webMethods Adapter Runtime User's Guide

Innovation Release

Version 10.0

April 2017

WEBMETHODS

This document applies to webMethods Adapter Runtime Version 10.0 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: ADAPTER-ART-UG-100-20170419

Table of Contents

About this Guide

This guide describes how to use the Adapter Runtime. It contains information for administrators and application developers who work with webMethods adapters.

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

- Terminology and basic operations of your operating system
- The setup and operations 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 storage locations for services on webMethods Integration Server, using the convention *folder.subfolder:service* .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
Italic	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
`Monospace font`	Identifies text you must type or messages displayed by the system.
{ }	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.

Convention	Description
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at "http://documentation.softwareag.com". The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at "https://empower.softwareag.com".

To submit feature/enhancement requests, get information about product availability, and download products, go to "Products".

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the "Knowledge Center".

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at "http://techcommunity.softwareag.com". You can:

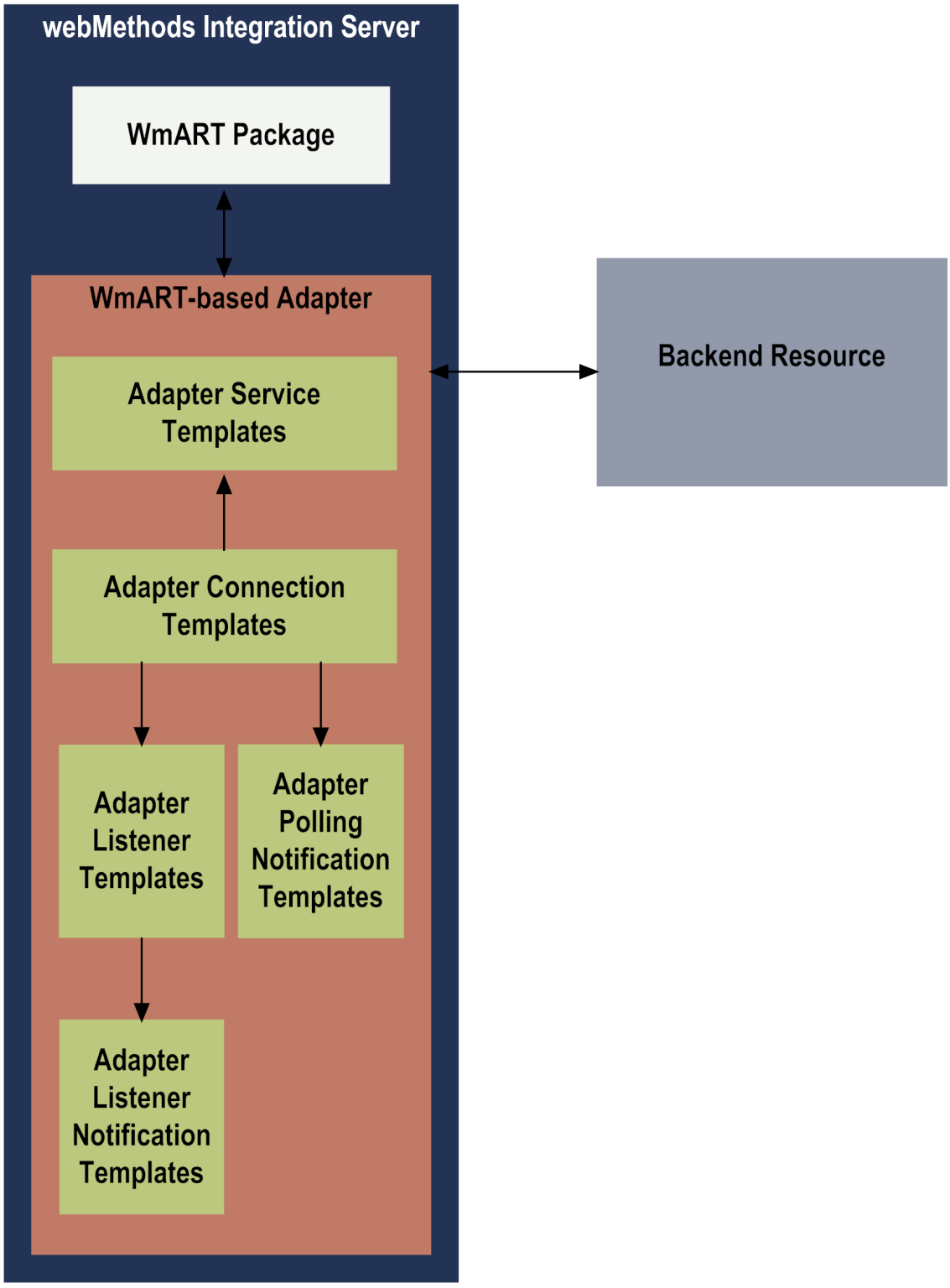
- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

