expected return value of type {2}, but received null**
	Explanation: The adapter service has successfully invoked the specified remote method on the EJB. However, the value returned by that method was null though an object of the indicated type was expected. This error indicates a logic defect in the EJB method itself.
	Action: Correct the defect and re-deploy the EJB.
5013	**Input record to InvokeEJB.execute() is empty**
	Explanation: The input record passed into this adapter service instance at runtime contained no data. This error indicates a defect in the client code that is calling this service. (InvokeEJB is the name of the underlying adapter service template class for an InvokeEJB 2.1 service.)

Error Code	Description
	Action: Ensure that the client service code is implemented correctly. At a minimum, all calls to InvokeEJB.execute must contain an EJB Handle instance in the input pipeline.
5014	**Unexpected exception removing EJB**
	Explanation: The standalone service pub.ejbadapter.removeEJB caught an unexpected exception when calling the method java.ejb.EJBObject.remove() on an EJB. The cause is indeterminate.
	Action: Contact Software AG Global Support.
5015	**java.rmi.RemoteException occurred trying to remove EJB**
	Explanation: The standalone service pub.ejbadapter.removeEJB caught the indicated exception when calling the method java.ejb.EJBObject.remove() on an EJB.
	Action: Contact Software AG Global Support.
5016	**Expected EJB is null or not of type javax.ejb.Handle in removeEJB**
	Explanation: The EJB passed into a call to pub.ejbadapter.removeEJB is null or not the correct type. This error indicates a defect in the client code invoking this service.
	Action: Correct the client code.
5017	**Input WmRecord object is null**
	Explanation: The input record passed to an adapter service instance is null. This error indicates a defect in the client code invoking this service.
	Action: Correct the client code.
5018	**EJBDescriptorCache is null or empty**
	Explanation: The adapter failed to locate any EJBs registered in the JNDI server. This error could be due to a number of causes.
	Action: Ensure that EJBs are configured and deployed properly on the application server and that they are accessible to Adapter for EJB.
5019	**EJBClassDescriptor is null**
	Explanation: This error indicates an internal logic defect in the adapter code.
	Action: Contact Software AG Global Support.
5020	**EJBMethodDescriptor is null**

Error Code	Description
	Explanation: This indicates an internal logic defect in the adapter code.
	Action: Contact Software AG Global Support.
5021	**Unsupported object of type {0} returned by EJB home method**
	Explanation: The adapter successfully invoked a remote home method on an EJB, but the object returned by that method is of the specified type, that is not supported. The cause is indeterminate.
	Action: Contact Software AG Global Support.
5022	**Invalid supported connection class found in impl directory: {0}**
	Explanation: A class file for a class that is not a subclass of com.wm.adapter.wmejb.connection.WmEJBConnection was found in the following adapter directory: WmEJBAdapter/code/classes/com/wm/adapter/wmejb/connection/impl/supported
	Action: Move the specified class file into another directory to prevent this message from displaying.
5023	**Cannot execute method {0}; Handle is null**
	Explanation: An instance of InvokeEJB 2.1 or CreateInvokeEJB 2.1 cannot execute the stated remote bean method because the Handle object it uses to get the EJBObject instance is null. For an InvokeEJB 2.1 service, the handle is passed in from the client code. For a CreateInvokeEJB 2.1 service, the handle instance is generated internally.
	Action: If this error occurs when your client code is calling InvokeEJB 2.1, ensure that you are passing a valid javax.ejb.Handle object to the service.
5024	**Remove not allowed on given EJB**
	Explanation: The standalone service pub.ejbadapter.removeEJB caught a javax.ejb.Remove exception when calling the method java.ejb.EJBObject.remove() on an EJB. The EJB container has determined that the EJB cannot be removed at this time, which may or may not be the expected behavior.
	Action: Check with the EJB deployer to ensure that it is configured to allow removal. In particular, stateful EJBs running within a transaction may be subject to containerimposed restrictions on their removal. Alternatively, omit the call to removeEJB in your client or trap the error and branch accordingly.
5025	**FetchEJB has failed to look-up resource domain: {0} with dependency value: {1}. This can happen if the service is using an EJB that no longer exists or has been modified.**

Error Code	Description
	Explanation: The EJB that the adapter service has been configured to use does not match the EJB deployed on the server or the EJB is no longer deployed. This error message may display multiple times. (FetchEJB is the name of the underlying adapter service template class for a CreateEJB 2.1 service.)
	Action: If the EJB no longer exists, delete this adapter service instance. If the EJB has changed, re-configure this adapter service instance.
5026	**InvokeEJB has failed to look-up resource domain: {0} with dependency value: {1}. This can happen if the service is using an EJB that no longer exists or has been modified.**
	Explanation: The EJB that the adapter service has been configured to use does not match the EJB deployed on the server or the EJB is no longer deployed. This error message may display multiple times. (InvokeEJB is the name of the underlying adapter service template class for an InvokeEJB 2.1 service.)
	Action: If the EJB no longer exists, delete this adapter service instance. If the EJB has changed, re-configure this adapter service instance.
5027	**FetchAndInvokeEJB has failed to look-up resource domain: {0} with dependency value: {1}. This can happen if the service is using an EJB that no longer exists or has been modified.**
	Explanation: The EJB that the adapter service has been configured to use does not match the EJB deployed on the server or the EJB is no longer deployed. This error message may display multiple times. (FetchandInvokeEJB is the name of the underlying adapter service template class for a CreateInvokeEJB 2.1 service.)
	Action: If the EJB no longer exists, delete this adapter service instance. If the EJB has changed, re-configure this adapter service instance.
5028	**The EJB object passed into InvokeEJB does not match the expected type: {0}**
	Explanation: The Invoke Adapter for EJB service you are running was configured for an EJB of the stated class type. However, an EJB of a different type was passed into the service at run time. (InvokeEJB is the name of the underlying adapter service template class for an InvokeEJB 2.1 service.)
	Action: Edit the flow service to map the correct EJB type to the 'EJB' input parameter of the configured Invoke Adapter for EJB service.
5029	**The parameter: {0} in the service you are running is a Java primitive type--it cannot be null.**

Error Code	Description
	Explanation: The service you are running attempts to call a remote EJB method with a primitive argument for which no value was provided.
	Action: Edit the service to ensure that all primitive method parameters have an appropriate value. Consult the EJB you are running against to determine what values might be appropriate.
5030	**Input record to InvokeEJB30.execute() is empty**
	Explanation: The input record passed into this adapter service instance at runtime contained no data. This error indicates a defect in the client code that is calling this service. (InvokeEJB30 is the name of the underlying adapter service template class for an InvokeEJB 3.0 service.)
	Action: Ensure that the client service code is implemented correctly. At a minimum, all calls to InvokeEJB30.execute must contain an EJB Handle instance in the input pipeline.
5031	**InvokeEJB30 has failed to look-up resource domain: {0} with dependency value: {1}. This can happen if the service is using an EJB that no longer exists or has been modified.**
	Explanation: The EJB that the adapter service has been configured to use does not match the EJB deployed on the server or the EJB is no longer deployed. This error message may display multiple times. (InvokeEJB30 is the name of the underlying adapter service template class for an InvokeEJB 3.0 service.)
	Action: If the EJB no longer exists, delete this adapter service instance. If the EJB has changed, re-configure this adapter service instance.
5032	**FetchAndInvokeEJB30 has failed to look-up resource domain: {0} with dependency value: {1}. This can happen if the service is using an EJB that no longer exists or has been modified.**
	Explanation: The EJB that the adapter service has been configured to use does not match the EJB deployed on the server or the EJB is no longer deployed. This error message may display multiple times. (FetchandInvokeEJB30 is the name of the underlying adapter service template class for a FetchInvokeEJB 3.0 service.)
	Action: If the EJB no longer exists, delete this adapter service instance. If the EJB has changed, re-configure this adapter service instance.
5033	**The EJB object passed into InvokeEJB30 does not match the expected type: {0}**
	Explanation: The InvokeEJB30 adapter service you are running was configured for an EJB of the stated class type. However, an EJB of a different type was passed into the service at run time. (InvokeEJB30 is the

Error Code	Description
	name of the underlying adapter service template class for an InvokeEJB 3.0 service.)
	Action: Edit the flow service to map the correct EJB type to the 'EJB' input parameter of the configured InvokeEJB30 adapter service.
5034	**FetchEJB30 has failed to look-up resource domain: {0} with dependency value: {1}. This can happen if the service is using an EJB that no longer exists or has been modified.**
	Explanation: The EJB that the adapter service has been configured to use does not match the EJB deployed on the server or the EJB is no longer deployed. This error message may display multiple times. (FetchEJB30 is the name of the underlying adapter service template class for a FetchEJB 3.0 service.)
	Action: If the EJB no longer exists, delete this adapter service instance. If the EJB has changed, re-configure this adapter service instance.
5998	**java.rmi.RemoteException explanation: {0}**
	Explanation: This is a supplementary message that is logged when a RemoteException has occurred. This message logs the output of RemoteException.getMessage().
	Action: This is a generic log message that may be caused by any of several RMI-related events. Ensure that the problem is not due to the application server EJB configuration. Contact Software AG Global Support.
5999	**java.rmi.RemoteException root cause: {0}**
	Explanation: This is a supplementary message that is logged when a RemoteException has occurred. This message logs the output of RemoteException.detail.toString().
	Action: This is a generic log message that may be caused by any of several RMI-related events. Ensure that the problem is not due to the application server EJB configuration.

A Scenarios

Overview

This appendix presents some scenarios for typical applications that use Adapter for Enterprise Javabeans. Depending on the complexity of your business function, you could implement one scenario or combine elements of several scenarios to create the necessary functionality.

The scenarios are intended to help you use Adapter for Enterprise Javabeans to create EJB client services and join them together in logical workflows to implement some desired business function.

While considering the scenarios, keep the following points in mind:

- Adapter for Enterprise Javabeans supports session and entity beans only. There are no scenarios for message-driven EJBs.

  Note:
  Entity beans are not supported by the EJB 3.0 standard. They are supported only by EJB 2.1 or earlier standards.

- The scenarios illustrate basic event flow with successful results only. No error or exception handling is illustrated.
- The scenarios assume that you are an authorized webMethods user and are familiar with Software AG Designer and in particular, the Flow Service Editor. For information about creating flow services, see the *webMethods Service Development Help* for your release.

For additional information to help you create flow services for use with Adapter for Enterprise Javabeans, see "Creating Flows for Adapter for Enterprise Javabeans Services" on page 147.

Note:
The CreateEJB 2.1, InvokeEJB 2.1, and CreateInvokeEJB 2.1 adapter services are used to access the 2.1 or earlier versions of EJBs. The FetchEJB 3.0, InvokeEJB 3.0, and FetchInvokeEJB 3.0 adapter services are used to access the 3.0 EJBs. The 2.1 adapter services cannot be used with the 3.0 adapter services and vice versa.

Running a Single Method on a Single Bean

This scenario illustrates how to invoke a single method on a single remote EJB and view the results.

Assumptions

In the steps below, the following assumptions are made:

- An adapter service has been configured from the CreateInvokeEJB 2.1 or the FetchInvokeEJB 3.0 adapter service template for the desired EJB, including the method used for creating the bean and the bean method to invoke.
- The associated adapter connection is enabled.
- You provide all method input parameters at runtime.
- All method input parameters and return values have types that are recognizable in Designer.

Steps

1. Using Designer, create a flow that calls the CreateInvokeEJB 2.1 or the FetchInvokeEJB 3.0 adapter service instance.

2. Execute the flow and enter the appropriate method inputs when prompted.

3. Observe the results in the service's output:

   - A successful status for the create or fetch service invocation
   - A successful status for the bean method invocation
   - The results of the method invocation

4. For information about the CreateInvokeEJB 2.1 adapter service, see "CreateInvokeEJB 2.1 Adapter Service" on page 34. For information about the FetchInvokeEJB 3.0 adapter service, see "FetchInvokeEJB 3.0 Adapter Service" on page 39.

Running Multiple Independent Methods on a Single Bean

This scenario illustrates how to invoke two or more methods on the same remote EJB and view the results. The methods have no dependencies on one another.

Assumptions

- An adapter service has been configured from the CreateEJB 2.1 or the FetchEJB 3.0 adapter service template for the desired EJB, including the method to use for creating the bean.
- Separate adapter services have been configured from the corresponding InvokeEJB 2.1 or the InvokeEJB 3.0 adapter service template for the desired EJB and each of the desired bean methods.
- You will provide all create and bean method input parameters at runtime.
- All method input parameters and return values have types that are recognizable in Designer.

Steps

1. Using Designer, create a flow consisting of the following adapter service calls:

   a. CreateEJB 2.1 or FetchEJB 3.0

   b. Each of the corresponding InvokeEJB 2.1 or InvokeEJB 3.0 services you want to execute (two or more)

   c. RemoveEJB (only for 2.1 services)

2. Edit the flow to map the output of CreateEJB 2.1 or FetchEJB 3.0 (which is the bean object) to the corresponding input bean object in each of the corresponding InvokeEJB 2.1 or InvokeEJB 3.0 calls and the RemoveEJB call.

   Note:
   RemoveEJB service is used only for 2.1 services.

3. Execute the flow and enter the appropriate method inputs when prompted.

4. Observe the results in the flow's output:

   - A successful status for the EJB create or fetch service invocation
   - A successful status for each of the bean method invocations
   - If successful, RemoveEJB returns nothing
   - The results of each bean method invocation

5. For information about the CreateEJB 2.1, FetchEJB 3.0, InvokeEJB 2.1, and InvokeEJB 3.0 adapter services, see “Adapter Services” on page 25. For information about the RemoveEJB service, see “Removing EJBs” on page 93.

Running Multiple Dependent Methods on a Single Bean

This scenario illustrates how to invoke two or more methods on the same remote EJB and view the results. Note that in one or more cases, the input parameter(s) passed to a method are the output return value(s) from one or more methods run previously.

Assumptions

- An adapter service has been configured from the CreateEJB 2.1 or FetchEJB 3.0 adapter service template for the desired EJB, including the method to use for creating the bean.
- Separate adapter services have been configured from the corresponding InvokeEJB 2.1 or InvokeEJB 3.0 adapter service template for the desired EJB and each of the desired bean methods.
- One or more of the bean methods will get some or all of their input parameters from the output return value(s) of methods invoked previously in the flow. At runtime, you will be prompted to supply all other method inputs.
- Any input parameters and/or return values that are not to be mapped as described above must have types that are recognizable in Designer.

Steps

1. Using Designer, create a flow consisting of the following adapter service calls:

a. CreateEJB 2.1 or FetchEJB 3.0

b. Each of the corresponding InvokeEJB 2.1 or InvokeEJB 3.0 services you want to execute (two or more)

c. RemoveEJB (only for 2.1 services)

2. Edit the flow to map the output of CreateEJB 2.1 or FetchEJB 3.0 (which is the bean object) to the corresponding input bean object in each of the corresponding InvokeEJB 2.1 or InvokeEJB 3.0 calls and the RemoveEJB call (only for 2.1 services).

3. Edit the flow to map the output of one or more InvokeEJB 2.1 or InvokeEJB 3.0 calls to the input(s) of one or more subsequent InvokeEJB 2.1 or InvokeEJB 3.0 calls.

4. Execute the flow and enter the appropriate method inputs (if any) when prompted.

5. Observe the results in the flow's output:

   - A successful status for the EJB create or fetch service invocation
   - A successful status for each of the bean method invocations
   - The results of each bean method invocation

6. For information about the CreateEJB 2.1, FetchEJB 3.0, InvokeEJB 2.1, and InvokeEJB 3.0 adapter services, see "Adapter Services" on page 25. For information about the RemoveEJB service, see "Removing EJBs" on page 93.

Running a Single Method on Multiple Beans of the Same Type

This scenario illustrates how to retrieve multiple instances of the same EJB class and then invoke the same bean method on each instance. This scenario presents two ways to accomplish the operation:

Alternative 1

This scenario uses the CreateEJB 2.1, InvokeEJB 2.1, and RemoveEJB services together in a flow.

Assumptions

- An adapter service has been configured from the CreateEJB 2.1 adapter service template for the desired EJB. The EJB must be an entity bean and its home interface must expose a method that returns multiple instances of the remote EJB. The CreateEJB 2.1 service has been configured to call this method.
- An adapter service has been configured from the InvokeEJB 2.1 adapter service template for the desired EJB to call one of its public methods.

- You will provide all create and bean method input parameters at runtime.
- The same input parameter value(s) will be passed into each bean invocation method.
- All input parameters and/or return values must have types that are recognizable in Designer.

Steps

1. Using Designer, create a flow consisting of the following adapter service calls:

    a. CreateEJB 2.1 (to retrieve the EJB instances)

    b. InvokeEJB 2.1 (the method to run on each instance)

    c. RemoveEJB (to release each of the bean instances)

2. Edit the flow to wrap the InvokeEJB 2.1 and RemoveEJB steps in a loop that will execute once for each EJB instance returned by the CreateEJB 2.1 step.

3. Edit the flow to map the current EJB instance (a bean object) in the loop to the corresponding input parameter of the InvokeEJB 2.1 and RemoveEJB steps.

4. Execute the flow and enter the appropriate method inputs (if any) when prompted.

5. Observe the results in the flow's output:

    - A successful status for the EJB create method invocation
    - A successful status for each of the bean method invocations
    - The results of each bean method invocation

6. For information about the CreateEJB 2.1 and InvokeEJB 2.1 adapter services, see "Adapter Services" on page 25. For information about the RemoveEJB service, see "Removing EJBs" on page 93.

Alternative 2

This scenario uses the CreateInvokeEJB 2.1 service, which combines all services in a single step.

Assumptions

- An adapter service has been configured from the CreateInvokeEJB 2.1 adapter service template for the desired EJB. The EJB must be an entity bean and its home interface must expose a create method that returns multiple instances of the remote EJB. The CreateInvokeEJB 2.1 service has been configured to call this method. The service has also been configured to call one of the remote EJB's methods.

- You will provide all create and bean method input parameters at runtime.
- All input parameters and/or return values must have types that are recognizable in Designer.
- The same input parameter values will be passed into each bean invocation method.

Steps

1. Using Designer, execute the CreateInvokeEJB 2.1 adapter service and enter the appropriate method inputs when prompted.

2. Observe the results in the service's output:

   - A successful status for the EJB create method invocation
   - A successful status for each of the bean method invocations
   - The results of each bean method invocation

3. For information about the CreateInvokeEJB 2.1 service, see "CreateInvokeEJB 2.1 Adapter Service" on page 34.

Running Multiple Methods on Multiple Beans

This scenario illustrates how to run one or more methods on two or more different EJB objects. The EJB objects are not instances of the same class.

Assumptions

- An adapter service has been configured from the CreateEJB 2.1 or FetchEJB 3.0 adapter service template for the desired EJBs, including the method to use for creating the bean. At least two such services, each creating different EJBs, should be configured.
- An adapter service has been configured from the corresponding InvokeEJB 2.1 or InvokeEJB 3.0 adapter service template for each of the desired method invocations.
- You will provide all create and bean method input parameters at runtime.
- All input parameters and/or return values must have types that are recognizable in Designer.

Steps

1. Using Designer, create a flow consisting of the following adapter service calls:

   a. CreateEJB 2.1 or FetchEJB 3.0 (one for each of the EJBs to be created or fetched)

   b. Corresponding InvokeEJB 2.1 or InvokeEJB 3.0 (one for each of the distinct method calls)

   c. RemoveEJB (one for each EJB created, only for 2.1 services)

2. Edit the flow to map the output of the CreateEJB 2.1 or FetchEJB 3.0 (the bean objects) calls to the corresponding input bean object in each of the corresponding InvokeEJB 2.1 or InvokeEJB 3.0 calls and the RemoveEJB calls (only for 2.1 services).

   Important:
   Be careful when naming instances of these CreateEJB 2.1 or FetchEJB 3.0 and InvokeEJB 2.1 or InvokeEJB 3.0 adapter services to avoid mismatching them. If a pair of CreateEJB 2.1/InvokeEJB 2.1 or FetchEJB 3.0/InvokeEJB 3.0 services are mismatched in a flow, this condition will not be detected until that flow is executed. For more information, see "InvokeEJB 2.1 Adapter Service" on page 32 and "InvokeEJB 3.0 Adapter Service" on page 37.

3. Execute the flow and enter the appropriate method inputs (if any) when prompted.

4. Observe the results in the flow's output:

   - A successful status for each EJB create or fetch service invocation
   - A successful status for each of the bean method invocations
   - The results of each bean method invocation

5. For information about the CreateEJB 2.1, FetchEJB 3.0, InvokeEJB 2.1, and InvokeEJB 3.0 adapter services, see "Adapter Services" on page 25. For information about the RemoveEJB service, see "Removing EJBs" on page 93.

Running a Single Method with Complex Input on a Single Bean

This scenario provides two approaches to invoking an EJB method to accommodate complex inputs. Complex input in this scenario is any non-native Java class. Its contents are arbitrary, however, the class must implement the java.io.Serializable interface.

Approach 1

This approach illustrates how to invoke an EJB method that requires an instance of a user-defined object as one of its input parameters.

In a flow service you can map an instance of an arbitrary third-party object to a parameter defined in the input pipeline of an adapter service. Depending on the EJB being used, this object can be passed to one of its home methods or one of its bean methods. In this approach, it is the bean method (that is, the method defined in the **Method to Invoke** tab) that will receive the custom object.

Assumptions

- An EJB exists on the application server that takes a user-defined object as one of its method parameters.
- An adapter service has been configured from the CreateInvokeEJB 2.1 or FetchInvokeEJB 3.0 adapter service template for the desired EJB, including the method used for creating the bean and the bean method to invoke (that is, the method that requires the non-standard object).
- You will provide all method input parameters at runtime.
- Except as noted above, all other method input parameters and return values have types that are recognizable in Designer.

Steps

1. Create a native Java service to create the user-defined object that will be passed to the EJB method. This Java service should expect on its input pipeline the property values needed to successfully create and initialize the user-defined object, which it should insert into its outbound pipeline.
2. Create a flow that includes the following steps:
   a. Calls the Java service created in step 1.
   b. Calls the configured CreateInvokeEJB 2.1 or FetchInvokeEJB 3.0 adapter service described in the assumptions above.
3. Edit the flow to map the output of the Java service to the input of the EJB bean method called in the corresponding CreateInvokeEJB 2.1 or FetchInvokeEJB 3.0 adapter service.
4. Execute the flow and provide the appropriate input values (if any) when prompted for all other EJB method parameters.
5. Observe the results in the flow's output:
   - A successful status for the create method invocation
   - A successful status for the bean method invocation
   - The results of the method invocation
6. For information about the CreateInvokeEJB 2.1 service, see "CreateInvokeEJB 2.1 Adapter Service" on page 34. For information about the FetchInvokeEJB 3.0 service, see "FetchInvokeEJB 3.0 Adapter Service" on page 39.

Approach 2

This approach illustrates how to invoke a method on an EJB that requires an instance of another EJB as one of its input parameters.

Assumptions

- An EJB exists on the application server that takes, as one of its method parameters, an instance of another EJB.
- An adapter service has been configured from the CreateInvokeEJB 2.1 or FetchInvokeEJB 3.0 adapter service template for the desired EJB, including the method used for creating the bean and the bean method to invoke (that is, the method that requires the other EJB).
- An adapter service has been configured from the corresponding CreateEJB 2.1 or FetchEJB 3.0 adapter service template to create the second EJB.
- You will provide all method input parameters at runtime.
- Except as noted above, all other method input parameters and return values have types that are recognizable in Designer.