1 Overview of the Adapter Runtime

What Is the Adapter Runtime?

The Adapter Runtime provides a common framework for webMethods adapters version 6 and later to use webMethods Integration Server's functionality, making Integration Server the run-time environment for the adapters. The Adapter Runtime functionality is delivered as the WmART package, which is automatically installed with Integration Server. The WmART package provides logging, transaction management and error handling for adapter connections, services, notifications, and listeners.

The following diagram shows at a high level how an adapter uses WmART to interact with the back end.


webMethods Integration Server
WmART Package
WmART-based Adapter
Adapter Service Templates
Adapter Connection Templates
Adapter Listener Templates
Adapter Polling Notification Templates
Adapter Listener Notification Templates
Backend Resource

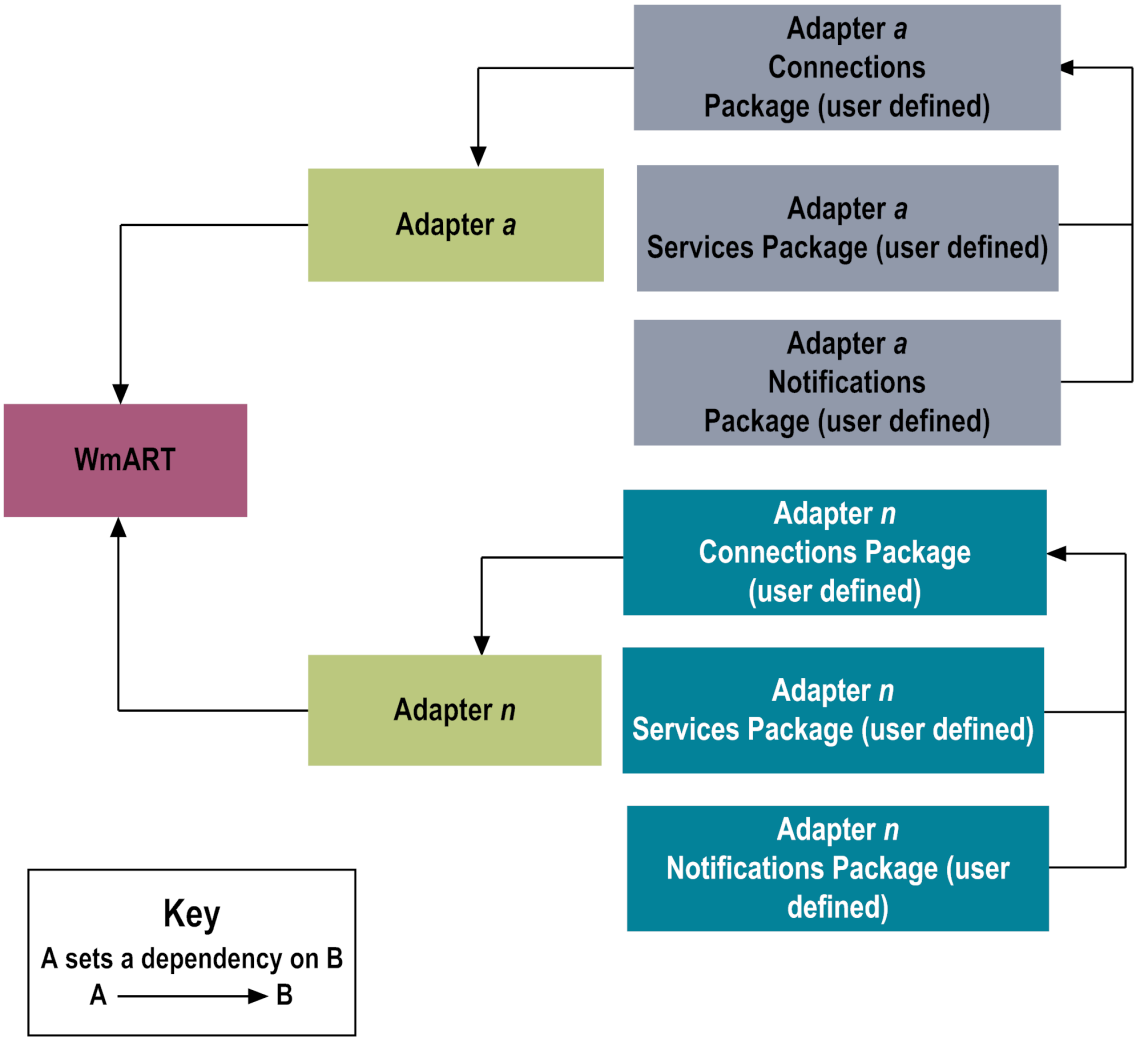
Package Management

Each webMethods adapter is provided as a separate package that has a dependency on the WmART package. When you create connections, adapter services, listeners, and notifications for an adapter, you must define them in user-defined packages rather than in the adapter package. The user-defined packages, in turn, have a dependency on the adapter package.

You manage the WmART package, the adapter package and user-defined packages as you would manage any package on Integration Server.

When Integration Server starts, it automatically loads or reloads the WmART package first, the adapter package next, and the user-defined packages last.The WmART package is automatically installed when you install Integration Server. You should not need to manually reload the WmART package.

The following diagram shows the various package dependencies.



User-Defined 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 *webMethods Service Development Help*.

When working with user-defined packages, keep in mind the following:

- When you create user-defined packages, use the package management functionality provided in Software AG Designer and set the user-defined packages to have a dependency on the adapter package. That way, when the adapter package loads or reloads, the user-defined packages load automatically.
- 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.

- If the connections and adapter services of an adapter are defined in different packages:
  - A package that contains the connections must have a dependency on the adapter package.
  - Packages that contain adapter services must have a dependency on their associated connection package.
- Integration Server will not allow you to enable a package if it has a dependency on another package that is disabled. Before you can enable your package, you must enable all packages on which your package depends.
- 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.
- You can give connections, adapter services, and notifications the same name provided that they are in different folders and packages.

Adapter Connections

You create one or more adapter connections at design time to use in integrations. The number of connections you create depends on your integration needs. When an adapter connection is created, the WmART package creates a connection object. An adapter connection enables Integration Server to connect to the back end at run time. You must configure an adapter connection before you can create adapter services or notifications.

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 **Minimum Pool Size** parameter of the connection. 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**. For information about configuring connections, see the installation and user's guide for the individual adapter.

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, 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.

Additionally, the pub.art.service:setAdapterServiceNodeConnection and pub.art.notification:setPollingNotificationNodeConnection services allow you to change the connection associated with an adapter service or polling notification respectively at design time.

Run-Time Connection Allocation for Adapter Services

When an adapter service is invoked, either directly or from a flow service, the Adapter Runtime provides a connection object to the adapter service. This section describes how connections are retrieved and managed and how to dynamically control the type of connection used for each service invocation.

At run time, all connection activity for adapter services is performed inside a transaction context that holds references to connections used while the context is open. This is true regardless of whether the referenced connections are transacted. There is an implicit transaction context that begins at the invocation of a top-level flow service (such as an HTTP invocation of an Integration Server service) and continues until that top-level service exits. Additional contexts can be created using the pub.art.transaction:startTransaction service and ended using pub.art.transaction:commitTransaction or pub.art.transaction:rollbackTransaction. For more information about using these services, see "Adapter Runtime Built-In Services Reference" on page 57.