Steps

1. Using Designer, create a flow consisting of the following adapter service calls:

   a. CreateEJB 2.1 or FetchEJB 3.0 (to create the EJB to be passed to the other EJB)

   b. Corresponding CreateInvokeEJB 2.1 or FetchInvokeEJB 3.0 (to create the main EJB and run the method that takes the EJB created in the previous step as one of its parameters)

   c. RemoveEJB (to release the EJB created by CreateEJB 2.1)

2. Edit the flow to map the output of the corresponding CreateEJB 2.1 or FetchEJB 3.0 service from step a to the input of the EJB method called in the corresponding CreateInvokeEJB 2.1 or FetchInvokeEJB 3.0 service.

3. Execute the flow and provide the appropriate input values (if any) when prompted for all other EJB method parameters.

4. Observe the results in the flow's output:

   - A successful status for the create method invocation of the first EJB
   - A successful status for the create method invocation of the second EJB
   - The results of the method invocation

B Built-In Transaction Management Services

Transaction Management Overview

This appendix provides an overview and examples of using transactions. It describes how Integration Server supports the built-in services used to manage explicit transactions for your Adapter for Enterprise Javabeans services in the WmART package. For descriptions of each of the specific built-in transaction management services that can be used with the WmART package, see "Built-In Transaction Management Services" on page 142.

For information about other built-in services available for use with Adapter for Enterprise Javabeans, see the *webMethods Integration Server Built-In Services Reference* for your release.

Transactions

Integration Server considers a transaction to be one or more interactions with one or more resources that are treated as a single logical unit of work (LUW). The interactions within a transaction are either all committed or all rolled back. For example, if a transaction includes one or more calls to Adapter for Enterprise Javabeans services and one of these services fails, all other services in the transaction are rolled back.

Transaction Types

Adapter for Enterprise Javabeans supports local transactions with all supported application servers, without other resource participants. Adapter for Enterprise Javabeans also supports XA transactions only on WebLogic Server 10.3. For a description of the transaction types supported by the Adapter for Enterprise Javabeans, see "Transaction Management of Adapter for Enterprise Javabeans Connections" on page 19.

In general, the Adapter for Enterprise Javabeans can communicate with multiple application servers (resources) at a time provided that those application servers are the same vendor and version. There are limitations when the adapter tries to access multiple resources using locally transacted connections within a single flow service.

Note:
If a transaction accesses multiple resources, and more than one of the resources only supports local transactions, the integrity of the transaction cannot be guaranteed. For example, if the first resource successfully commits, and the second resource fails to commit, the first resource interaction cannot be rolled back; it has already been committed. To help prevent this problem, Integration Server detects this case when connecting to more than one resource that does not support two-phase commits. It throws a run-time exception and the service execution fails.

Also remember that the application server, the EJB container, and the EJB itself may impose their own set of restrictions on how an EJB may be used in a transaction.

Implicit and Explicit Transactions

Implicit transactions are handled automatically by Integration Server's transaction manager. Implicit transaction support is enabled when you configure an adapter service to use an EJB Local Connection or an EJB XA Connection and invoke this adapter service from a flow. When you define an explicit transaction, you define the start-on-completion boundaries of the transaction. As such, implicit and explicit transactions need to be created and managed differently.

The following sections describe implicit and explicit transactions and how to manage them.

Implicit Transactions

With implicit transactions, Integration Server automatically manages local transactions without requiring you to explicitly do anything. That is, Integration Server starts and completes an implicit transaction with no additional service calls required by the adapter user.

A transaction context, which the transaction manager uses to define a unit of work, starts when a flow service executes an adapter service. The connection required by the adapter service is registered with the newly created context and used by the adapter service. If the flow executes another adapter service, the transaction context is searched to see if the connection is registered already. If the connection is already registered, the adapter service uses this connection. If the connection is not registered, Integration Server retrieves a new connection instance and registers it with the transaction.

Note that if the top level flow invokes another flow, adapter services in the child flow use the same transaction context.

When the top level flow completes, the transaction completes and either is committed or rolled back, depending on the status (success or failure) of the top level flow.

A single transaction context can contain no more than one EJB Local Connection connection. If your flow contains adapter services that use more than one EJB Local Connection connection, you must use explicit transactions, which are described in the next section.

For more information about designing and using flows, see the *webMethods Service Development Help* for your release.

For more information about transaction types, see "Transaction Management of Adapter for Enterprise Javabeans Connections" on page 19.

Explicit Transactions

You use explicit transactions when you need to explicitly control the transactional units of work. To do this, you use additional services, known as built-in services, in your flow.

A transaction context starts when the pub.art.transaction.startTransaction() service executes. The transaction context completes when either the pub.art.transaction.commitTransaction() or pub.art.transaction.rollbackTransaction() service executes. As with implicit transactions, a single transaction context can contain no more than one EJB Local Connection connection.

Note:
With explicit transactions, you must be sure to call either a commitTransaction() or rollbackTransaction() for each startTransaction() service, or you will have dangling transactions, which will require you to reboot Integration Server.

A new explicit transaction context can be started within an existing transaction context, provided that you ensure that the transactions are committed in the reverse order they were started-that is, the last transaction to start should be the first transaction to complete, and so forth.

For example, consider the following is a valid construct:

```
pub.art.transaction.startTransaction()
pub.art.transaction.startTransaction()
pub.art.transaction.startTransaction()
pub.art.transaction.commitTransaction()
pub.art.transaction.commitTransaction()
pub.art.transaction.commitTransaction()
```

The following example shows an *invalid* construct:

```
pub.art.transaction.startTransaction()
pub.art.transaction.startTransaction()
pub.art.transaction.commitTransaction()
pub.art.transaction.commitTransaction()
```

For more information about designing and using flows, see the *webMethods Service Development Help* for your release.

For more information about transaction types, see "Transaction Management of Adapter for Enterprise Javabeans Connections" on page 19.

Built-In Transaction Management Services

The following sections describe each of the built-in services you can use with the WmART package.

pub.art.transaction:commitTransaction

This service commits an explicit transaction. It must be used in conjunction with the pub.art.transaction:startTransaction service. If it does not have a corresponding pub.art.transaction:startTransaction service, your flow service will receive a runtime error.

For more information about implicit and explicit transactions, see "Overview" on page 148.

Input Parameters

commitTransactionInput	**Document.**A document that contains the variable *transactionName*, described below.
transactionName	**String.**Used to associate a name with an explicit transaction. The *transactionName* must correspond to the *transactionName* in any pub.art.transaction:startTransaction or pub.art.transaction:rollbackTransaction services associated with the explicit transaction.

	This value must be mapped from the most recent pub.art.transaction:startTransaction that has not previously been committed or rolled back.

Output Parameters

None.

pub.art.transaction:rollbackTransaction

This service rolls back an explicit transaction. It must be used in conjunction with the pub.art.transaction:startTransaction service. If it does not have a corresponding pub.art.transaction:startTransaction service, your flow service will receive a runtime error.

For more information about implicit and explicit transactions, see "Overview" on page 148.

Input Parameters

rollbackTransactionInput	**Document.** A document that contains the variable *transactionName*, described below.
transactionName	**String.** Used to associate a name with an explicit transaction. The *transactionName* must correspond to the *transactionName* in any WmART.pub.art.transaction:startTransaction or WmART.pub.art.transaction:commitTransaction services associated with the explicit transaction. This value must be mapped from the most recent pub.art.transaction:startTransaction that has not previously been committed or rolled back.

Output Parameters

None.

pub.art.transaction:setTransactionTimeout

This service enables you to manually set a transaction timeout interval for implicit and explicit transactions. When you use this service, you are temporarily overriding Integration Server's transaction timeout interval. To change the server's default transaction timeout, see "Changing Integration Server's Transaction Timeout Interval" on page 145.

You must call this service within a flow before the start of any implicit or explicit transactions. Implicit transactions start when you call an adapter service in a flow. Explicit transactions start when you call the pub.art.transaction:startTransaction service.

If the execution of a transaction takes longer than the transaction timeout interval, all current executions associated with the flow are cancelled and rolled back if necessary.

This service overrides only the transaction timeout interval for the flow service in which you call it.

Input Parameters

timeoutSeconds	**Integer.** The number of seconds that the implicit or explicit transaction stays open before the transaction manager aborts it.

Output Parameters

None.

pub.art.transaction.startTransaction

This service starts an explicit transaction. It must be used in conjunction with either a pub.art.transaction:commitTransaction service or pub.art.transaction:rollbackTransaction service. If it does not have a corresponding pub.art.transaction:commitTransaction service orpub.art.transaction:rollbackTransaction service, your flow service will receive a runtime error.

For more information about implicit and explicit transactions, see "Overview" on page 148.

Input Parameters

startTransactionInput	**Document.** A document that contains the variable *transactionName*, described below.
transactionName	**String.** Specifies the name of the transaction to be started. This parameter is optional. If you leave this parameter blank, Integration Server will generate a name for you. In most implementations, it is not necessary to provide your own *transactionName* as input.

Output Parameters

startTransactionOutput	**Document.** A document that contains the variable *transactionName*, described below.
transactionName	**String.** The name of the transaction the service just started.

Changing Integration Server's Transaction Timeout Interval

Integration Server's default transaction timeout is no timeout (NO_TIMEOUT). To change the server's transaction timeout interval, use a text editor to modify the server.cnf file and add the parameter below. Note that this parameter does not exist by default in the server.cnf file; you must add it to the file as described below.

Be sure to shut down Integration Server before you edit this file. After you make changes, restart the server.

Add the following parameter to the server.cnf file:

`watt.art.tmgr.timeout=`*TransactionTimeout*

where *TransactionTimeout* is the number of seconds before transaction timeout.

This transaction timeout parameter does not halt the execution of a flow; it is the maximum number of seconds that a transaction can remain open and still be considered valid. For example, if your current transaction has a timeout value of 60 seconds and your flow takes 120 seconds to complete, the transaction manager will rollback all registered operations regardless of the execution status.

For more information about adding parameters to the server.cnf file, see the *webMethods Integration Server Administrator’s Guide* for your release.

Transaction Error Situations

When Integration Server encounters a situation that could compromise transactional integrity, it throws an error. Such situations include the following:

- A transaction includes two or more different resources that only support local transactions.

  If a transaction accesses multiple resources, and more than one of the resources supports only local transactions, the integrity of the transaction cannot be guaranteed. For example, if the first resource commits successfully, and the second resource fails to commit, the first resource interaction cannot be rolled back; it has been committed already. To help prevent this problem, Integration Server detects this case when connecting to more than one resource that does not support two-phase commits. It throws a run-time exception and the service execution fails.

  Note:
  Because this situation may be acceptable in some applications, the adapter user can include an input in the startTransaction service to cause Integration Server to allow this situation.

- A resource is used in both a parent transaction and a nested transaction.

  This situation is ambiguous, and most likely means that a nested transaction was not closed properly.

- A parent transaction is closed before its nested transaction.

After a service request has invoked all its services, but before returning results to the caller, the service may commit its work. This commit could fail if the resource is unavailable or rejects the commit. This will cause the entire server request to fail and to roll back the transaction.

C Creating Flows for Adapter for Enterprise Javabeans Services

Overview

This appendix explains how to build flow services in Designer that use adapter services created with Adapter for Enterprise Javabeans. For more information about working with flow services, see the *webMethods Service Development Help* for your release.

Note:
The examples shown assume that the class files for deployed EJBs are located on Designer's classpath. If they are not, Designer will incorrectly display the value of any object it cannot de-serialize as “null” in the run-time results. This does not prevent the flow service from working properly. However, it can cause confusion for the user. To avoid this situation, make sure that the class files for any relevant objects (EJB or otherwise) are available to Designer.