When the Adapter Runtime retrieves a connection from a connection pool for use by an adapter service, a reference to that connection is placed in the transaction context, and the connection is not returned to the pool until the transaction context is closed. If another adapter service call is made within the transaction context, Integration Server will first determine whether a connection from the required connection pool and partition is in the context; if so, Integration Server will use the connection from the transaction context to the adapter service instead of requesting another from the connection pool.

Dynamically Selecting a Connection Node

Each connection node should be used to access a single physical resource. In some integration environments, similar functionality is available on multiple physical

resources. In these cases, a single adapter service node may be used to access those resources by dynamically specifying which connection node to use for a particular service invocation.

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

Alternatively, you can use the configured connection for an adapter service, but at run time override the user credentials defined in the connection. To override the user credentials, you must code your flow service to pass values through the pipeline into the adapter service's *user name* and *password* fields.

Adapter Services

An adapter service defines an operation that the adapter will perform on an adapter resource. Adapter services operate like Integration Server flow services or Java services. You call adapter services within flow or Java services, and you can audit them from the Integration Server's audit system.

Adapter services have *input* and *output* signatures. An input signature describes the data that the service expects to find in the flow service pipeline at run time. An output signature describes the data that the service expects to add to the pipeline when it has successfully executed. You can view an adapter service node's signature on the **Input/Output** tab of the Adapter Service Editor in Software AG Designer.

Adapter services are based on templates provided with each adapter. Each template represents a specific technique for doing work on a resource, such as using the a template to retrieve specified information from a database.

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 create a new adapter service in Designer. Before configuring an adapter service, you must assign it a connection that you created earlier.

Adapter Polling Notifications

A polling notification is a facility that enables an adapter to initiate activity on Integration Server, based on events that occur in the adapter resource. A polling notification monitors an adapter resource for changes (such as an insert, update, or delete operation) so that the appropriate flow or Java services can react to the data, such as sending an invoice or publishing an invoice to Integration Server.

You create a polling notification node using Designer. You assign to the notification an adapter connection node that you created earlier.

Polling notifications cannot be directly invoked from a flow service (or from Designer). Instead, the server invokes a polling notification automatically, based on a fixed time

interval. When a polling notification determines that a specified event has occurred in the adapter resource, it produces a document describing the event. These documents are automatically published to Integration Server (or webMethods Broker) as they are generated by the notification. The processing of the published document is based on triggers that are configured to invoke flow services when the given document type is published. For more information on Integration Server publishable documents, see *Publish-Subscribe Developer's Guide*.

Adapter Listeners and Listener Notifications

Listeners and listener notifications work together to create a much more powerful model for detecting and processing events in the adapter resource than is possible with polling notifications.

With a listener notification, the responsibility for monitoring the adapter resource and processing any events is divided between a listener and its notification(s). A listener object is instantiated and is given a connection when you enable the associated node. The listener object remains active with the same connection to monitor the resource activity until it is disabled (either explicitly or by disabling the containing package, the adapter, the connection, or Integration Server). When the listener detects a publishable event in the resource, it passes an object back to the server. The server will interrogate a configured list of listener notifications associated with the listener node until it finds a listener notification node that can process the event. The listener notification processes the event either asynchronously or synchronously.

Synchronous and Asynchronous Listener Notifications

When you create an *asynchronous* listener notification, the adapter creates a publishable document type. At run time, when the listener detects an event in the back end, it invokes the asynchronous notification. The asynchronous notification then publishes a document, which has the structure defined by the publishable document type, in either of the following ways:

- to webMethods Broker when Integration Server is connected to Broker.
- to webMethods Universal Messaging when Integration Server is connected to Universal Messaging.
- to Software AG Digital Event Services using IS_DES_CONNECTION alias.
- to a JMS queue or topic when Integration Server is connected to a JMS provider.
- locally to Integration Server when Integration Server is not connected to Broker or to a JMS provider.