About Flow Services and Adapter for Enterprise Javabeans

Adapter for Enterprise Javabeans enables you to configure adapter services that create or fetch EJB instances on the application server. You can then create adapter services that execute a public method exposed by an EJB. This model implies a logical sequence, or flow of events:

- Execute the adapter service to create or fetch the EJB.
- Then execute the adapter service that, in turn, invokes one of the EJB's business methods.

Standalone Adapter for Enterprise Javabeans services have little value. Their value is only realized when the services are combined. There are several ways to do this on webMethods Integration Server, but the most common way is to combine the two operations in a single flow service. Using Designer’s flow service editor you can link or *map* the inputs and outputs of any individual step in a flow service to other steps in the same flow service. This function is important for the following reasons:

- Any execution of an InvokeEJB 2.1 or InvokeEJB 3.0 service must be preceded by an execution of a corresponding CreateEJB 2.1 or FetchEJB 3.0 service.
- Any execution of an InvokeEJB 2.1 or InvokeEJB 3.0 service always requires at least one input: the output of a prior CreateEJB 2.1 or FetchEJB 3.0 execution. That is, before you can execute an EJB method, you must first obtain that EJB.

  Note:
  The CreateEJB 2.1, InvokeEJB 2.1, and CreateInvokeEJB 2.1 adapter services are used to access the 2.1 or earlier versions of EJBs. The FetchEJB 3.0, InvokeEJB 3.0, and FetchInvokeEJB 3.0 adapter services are used to access the 3.0 EJBs. The 2.1 adapter services cannot be used with the 3.0 adapter services and vice versa. For more information about the adapter’s services, see “Adapter Services” on page 25.

Obtaining an EJB

To obtain an EJB object, you can use either a CreateEJB 2.1 or FetchEJB 3.0 adapter service or a CreateInvokeEJB 2.1 or FetchInvokeEJB 3.0 adapter service. The EJB object returned can be a single EJB object, or multiple EJB objects. Multiple EJB objects are all of the same type.

The following sections explain how to:

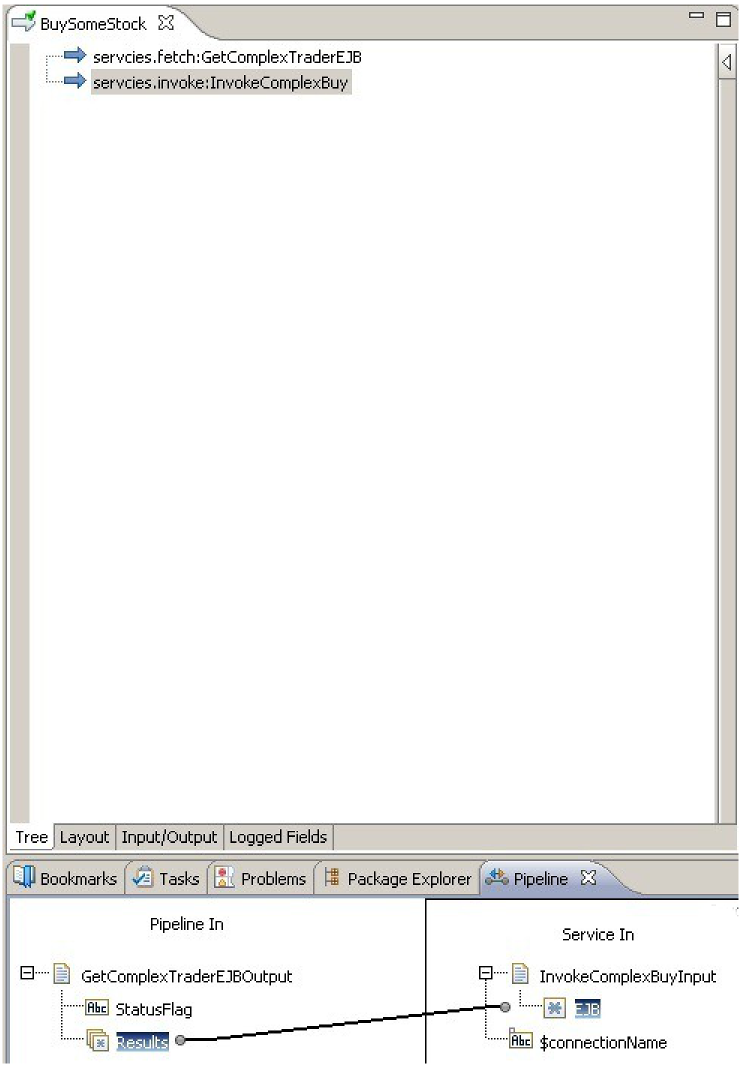
- Pass a single EJB object generated by calling a configured CreateEJB 2.1 or FetchEJB 3.0 adapter service instance to a corresponding configured InvokeEJB 2.1 or InvokeEJB 3.0 adapter service instance.
- Use a loop to pass a single EJB object generated when multiple EJBs are returned by CreateEJB 2.1 or FetchEJB 3.0.
- Use services to pass third-party or user-defined objects.

Working with a Single EJB Object Instance

To obtain an EJB object, use a CreateEJB 2.1 or FetchEJB 3.0 adapter service. Then in a flow service in Designer you use the pipeline to pass the EJB obtained by a CreateEJB 2.1 or FetchEJB 3.0 execution to a subsequent InvokeEJB 2.1 or InvokeEJB 3.0 instance. By editing the inbound pipeline of the corresponding InvokeEJB 2.1 or InvokeEJB 3.0 service, you can map the *Results* array of the CreateEJB 2.1 or FetchEJB 3.0 instance to the *EJB* parameter of the corresponding InvokeEJB 2.1 or InvokeEJB 3.0 instance.

For example, a CreateEJB 2.1 or FetchEJB 3.0 adapter service, services.fetch.GetComplexTraderEJB, creates a single EJB instance and returns it as the first element of the *Results* array, which Designer places in the pipeline.

To then pass this value to the corresponding InvokeEJB 2.1 or InvokeEJB 3.0 adapter service, services.invoke.InvokeComplexBuy, insert a link from the *Results* array in the pipeline to the *EJB* parameter in the corresponding InvokeEJB 2.1's or InvokeEJB 3.0's input signature as shown in the figure below.



To insert the link, click ⊶ on the input side of the Pipeline view. At run time, the link tells Integration Server to assign the values in *Results* to *EJB*.

However, because the input is an array and the target destination parameter is a scalar, you need to indicate which array value to assign. To do this, open the Link Indices dialog box by selecting the link between the variables and clicking ▦ on the Pipeline view toolbar.

The Link Indices dialog box appears indicating all the parameters at each end of the selected link. Use the **Results** parameter to specify which element of *Results* to assign. Note that the services.fetch.GetComplexTraderEJB service will always return exactly one EJB instance, placing the EJB instance in the very first element of the *Results* array. Therefore in this example, you would enter the value of “0” into the **Results** parameter.

For more information about mapping data in flow services, see the *webMethods Service Development Help* for your release.

Working with Multiple EJB Object Instances

To build on the example in “Working with a Single EJB Object Instance” on page 149, suppose that instead of returning a single EJB in the *Results* array, the CreateEJB 2.1 or FetchEJB 3.0 adapter service instance returns multiple EJB instances. In this case you use the LOOP step in Designer to make the flow service iterate over the contents of the array, and invoke one (or more) methods on each EJB in the array.

Note:
For information about working with loops and the logic involved, see the *webMethods Service Development Help* for your release.

Any given instance of a CreateEJB 2.1 adapter service will invoke a single method on the javax.ejb.EJBHome interface. This method is called to construct an instance of the actual EJB remote class and return it to the caller. This remote EJB instance is the EJB and is where the business methods of the EJB are found.

It is possible for a home interface method to return multiple remote EJB objects (“finder” methods on entity beans, for example). The CreateEJB 2.1 adapter service accounts for this with the *Results* array.

Similarly, for any given instance of a FetchEJB 3.0 adapter service, the service fetches the EJB based on the selected JNDI Name and returns the EJB Object to the caller.

The same principle applies to instances of CreateInvokeEJB 2.1 and FetchInvokeEJB 3.0 as well. However, it is somewhat more restrictive. CreateInvokeEJB 2.1 or FetchInvokeEJB 3.0 will automatically loop over each EJB returned and call the same business method on each. However, if you must be able to call more than one business method, you need to configure the necessary CreateEJB 2.1 or FetchEJB 3.0 and the corresponding InvokeEJB 2.1 or InvokeEJB 3.0 services and incorporate them into a flow service.

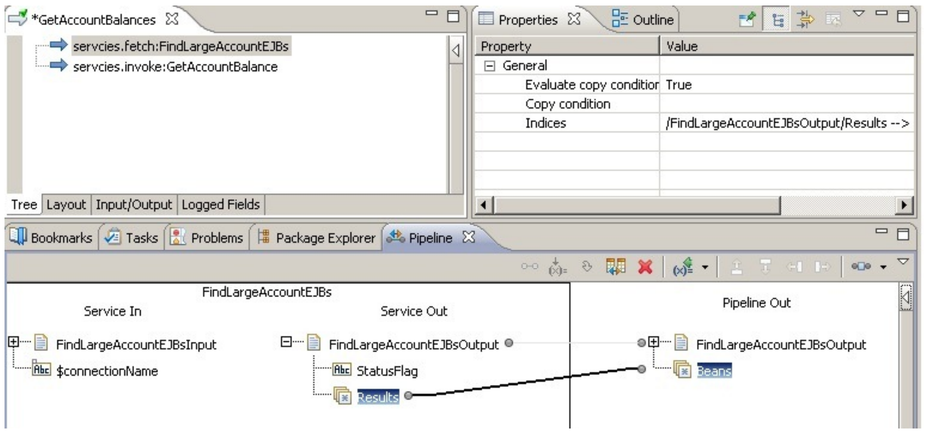
Regardless of how many EJB objects are returned by CreateEJB 2.1 or FetchEJB 3.0, each object is of the same remote EJB class. That is, they are the same type. Therefore, any invocation of a CreateEJB 2.1 or FetchEJB 3.0 adapter service will return one or more remote EJB objects of the same class.

This is important to note when designing flows. Because any given CreateEJB 2.1 or FetchEJB 3.0 instance is configured to work with a single EJB class, you can only expect to call the business methods exposed by that class. If in your flow you need to access some other EJB type, you will need to configure a separate CreateEJB 2.1 or FetchEJB 3.0 service for that type. If you attempt to pass an EJB object of one class into a corresponding InvokeEJB 2.1 or InvokeEJB 3.0 service configured for an entirely different class, you will see run-time exceptions.

The following example shows how to use the LOOP step in Designer to implement this processing. To begin, you configure a CreateEJB 2.1 or FetchEJB 3.0 adapter service instance (services.fetch.FindLargeAccountEJBs) that may return one or more EJBs in its *Results* array. You want to execute the same InvokeEJB 2.1 or InvokeEJB 3.0 adapter service against each EJB returned. However, you do not know how many EJBs might be returned.

For information about configuring CreateEJB 2.1 and FetchEJB 3.0 adapter services, see "Configuring CreateEJB 2.1 Services" on page 81 and "Configuring FetchEJB 3.0 Services" on page 88 respectively.

You must first define an Object list in the outbound pipeline immediately following **FindLargeAccountEJBs**. This list can have any valid name but it cannot be nested inside any other structure (for example, a Document). Create a link between the *Results* array and the new Object list in the outbound pipeline as shown in the figure below.