Adapter users can process the document's data any way they want to. For example, they can create an Integration Server trigger that receives the document and executes an Integration Server flow service or a Java service.

At run time, a *synchronous* listener notification invokes a specified IS service, and potentially receives a reply from the service and delivers the results back to the adapter resource. The listener notification waits until the service has finished processing before it begins to process the next message from the adapter resource. A synchronous listener notification does not publish a document.

Synchronous listener notifications do not support session handling. When a synchronous listener notification calls a service that needs information contained in the session data, that service can fail. However, note that the same service may appear to work for an asynchronous listener notification. This is because asynchronous listener notifications themselves do not execute a service. Instead, an Integration Server trigger, which supports session handling, is used to receive the document and execute an Integration Server flow service or a Java service.

Single-Threaded and Multi-Threaded Listeners

Beginning with Integration Server 9.5, Adapter Runtime based adapters support *single-threaded* listeners and *multi-threaded* listeners, based on the number of threads that a listener uses to process messages coming from the back end. Multi-threaded listeners improve performance by streamlining message processing.

With a single-threaded listener, the adapter processes incoming events from the back end one after the other in the order in which they arrive. The thread executes the most suitable notification for each event.

With a multi-threaded listener, the adapter uses a new thread for each event coming from the back end and processes the messages concurrently, which boosts performance. Therefore, if you have a long-running flow service for processing a message, subsequent messages do not have to wait for the first message to be processed.

Multi-threaded listeners rely on the Integration Server thread pool to concurrently read and process messages. When you enable a multi-threaded listener, for each incoming event from the back end an idle thread from the thread pool is allocated to execute the listener notification. When the maximum number of threads is reached, as specified in the **Thread Count** parameter when configuring a listener, new events have to wait and are processed only when threads become available.

With a multi-threaded listener, if the thread executing the notification uses a connection providing transaction support, the thread is responsible for committing or rolling back the transaction. In this case, each thread uses a separate connection.

Event Publishing Support for Adapter Notifications

Beginning with Integration Server 9.5, adapter polling and listener notifications can send events to an event bus. The event bus functionality is implemented by webMethods Universal Messaging using the JMS protocol. Publishing events to the event bus is *not* enabled by default.

For more information about how to implement an Event-Driven Architecture (EDA) system with webMethods products, see *Implementing Event-Driven Architecture with Software AG Products*.

The following table lists the tasks required to use event publishing for adapter notifications.

Task	Use This Tool
1. Configure a polling or listener notification. For information about configuring a polling or listener notification, see the notifications chapter in the installation and user's guide for the adapter.	Designer
2. Create an event type definition. For information about creating an event type definition, see "Creating an Event Type Definition" on page 19.	Designer
3. Copy the event type definition to the Event Type Store. For information about copying the event type definition, see "Copying the Event Type Definition" on page 20.	Designer
4. Enable the notification. For information about enabling notifications, see the notifications chapter in the installation and user's guide for the adapter.	Integration Server Administrator
5. Enable the publishing of events for the notification. For information about enabling the publishing of events, see "Enabling the Publishing of Events" on page 20.	Integration Server Administrator

Creating an Event Type Definition

You create event types using the Events Development perspective in Designer. For more information about working with event types, see *webMethods BPM Process Development Help* .

Important: The event type definition must have the same name as the adapter notification.

To create an event type definition

1. In the **Project Explorer** view in Designer, right-click the project in which you want to create the event type and select **New > Event Type**. The New Event Type wizard appears displaying the project name.
2. Specify values for the following fields:

Field	Description
Folder	Optional. The folder where the new event will be stored. If you do not enter a name, the event type will be stored directly in the Event Types folder.
Name	The name of the event type.

3. Select **Start with Integration Server Document Type**.
4. In the **Document Type** field, browse to the publishable document of the adapter notification for which you want to create an event type definition.
5. Click **Finish**.

Copying the Event Type Definition

After you create an event type definition in Designer, you must manually copy it to the Event Type Store. The namespace of the event type in the Event Type Store must be the same as the namespace of the associated adapter notification.

For example, if you configured an adapter notification named Notification1 in Folder1 of Package1, and then you created an event type named Notification1, you must copy the event type to C:\SoftwareAG\common\EventTypeStore\Package1\Folder1.

The default path to the Event Type Store is C:\SoftwareAG\common\EventTypeStore. For more information about the Event Type Store, see *Implementing Event-Driven Architecture with Software AG Products*.

Enabling the Publishing of Events

You enable the publishing of events for an adapter notification either in Integration Server Administrator or using the pub.art.notification:enablePublishEvents service.

To enable the publishing of events in Integration Server Administrator

1. In Integration Server Administrator, go to **Adapters > *Adapter Name* > Notifications**.
2. In the **Publish Events** column, click **No** for the notification for which you want to enable the feature. Integration Server Administrator displays **Yes** in the **Publish Events** column.

Adapter Runtime uses the properties watt.art.notification.eventBus.retries and watt.art.notification.eventBus.retryInterval to verify that the event type definition is copied to the Event Type Store. Publishing of events is not enabled if the Adapter Runtime does not find the event type definition in the Event Type Store. Instead, Integration Server Administrator displays an error message.

Disabling the Publishing of Events

You disable the publishing of events for an adapter notification either in Integration Server Administrator or using the pub.art.notification:disablePublishEvents service.

To disable the publishing of events

1. In Integration Server Administrator, go to **Adapters > *Adapter Name* > Notifications**.
2. In the **Publish Events** column, click **Yes** for the notification for which you want to disable the feature. Integration Server Administrator displays **No** in the **Publish Events** column.

Transaction Support

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. The interactions within a transaction are either all committed or all rolled back. For example, if a transaction includes multiple database inserts, and one or more inserts fail, all inserts are rolled back.

Integration Server supports the following kinds of transactions:

- A *local transaction*, which is a transaction to a resource's local transaction mechanism
- An *XAResource transaction*, which is a transaction to a resource's XAResource transaction mechanism

Integration Server can automatically manage both kinds of transactions, without requiring the adapter user to do anything. Integration Server uses the container-managed (implicit) transaction management approach as defined by the JCA standard and also performs some additional connection management. This is because adapter services use connections to create transactions. However, there are cases where the adapter user needs to explicitly control the transactional units of work.

To support transactions, Integration Server relies on a built-in transaction manager. The transaction manager is responsible for beginning and ending transactions, maintaining a transaction context, enlisting newly connected resources into existing transactions, and ensuring that local and XAResource transactions are not combined in illegal ways.

For more information about transactions, see on page 23“Transaction Management”.

Controlling Pagination

When using the adapter on Integration Server 8.2 and later, you can control the number of items that are displayed on the adapter Connections screen and Notifications 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. For example, to display 50 items per page, specify:

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

For information about working with extended configuration settings, see *webMethods Integration Server Administrator's Guide*.

2 Transaction Management

Overview

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. The interactions within a transaction are either all committed or all rolled back. For example, if a transaction includes multiple database inserts, and one or more inserts fail, all inserts are rolled back.

Integration Server supports the following kinds of transactions:

- A local transaction (LOCAL_TRANSACTION), which is a transaction to a resource's local transaction mechanism.
- An XAResource transaction (XA_TRANSACTION), which is a transaction to a resource's XAResource transaction mechanism.

Integration Server can automatically manage both kinds of transactions, without requiring the adapter user to do anything. Integration Server uses a container-managed (implicit) transaction management approach based on the Connector Architecture standard, and also performs some additional connection management because adapter services use connections to create transactions. For more information about implicit transactions, see "Implicit Transactions" on page 24.

However, there are cases where you need to explicitly control the transactional units of work. For more information about explicitly managing transactions, see "Explicit Transactions" on page 26.

To support transactions, Integration Server relies on a transaction manager. The transaction manager is responsible for beginning and ending transactions, maintaining a transaction context, enlisting newly connected resources into existing transactions, and ensuring that local and XAResource transactions are not combined in illegal ways.