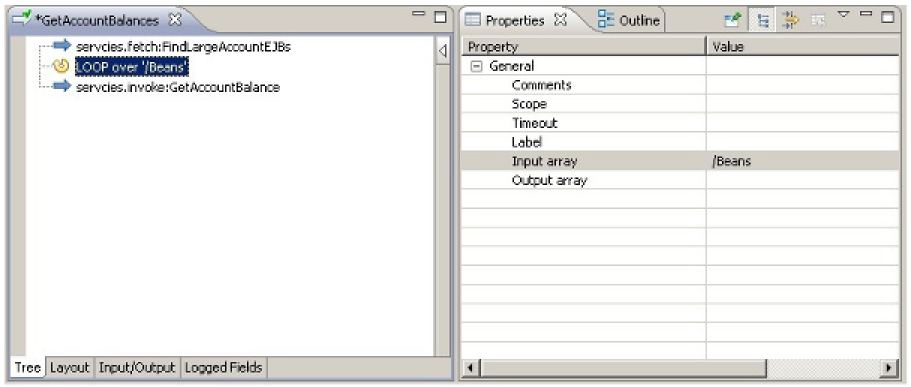


Property	Value
General	
Evaluate copy conditior	True
Copy condition	
Indices	/FindLargeAccountEJBsOutput/Results -->

Now insert a LOOP between the two adapter service instances (services.fetch.FindLargeAccountEJBs and services.invoke:GetAccountBalance). In this example there is a single child step to be executed within the body of the LOOP step: this is the InvokeEJB 2.1 or InvokeEJB 3.0 adapter service instance (services.invoke.GetAccountBalance).

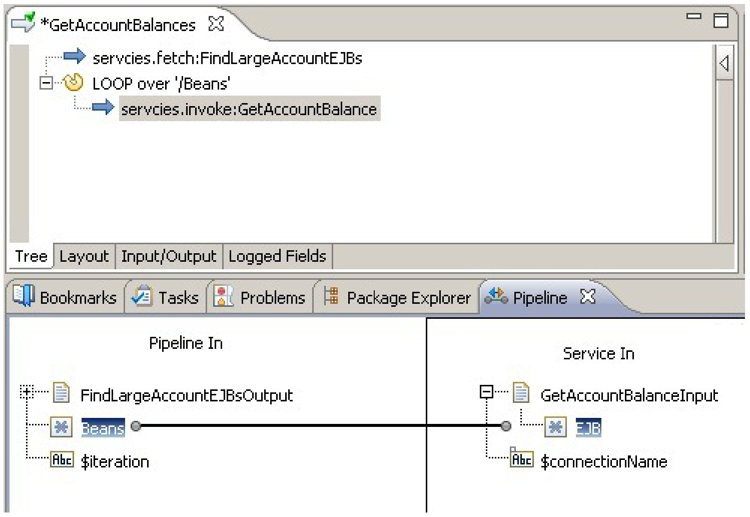
The loop should iterate once for each EJB found in the Beans array (the Object list defined earlier). To do this, enter the array's name (without the leading forward slash-Designer inserts this) in the **Input array** field in the Properties view.

You can see the steps for inserting a LOOP and entering the array's name in the figure below.



Property	Value
General	
Comments	
Scope	
Timeout	
Label	
Input array	/Beans
Output array	

Finally, complete the flow by selecting the services.invoke.GetAccountBalance service and creating a link between the *Beans* parameter (Designer automatically generated this parameter for you) and the input *EJB* parameter to the service as shown in the figure below.



Working with Different Object Types

EJB home and remote methods can take and return objects of practically any arbitrary type. The only real restriction is that the objects must implement the java.io.Serializable interface. How you work with these objects in your integrations depends on what it is that you need to do with the objects.

For example, if you need to pass an object generated by one adapter service call into a subsequent service in the same flow, you create the appropriate links in the flow service editor. Designer supports the java.lang.Object type natively. Any object that appears in the pipeline that is not recognized is treated as an Object. This is what happens when passing EJB objects themselves.

However, you may, at some point in your flow, need to interact directly with these objects. Suppose you need to assign a value to an object during the flow's execution, but have no service available for that purpose. Additionally, you want to create a document from such an object. How you provide this functionality depends on what *type* of object you are dealing with:

- Designer and Integration Server provide some degree of support for the basic Java types: java.lang.String, java.lang.Float, float, etc. If you only need to do transformations on these types of objects, the support typically already exists either natively within Designer or is provided in one of the WmPublic services.
- If you need to operate on a third-party, user-defined object type, you can create a utility as a Java service or a coded service using C/C++. The utility gets or creates the object you need, manipulates its properties, and then puts it on the pipeline.

  Note:
  You can create a coded service from within Designer or by using an outside Integrated Development Environment (IDE). For more information about building coded services, see the *webMethods Service Development Help* for your release.

Creating Java Services to Use with Objects

This section provides two examples that demonstrate how to write simple Java services to get or create the necessary objects, manipulate the properties, and then place the objects on the pipeline within a flow.

- The first example illustrates how to write a service that creates an instance of a user-defined class and then inserts it into the pipeline where it can be passed to an adapter service instance in a flow.
- The second example shows how to extract a user-defined object from the pipeline and transform it into a document.

The EJB and associated classes used in these examples are as follows:

- The target bean is a stateless session EJB-ComplexTrader. Its home interface exposes a create method that takes no arguments and returns an instance of a Trader.
- The Trader object itself exposes two public methods: buy and sell. Each of these has the same signature:

```
public  TradeResult buy (StockOrder order) throws RemoteException;
public  TradeResult sell (StockOrder order) throws RemoteException;
```

Example 1

The goal of this example is to write a flow that executes the buy method. Note that the buy signature contains two user-defined classes: TradeResult and StockOrder. The first example focuses primarily on StockOrder. In the flow, you need to create an instance of StockOrder with the appropriate attributes and then pass it to an InvokeEJB 2.1 or InvokeEJB 3.0 adapter service instance configured to execute the buy method. The easiest way to do this is to create a Java coded service that is specifically designed to accept input from the user at run time, create a StockOrder with those inputs, and then insert the StockOrder into the pipeline.

When creating the Java service, Designer automatically generated the public signature of the Java service, and its lone input argument is an instance of com.wm.data.IData.This is the pipeline referred to earlier.

Now look at the public interface of the StockOrder class:

```
package examples.ejb20.testbeans.stateless.ComplexTrader;
import java.io.Serializable;

/**
 * This class represents a buy/sell order for a single security.
 */

public final class StockOrder implements Serializable
{
  // Order duration constants...
  public static final int DAY = 0;   // Good for the Day
  public static final int GTC = 1;   // Good 'Til Cancelled
  // Order type constants (only long positions allowed)...
  public static final int MARKET = 0;      // Execute at current market price
  public static final int LIMIT = 1;       // For buys; execute at limit price or
lower
  public static final int STOP = 2;        // For sells; execute at stop price or
higher
  public static final int STOPLIMIT = 3;   // For sells; execute at no less that
stop, no more than limit
  public static final int ALLORNONE = 4;   // Execute entire order at market or
none of it
  public static final int FILLORKILL = 5;  // Execute entire order immediately at
market or none of it

  // Constructors
  /**
   * This constructor used for STOPLIMIT order types.
   */
  public StockOrder(String symbol, int quantity, int duration, int type, double
stop, double limit);

  /**
   * This constructor used for LIMIT, STOP order types.
   */
  public StockOrder(String symbol, int quantity, int duration, int type, double
price);

  /**
   * This constructor used for MARKET, ALLORNONE, FILLORKILL orders.
   */
  public StockOrder(String symbol, int quantity, int duration, int type);

  // "Getters"…
  public String getSecurity();
  public int getNumberUnits();
  public int getOrderDuration();
  public int getOrderType();
  public double getPrice();
  public double getStopPrice();
  public double getLimitPrice();
}
```

This class is used for both “buy” and “sell” orders. Essentially, you create and initialize a StockOrder object by calling one of its three constructors. The constructor used determines if the order is a market order, a stop order, a limit order, or a stop-limit order (see the embedded comments for descriptions of these terms in the public interface above).

For this example, assume that you want to execute a “buy” order, but do not want to pay more than a certain amount per share. In this case you would need to create the StockOrder object using the second constructor:

```
public StockOrder(String symbol, int quantity, int duration, int type, double
price);
```

You can see how to construct the StockOrder, but where do the actual values come from? You could simply generate them within the BuildLimitOrder service itself, but this is not very flexible. A better approach is to define an input signature for the service that allows the caller to provide the values that will be passed to the StockOrder constructor.

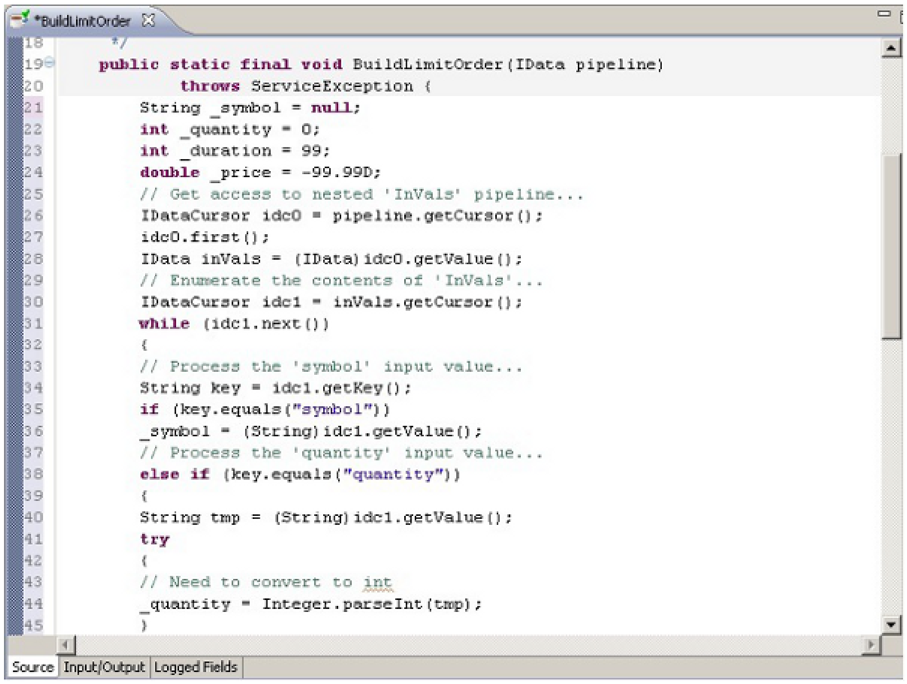
In addition to the *InVals* structured document declared as the input signature of BuildLimitOrder, a single Object definition, *order*, is declared in the output signature. The content of *InVals* consists of four String types: *symbol*, *quality*, *duration*, and *price*. These types correspond to four of the five arguments required by the StockOrder constructor. (The missing constructor argument, *type*, will be provided within the implementation of BuildLimitOrder itself.) Note, however, that the respective types of these parameters do not correspond in all cases (for example, the ‘price' constructor type is a primitive double). The easiest approach is to treat all the inputs as strings and simply perform the necessary type transformations within the service's implementation.

The following code is a complete implementation of the BuildLimitOrder service:

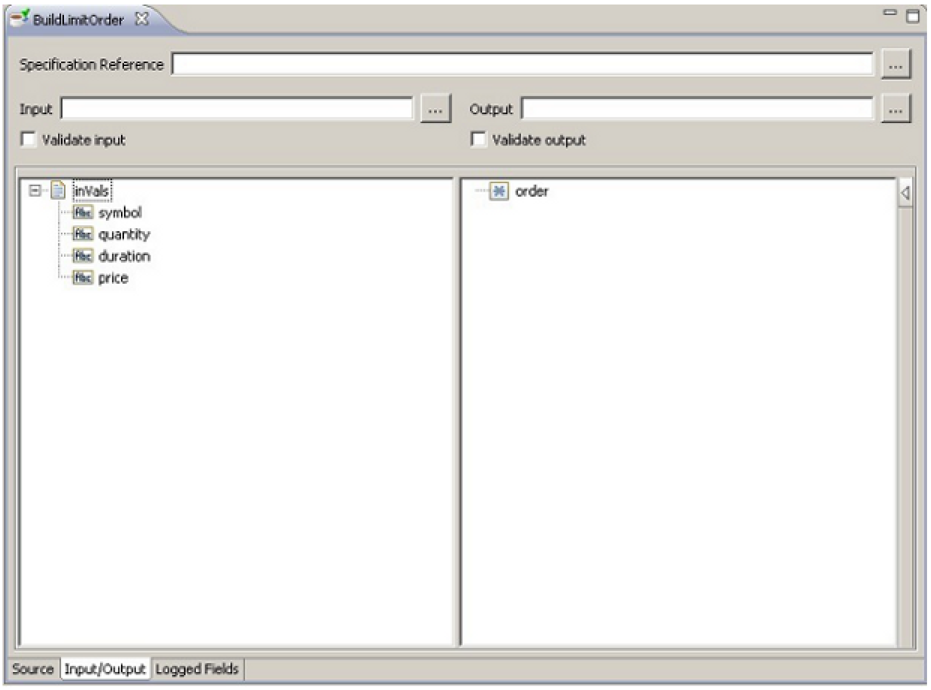
```
 public static final void BuildLimitOrder (IData pipeline) throws
ServiceException {
      String _symbol = null;
      int    _quantity = 0;
      int    _duration = 99;
      double _price = -99.99D;

      // Get access to nested 'InVals' pipeline...
      IDataCursor idc0 = pipeline.getCursor();
      idc0.first();
      IData inVals = (IData)idc0.getValue();

      // Enumerate the contents of 'InVals'...
      IDataCursor idc1 = inVals.getCursor();
      while (idc1.next())
      {
          // Process the 'symbol' input value...
          String key = idc1.getKey();
          if (key.equals("symbol"))
             _symbol = (String)idc1.getValue();

          // Process the 'quantity' input value...
          else if (key.equals("quantity"))
          {
             String tmp = (String)idc1.getValue();
             try
             {
                 // Need to convert to int
                 _quantity = Integer.parseInt(tmp);
              }
              catch (NumberFormatException e) {}
           }

           // Process the 'duration' input value...
           else if (key.equals("duration"))
           {
               // Convert string value to its enumerated counterpart
               String tmp = (String)idc1.getValue();
               if (tmp.equalsIgnoreCase("DAY"))
                   _duration = StockOrder.DAY;
 else if (tmp.equalsIgnoreCase("GTC"))
     _duration = StockOrder.GTC;
               else
                  _duration = 999;
            }

            // Process the 'price' input value...
            else if (key.equals("price"))
            {
                String tmp = (String)idc1.getValue();
                try
                {
                    // Need to convert to double
                    _price = Double.parseDouble(tmp);
                }
                catch (NumberFormatException e){}
            }
    }
    idc1.destroy();

    // Create a new LIMIT StockOrder...
    StockOrder so = new StockOrder(_symbol, _quantity, _duration,
StockOrder.LIMIT, _price);

    // Insert it in the outbound pipeline...
    idc0.insertAfter("order", so);
    idc0.destroy();
}
```

In the code that deals with the pipeline, note how when accessing elements in the input or output signature, it refers explicitly to the configured names of these elements (for example, “quantity”, “price”, “order”). This is one approach that can be taken. It is also possible to access items in the pipeline without referring to them by name. For a complete explanation of how to use the com.wm.data.IData and com.wm.data.IDataCursor classes, see the Javadocs provided in Designer and the *webMethods Service Development Help* for your release.

You can see the final, completed BuildLimitOrder service in the figures below.

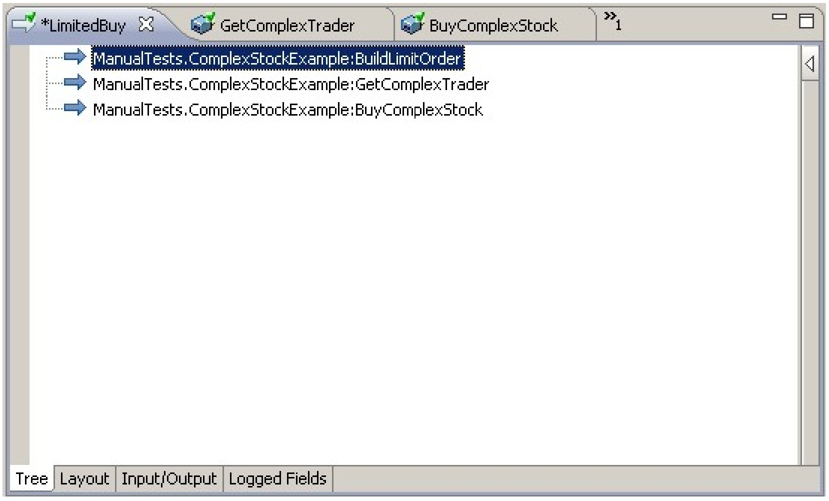


```
      */
    public static final void BuildLimitOrder(IData pipeline)
            throws ServiceException {
        String _symbol = null;
        int _quantity = 0;
        int _duration = 99;
        double _price = -99.99D;
        // Get access to nested 'InVals' pipeline...
        IDataCursor idc0 = pipeline.getCursor();
        idc0.first();
        IData inVals = (IData)idc0.getValue();
        // Enumerate the contents of 'InVals'...
        IDataCursor idc1 = inVals.getCursor();
        while (idc1.next())
        {
        // Process the 'symbol' input value...
        String key = idc1.getKey();
        if (key.equals("symbol"))
        _symbol = (String)idc1.getValue();
        // Process the 'quantity' input value...
        else if (key.equals("quantity"))
        {
        String tmp = (String)idc1.getValue();
        try
        {
        // Need to convert to int
        _quantity = Integer.parseInt(tmp);
        }
```

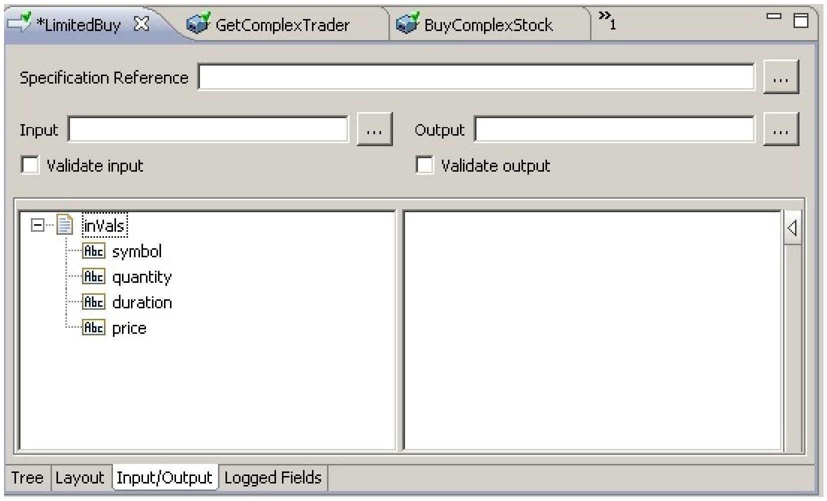


When you save the BuildLimitOrder service, Designer will compile it and report any errors it finds. When you have addressed all errors you can run the service standalone and debug it in Designer. After the BuildLimitOrder service is in place and tested you can add it to a flow service.

The next figures show a simple flow service named LimitedBuy that combines the new coded service, BuildLimitOrder, with the previously configured CreateEJB 2.1 or FetchEJB 3.0 adapter service (Trader) and a corresponding InvokeEJB 2.1 or InvokeEJB 3.0 adapter service (BuyComplexStock) instances.

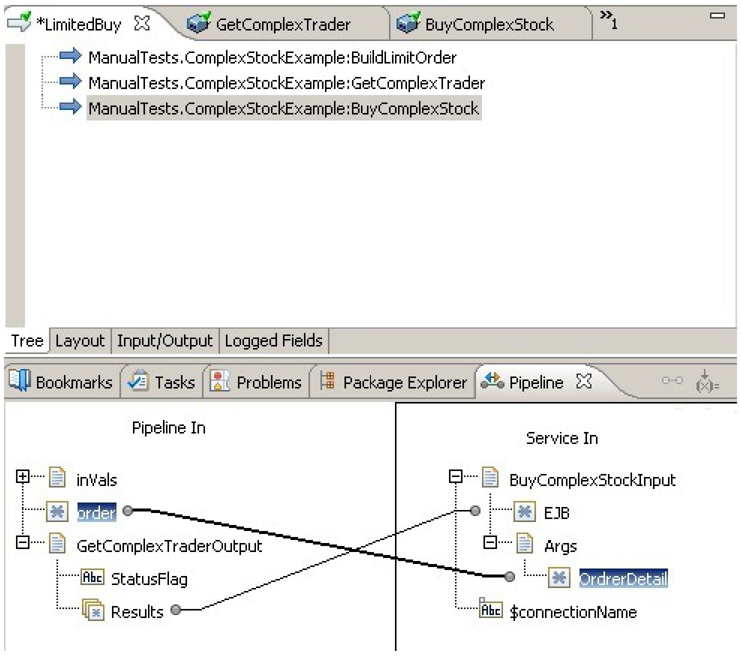


ManualTests.ComplexStockExample:BuildLimitOrder
ManualTests.ComplexStockExample:GetComplexTrader
ManualTests.ComplexStockExample:BuyComplexStock



Note that the input signature for the LimitedBuy flow mimics that of the BuildLimitOrder coded service. This allows you to map the values provided to the flow service at run time to the input signature of BuildLimitOrder.

Because the GetComplexTrader adapter service has no input signature, its inbound pipeline is simple. The signature of the BuyComplexStock adapter service is more complex. To fulfill its input signature, the output of BuildLimitOrder (the *order* Object) is mapped to the *OrderDetail* input Object. The mapping of the first element of *Results* to *EJB* passes the actual EJB into the BuyComplexStock adapter service as discussed earlier. These mappings are shown in the figure below.



Example 2

This example builds on "Example 1" on page 154 and shows how to convert the output of the BuyComplexStock adapter service into a simple document. The signature of the remote EJB method that is executed by BuyComplexStock looks like this:

```
public  TradeResult buy (StockOrder order) throws RemoteException;
```

Note that this method returns an object of type *TradeResult*. Examine the public interface of this class:

```
 public final class TradeResult implements Serializable
  {
    public TradeResult(String symbol, int quantity, double price);

    public String getStockSymbol();
    public int getSharesTraded();
    public double getExecutePrice();
  }
```

Notice that the output signature of the BuyComplexStock adapter service does not mention any *TradeResult* objects. Rather, it contains a document that in turn contains a String status field and another instance of an Object list named *Results*. This is not the same *Results* Object list as was returned by the GetComplexTrader adapter service. It just has the same name.

In this example, you will take the contents of *Results* (which is a list of opaque Objects to Designer) and turn it into a document that contains the actual values in the underlying *TradeResult* object. As with all CreateEJB 2.1 or FetchEJB 3.0 adapter services, all corresponding InvokeEJB 2.1 or InvokeEJB 3.0 adapter services will fill their outbound *Results* array starting at the first element (that is, the "zero-th" element). For this example, assume that the BuyComplexStock adapter service always returns a single *TradeResult* object in its *Results* array.

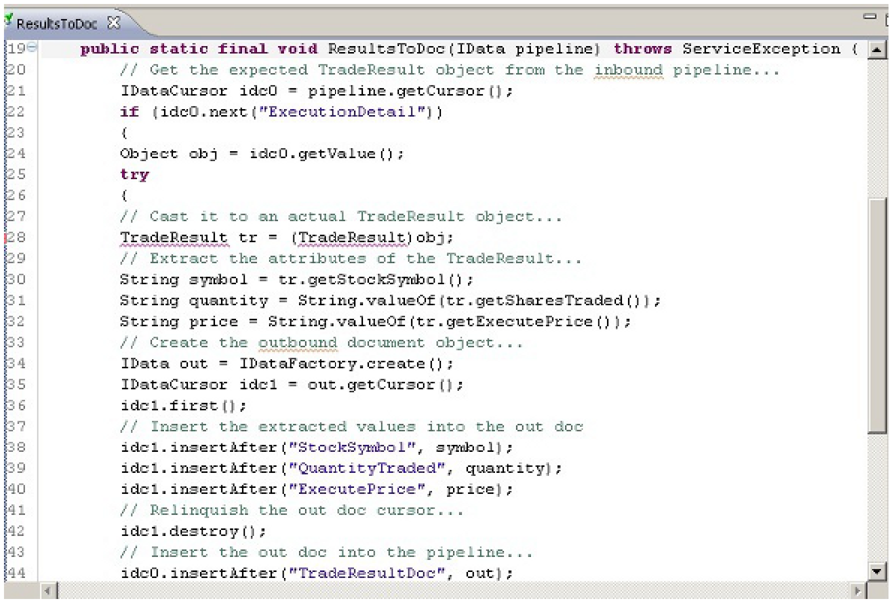
To accomplish this task, you need to write another coded Java service. Its implementation could be something like the following:

```
 public static final void ResultsToDoc (IData pipeline) throws ServiceException {
// Get the expected TradeResult object from the inbound pipeline...
IDataCursor idc0 = pipeline.getCursor();
if (idc0.next("ExecutionDetail"))
{
  Object obj = idc0.getValue();
  try
  {
    // Cast it to an actual TradeResult object...
    TradeResult tr = (TradeResult)obj;

    // Extract the attributes of the TradeResult...
    String symbol = tr.getStockSymbol();
    String quantity = String.valueOf(tr.getSharesTraded());
    String price = String.valueOf(tr.getExecutePrice());

    // Create the outbound document object...
    IData out = IDataFactory.create();
    IDataCursor idc1 = out.getCursor();
    idc1.first();

    // Insert the extracted values into the out doc
    idc1.insertAfter("StockSymbol", symbol);
    idc1.insertAfter("QuantityTraded", quantity);
    idc1.insertAfter("ExecutePrice", price);

    // Relinquish the out doc cursor...
    idc1.destroy();

    // Insert the out doc into the pipeline...
    idc0.insertAfter("TradeResultDoc", out);
  }
  catch (ClassCastException e) {}
}

// Relinquish the pipeline cursor...
idc0.destroy();
}
```

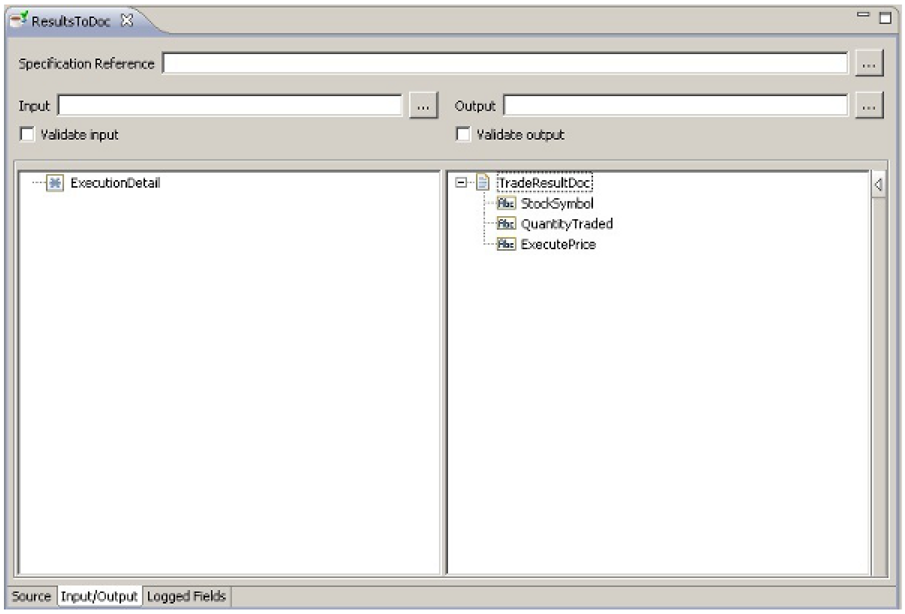
This coded service, ResultsToDoc, manipulates the content of the pipeline, performing the following tasks:

- Retrieves an object labeled ExecutionDetail from the pipeline
- Casts the ExecutionDetail object to the user-defined TradeResult type
- Extracts the values of the TradeResult object into three Strings: *StockSymbol*, *QuantityTraded*, and *ExecutePrice*
- Creates a new IData document called *TradeResultDoc*
- Inserts the three String values (*StockSymbol*, *QuantityTraded*, and *ExecutePrice*) into the *TradeResultDoc* document
- Adds the *TradeResultDoc* to the pipeline

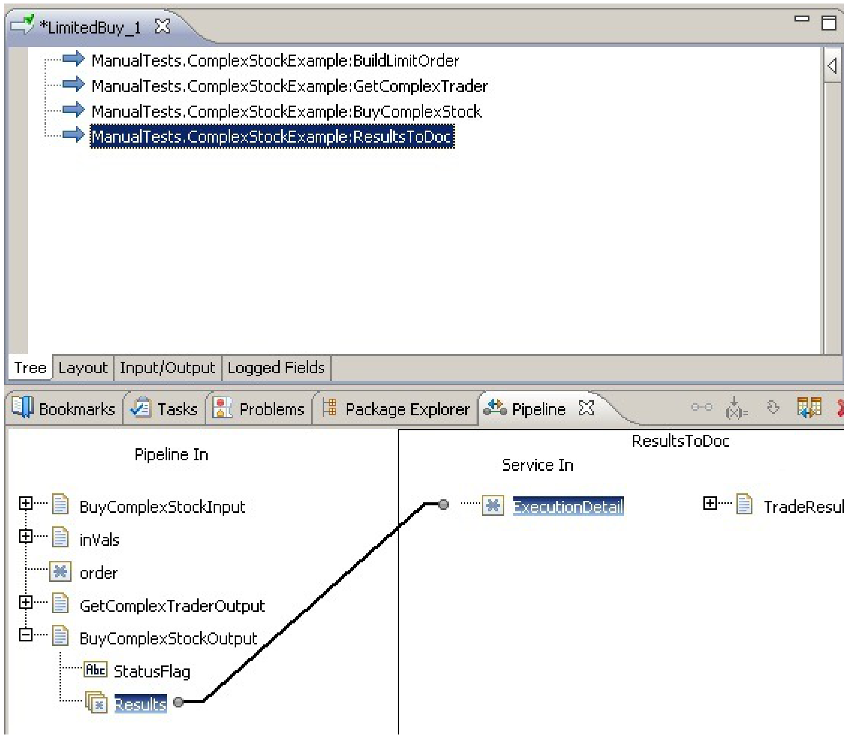
In order for this service to work properly, you must declare its input (*ExecutionDetail*) and output (*TradeResultDoc*) signatures as shown below.



```
    public static final void ResultsToDoc(IData pipeline) throws ServiceException {
        // Get the expected TradeResult object from the inbound pipeline...
        IDataCursor idc0 = pipeline.getCursor();
        if (idc0.next("ExecutionDetail"))
        {
        Object obj = idc0.getValue();
        try
        {
        // Cast it to an actual TradeResult object...
        TradeResult tr = (TradeResult)obj;
        // Extract the attributes of the TradeResult...
        String symbol = tr.getStockSymbol();
        String quantity = String.valueOf(tr.getSharesTraded());
        String price = String.valueOf(tr.getExecutePrice());
        // Create the outbound document object...
        IData out = IDataFactory.create();
        IDataCursor idc1 = out.getCursor();
        idc1.first();
        // Insert the extracted values into the out doc
        idc1.insertAfter("StockSymbol", symbol);
        idc1.insertAfter("QuantityTraded", quantity);
        idc1.insertAfter("ExecutePrice", price);
        // Relinquish the out doc cursor...
        idc1.destroy();
        // Insert the out doc into the pipeline...
        idc0.insertAfter("TradeResultDoc", out);
```



With the ResultsToDoc coded service complete, you can now add it to the LimitedBuy flow service, as shown in the figure below:



The LimitedBuy flow service is now complete. When you execute it with valid input values, you should see those values echoed in the *TradeResultDoc* document created by the final flow step.

D Application Server Configuration Notes

Overview

This appendix describes specific settings for required properties, and other information for the supported application servers that can be used with webMethods Adapter for EJB. For a complete list of supported application servers and versions, see "Installing, Upgrading, and Uninstalling" on page 45.

The information in this appendix supplements the information in "Adapter Connections" on page 17 and "Adapter Services" on page 25.

WebLogic Server

This section describes additional deployment and configuration information of BEA's WebLogic J2EE application server when used with Adapter for Enterprise Javabeans. Specifically, transaction support, security, and encryption are discussed. For information about the supported WebLogic Server versions, Java Runtime, and required class files, see "Installing, Upgrading, and Uninstalling" on page 45.

JAAS Authentication

When using multiple instances of WebLogic application server, JAAS Authentication should be used.

To configure JAAS Authentication

1. Open the is_jaas.cnf file located in *Integration Server_directory* \instances\ *instance_name* \config directory and add the following content:

```
WmEJBAdapter
{
weblogic.security.auth.login.UsernamePasswordLoginModule required debug=false;
};
```

2. Set the watt.ejbadapter.weblogic property to true.

Connection Properties

The content of the Java properties file required to create adapter connections is similar for all supported WebLogic versions. At a minimum, Adapter for Enterprise Javabeans requires that the standard java.naming.factory.initial and java.naming.provider.url properties be set as specified in "Required Properties and Values" on page 22.

Transaction Support

Adapter for Enterprise Javabeans is designed to interact with Integration Server's built-in transaction manager. For more information about connections and the associated transaction types, see "Adapter Connections" on page 17.

No Transactions

For all supported versions of WebLogic, full support is provided for connections that will not be transacted.

Local Transactions

For all supported versions of WebLogic, full support is provided for local transactions.

XA Transactions

XA support for WebLogic 10.3 is subject to the limitations of BEA's XAResource implementation as given below.

The XAResource that the adapter obtains from WebLogic and passes on to the built-in transaction manager represents an interface to the WebLogic server's transaction manager. In a distributed context, the WebLogic transaction manager would be responsible for coordinating all resources local to that server, yet still subordinate to Integration Server transaction manager. Integration Server transaction manager in effect, is the master transaction manager.

WebLogic's XAResource is designed to work in this environment provided that it is the only WebLogic XAResource involved in the entire distributed transaction. For example, you can use two or more XA-connected adapter services in a distributed transaction as long as those services are connected to the same WebLogic server. For this same distributed transaction, other, non-WebLogic resources may be included.

However, you cannot include an XAResource from two or more different WebLogic servers. For instance, two XA-connected adapter services that are connected to different WebLogic servers cannot be involved in the same distributed transaction.

For more information on WebLogic Server 10.3, see the Oracle web site.

Note:
Due to conflicts between the required jar files for Adapter for Enterprise Javabeans and the WmTomcat libraries, the WmTomcat package may not work as intended. Disable the WmTomcat package when you are using Adapter for Enterprise Javabeans with any of the supported application servers.

Security

This section provides a general description of those features of Adapter for Enterprise Javabeans that may impact security, in addition to some common security issues that may arise when integrating with WebLogic. This information should not be considered comprehensive.

JNDI Authentication

JNDI provides the following security-related properties that, if provided by the client, are passed into the WebLogic naming service implementation:

- java.naming.security.protocol
- java.naming.security.authentication
- java.naming.security.principal
- java.naming.security.credentials

For all supported versions of WebLogic:

- java.naming.security.protocol and java.naming.security.authentication are not used and may be ignored.
- java.naming.security.principal and java.naming.security.credentials may be used if a username and password are required to access the JNDI server. If they are not provided by the client, WebLogic uses a default principal value ("guest") and a default credentials value ("guest").

If the default values are sufficient to gain access to the deployed EJBs, you do not need to specify the principal and credentials properties. However, if the WebLogic administrator has defined a security policy that requires a specific username and password for accessing the server's EJBs, you must supply those values to Adapter for Enterprise Javabeans when configuring connections against that server. For information about how to specify the properties, see "Specifying JNDI Credentials" on page 23.

EJB Access

The EJB standard accommodates the definition of security roles in the EJB container that may be implemented by the application server. These roles, which are determined by the security credentials (if any) given to JNDI during construction of the initial context, can be used to determine which EJBs a client (that is, Adapter for Enterprise Javabeans) can access and which methods on a specific EJB a client can invoke. Any unauthorized attempts to access a restricted EJB will result in runtime exceptions being reported back to the adapter and the adapter runtime. All supported WebLogic versions can be configured to utilize these security features.

Encryption

Integration Server has built-in support for validating server certificates generated by the most widely-used certificate authorities. Adapter for Enterprise Javabeans can take full advantage of this capability for the purpose of establishing SSL connections to those WebLogic application installations that require it.

All supported versions of WebLogic can be configured to use SSL between the server and an EJB client. Additionally, SSL ports can be configured for either the HTTPS or T3S protocols. An EJB client may choose which of these protocols to connect with. This choice is explicitly set in the mandatory java.naming.provider.url property which must be present in the JNDI property file.

- If using HTTPS, the value of this property has the form:

  java.naming.provider.url = https://*WebLogic_server*:*port_number*

  where *WebLogic_server* and *port_number* are site-defined values.

- If using T3S, the value of this property has the form:

  java.naming.provider.url = t3s://*WebLogic_server*:*port_number*

Though the adapter supports the use of either protocol, we recommend using the proprietary T3/T3S for performance reasons. T3 is highly optimized and multiplexes all of a client's network traffic with WebLogic over a single socket connection. This is important when using SSL because it guarantees that both JNDI and RMI packets will be encrypted.

Configuring the WebLogic server for SSL is fairly straightforward on all supported versions, though there are some minor variations in the steps taken. For configuration information, see the Oracle web site.

Note:
All versions of WebLogic support services of EJB 2.1 and earlier standards. Services of the EJB 3.0 and 3.1 standards are supported only by WebLogic 10.3 and 12c.

WebSphere Server

This section describes additional deployment and configuration information of IBM's WebSphere J2EE application server when used with Adapter for Enterprise Javabeans. For information about the supported WebSphere Server versions, Java Runtime, and required class files, see “Installing, Upgrading, and Uninstalling” on page 45.

Connection Properties

The content of the Java properties file required to create adapter connections is similar for all supported WebSphere versions. At a minimum, Adapter for Enterprise Javabeans requires that the standard java.naming.factory.initial and java.naming.provider.url properties be set as specified in “Required Properties and Values” on page 22.

Depending upon the security configuration of the JNDI server, additional properties (for example, java.naming.security.authentication) may also be defined in this file. In addition to these standard JNDI property definitions, WebSphere itself recognizes the following properties that are relevant to client applications:

- com.ibm.CORBA.ConfigURL = file:*path-to-sas.client.props*

  See “JNDI Authentication” on page 168 for a discussion of how this property affects JNDI authentication.

- com.ibm.websphere.naming.jndicache.cacheobject = none

  By default, WebSphere caches connections to the JNDI server. However, the application server will attempt to re-use these connections even after connectivity between the client and WebSphere has been disrupted. In this case, you must restart the client JVM (that is, Integration Server). To ensure that WebSphere does not attempt to re-use potentially stale connections when interacting with JNDI, we recommend that you specify the cacheobject property in the JNDI properties file that is used by your adapter connections.

- java.util.logging.manager = com.ibm.ws.bootstrap.WsLogManager and java.util.logging.configureByServer = true

  For WebSphere Application Server (WAS) 6.0 clients, these two properties can be specified in the JNDI properties file that is used by your adapter connections. When specified, these properties eliminate logging of various javaAccessorNotSet and jndiGetObjInstNoop exceptions to the Integration Server console.

Transaction Support

Adapter for Enterprise Javabeans is designed to interact with Integration Server's built-in transaction manager. For more information about connections and the associated transaction types, see "Adapter Connections" on page 17.

For all supported versions of WebSphere, full support is provided for connections that will not be transacted.

For all supported versions of WebSphere, full support is provided for local transactions.

WebSphere does not expose its javax.transaction.xa.XAResource implementation to client applications. Therefore, XA transactions are not supported for WebSphere.

Note:
Due to conflicts between the required jar files for Adapter for Enterprise Javabeans and the WmTomcat libraries, the WmTomcat package may not work as intended. Disable the WmTomcat package when you are using Adapter for Enterprise Javabeans with any of the supported application servers.

Security

The following notes are a general description of those features of Adapter for Enterprise Javabeans that may impact security in addition to some common security issues that may arise when integrating with WebSphere. They should not in any way be considered comprehensive. For additional information, see the IBM documentation.

JNDI Authentication

JNDI provides the following security-related properties that, if provided by the client, are passed into the WebSphere naming service implementation:

- java.naming.security.protocol
- java.naming.security.authentication
- java.naming.security.principal
- java.naming.security.credentials

Though we have performed no testing in this area, it appears that WebSphere does not recognize these four properties. Instead, it uses the property settings in its sas.client.props file. This file defines IBM-specific properties that appear to mirror the functionality of the java.naming.security.* properties. The file contains embedded comments for each property.

To ensure Adapter for Enterprise Javabeans uses the sas.client.props file for JNDI-specific credentials

1. Copy the sas.client.props to the Integration Server disk. Ensure that the target location is available to Integration Server's JVM.

   This file is located on the WebSphere installation disk in the *WAS-Install-Root-Dir*\properties directory.

2. An Adapter for Enterprise Javabeans connection is then configured to use this file by setting the following property in the connection properties file:

   com.ibm.CORBA.ConfigURL = file:*path-to-sas.client.props*

   where *path-to-sas.client.props* is the full path to the sas.client.props file on the Integration Server disk.

EJB Access

The EJB standard accommodates the definition of security roles in the EJB container that may be implemented by the application server. These roles, which are determined by the security credentials (if any) specified in the sas.client.props file, ultimately can be used to determine which EJBs a client (that is, Adapter for Enterprise Javabeans) can access and even which methods on a specific EJB a client can invoke. Any unauthorized attempts to access a restricted EJB will result in runtime exceptions being reported back to the adapter and the adapter runtime. All supported WebSphere versions can be configured to utilize these security features.

Encryption

Integration Server has built-in support for validating server certificates generated by the most widely-used certificate authorities. Adapter for Enterprise Javabeans can take full advantage of this capability for the purpose of establishing SSL connections to all supported WebSphere application installations that require it.

The following notes may be helpful for enabling SSL to work with a thin EJB client. The information pertains to WAS 6.0, though the issues and setup should be similar for WAS 5.x.

Server Setup

- WAS uses two protocols for client-server communication: CSIv2 and SAS:
  - CSI is used for v5.0 and higher, and provides support for SSL.
  - SAS is generally used for v4.0 and earlier, and does not support SSL.

- There are five port numbers related to these two protocols; however, this does not matter to the client. You use the same URL (and the same port number) regardless of whether SSL is involved.
- The one port that matters is called "CSI inbound transport" on the server. Set this port to "SSL supported" or "SSL required".
- WebSphere installs with a default SSL "repertoire" already set up. The repertoire basically contains the details about the SSL configuration: where the keystore files are located, cipher suites in effect, keystore password. In WebSphere's administrative console you need to associate the CSIv2 protocol with this repertoire.
- To use SSL, you must enable "global security" in WebSphere's administrative console. (In our testing we found that it was necessary to enable this feature.) The implication of this is that you have to now assign login credentials to access the server from anywhere, including its administrative console.

  You can configure WAS to defer to an external authentication mechanism such as that imposed by the local operating system. Depending on your operating system, you may need to assign specific privileges to the process in which WebSphere runs in order to do this:

  - For Unix, WebSphere needs to run with root access.
  - For Windows, the user account from which WebSphere starts must be in the Administrators group and must have the "Act as part of the operating system" privilege enabled.

Client Setup

- WebSphere's default SSL repertoire uses a self-signed certificate, so you need to copy the DummyServer*.jks files to the client's filesystem.
- Copy the *WAS-Install-Root-Dir*\properties\sas.client.props file and edit it according to the embedded comments (this includes pointing to the two .jks files)
- Add the com.ibm.CORBA.ConfigURL property to your WebSphere JNDI properties file. Set its value to file:*path-to-sas.client.props*

After setting up the server and the client you will be using SSL for all RMI traffic between client and server. It is not clear from IBM's documentation whether JNDI traffic is likewise encrypted.

Note:
All versions of WebSphere support services of EJB 2.1 and earlier standards. Services of the EJB 3.0 and 3.1 standards are supported only by WebSphere 8.5.

JBoss Server

This section describes the deployment and configuration of webMethods Adapter for Enterprise JavaBeans for use with JBoss J2EE application server. For information about the supported JBoss Server versions, Java Runtime, and required class files, see "Installing, Upgrading, and Uninstalling" on page 45.

Connection Properties

The content of the Java properties file required to create adapter connections is consistent for all supported JBoss versions. At a minimum, Adapter for Enterprise Javabeans requires that the standard java.naming.factory.initial and java.naming.provider.url properties be set, as well as the java.naming.factory.url.pkgs. Values for these properties are as specified in "Required Properties and Values" on page 22.

Transaction Support

Adapter for Enterprise Javabeans is designed to interact with Integration Server's built-in transaction manager. For more information about connections and the associated transaction types, see "Adapter Connections" on page 17.

For all supported versions of JBoss, full support is provided for connections that will not be transacted.

For all supported versions of JBoss, full support is provided for local transactions.

JBoss does not expose any XA data source to clients. Therefore, XA transactions are not supported for JBoss.

Note:
Due to conflicts between the required jar files for Adapter for Enterprise Javabeans and the WmTomcat libraries, the WmTomcat package may not work as intended. Disable the WmTomcat package when you are using Adapter for Enterprise Javabeans with any of the supported application servers.

Security

The following notes are a general description of those features of Adapter for Enterprise Javabeans that may impact security in addition to some common security issues that may arise when integrating with JBoss. They should not in any way be considered comprehensive.

JNDI Authentication

JNDI provides the following security-related properties that, if provided by the client, are passed into the JBoss naming service implementation:

- java.naming.security.protocol
- java.naming.security.authentication
- java.naming.security.principal
- java.naming.security.credentials

JBoss will recognize all of the above except java.naming.security.authentication, if provided. The default JBoss installation requires none of the properties. They would only be needed if your JBoss administrator has configured the server to expect them.

If security credentials are required, these values may be specified as discussed in "Specifying JNDI Credentials" on page 23.

EJB Access

The EJB standard accommodates the definition of security roles in the EJB container that may be implemented by the application server. These roles, which are determined by the security credentials (if any) given to JNDI during construction of the initial context, can be used to determine which EJBs a client (that is, Adapter for Enterprise Javabeans) can access and which methods on a specific EJB a client can invoke. Any unauthorized attempts to access a restricted EJB will result in runtime exceptions being reported back to the adapter and the adapter runtime. All supported JBoss versions can be configured to utilize these security features.

Encryption

Integration Server has built-in support for validating server certificates generated by the most widely-used certificate authorities. Adapter for Enterprise Javabeans can take full advantage of this capability for the purpose of establishing SSL connections to those JBoss application servers that support it.

JBoss supports JNDI over SSL. For more information on JBoss support for JNDI over SSL, see the JBoss documentation.