The transaction manager *only* manages operations performed by adapter services, a transacted JMS trigger, or a built-in JMS service that uses a transacted JMS connection alias. Since the transaction manager cannot manage operations performed by any other service, a commit or rollback is not applicable for operations performed by those services.

Important: You cannot step or trace a flow that contains a transacted adapter service.

Implicit Transactions

With implicit transactions, Integration Server automatically manages both local and XAResource transactions without requiring you to explicitly do anything. Integration Server starts and completes the implicit transaction.

An implicit transaction context, which the transaction manager uses to define a unit of work, starts when an adapter service is encountered in a flow execution. The connection

required by the adapter service is registered with the newly created context and used by the adapter service. If another adapter service is encountered, the transaction context is searched to see if the connection is already registered. If the connection is already registered, the adapter service uses this connection. If the connection is not registered, a new connection instance is retrieved and registered 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 is completed and is either committed or rolled back, depending on the status (success or failure) of the top level flow.

A single implicit transaction context can contain any number of XA_TRANSACTION connections but no more than one LOCAL_TRANSACTION connection. If you choose to provide dynamic user credentials at run time, all the adapter services using the LOCAL_TRANSACTION connection within a single transaction must use the same user credentials. For example, if you have two adapter services, s1 and s2, configured using the LOCAL_TRANSACTION connection c1 in a single transaction context, both s1 ands2 must either use the same dynamic credentials at run time or the default configured credentials provided at design time. For more information about providing dynamic user credentials for a service's associated connection, see “Changing the User Credentials of a Service's Associated Connection at Run Time”.

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

Implicit Transaction Usage Cases

Manging a flow implicitly requires any of the following:

- One local transaction, interacting with one resource. For an example, see “One Local Transaction” on page 25.
- One or more XAResource transactions. Each transaction can interact with one or more resources. For an example, see “Three XAResource Transactions” on page 26.
- One or more XAResource transactions and one local transaction. For an example, see “One Local Transaction and One XAResource Transaction” on page 26.

If a flow contains multiple local transactions, you must explicitly control the transactional units of work, as described in “Explicit Transaction Usage Cases” on page 27.

One Local Transaction

In this example, a flow with two adapter services interacts with the same local transaction resource. The flow performs two inserts into two tables of a database:

```
BEGIN FLOW
INVOKE insertDatabase1TableA // Local Transaction Resource1
INVOKE insertDatabase1TableB // Local Transaction Resource1
END FLOW
```

Integration Server starts the transaction when insertDatabase1TableA is invoked. It opens a connection to the resource, enlists it in the transaction, and performs the insert into TableA. When insertDatabase1TableB is invoked, Integration Server reuses the same connection to insert data into TableB. When the request is complete, Integration Server closes the connection and commits the transaction.

The following flow is *illegal* because it tries to interact with two local transaction resources as follows:

```
BEGIN FLOW
INVOKE insertDatabase1TableA // Service for Resource1
INVOKE insertDatabase2TableA // Service for Resource2
END FLOW
```

Three XAResource Transactions

The following flow is valid because a flow can contain any number of XAResource transactions.

```
BEGIN FLOW
INVOKE insertDatabase1TableA // XAResource Transaction Resource1
INVOKE insertDatabase2TableA // XAResource Transaction Resource2
INVOKE insertDatabase3TableA // XAResource Transaction Resource3
END FLOW
```

One Local Transaction and One XAResource Transaction

Continuing with the previous case, notice this additional insert to a different database that accepts XAResource transactions as follows:

```
BEGIN FLOW
INVOKE insertDatabase1TableA // Local Transaction Resource1
INVOKE insertDatabase1TableB // Local Transaction Resource1
INVOKE insertDatabase2TableA // XAResource Transaction Resource1
END FLOW
```

When Integration Server invokes insertDatabase2TableA, a transaction is already in progress with the first database enlisted. It then establishes a second connection (to Database2), enlists the new connection in the XAResource transaction, and performs the insert to tableA.

When the request is complete, Integration Server closes both connections and the Transaction Manager performs a local commit for the non-XAResource and then a two-phase commit for the XAResource enlisted in the transaction.

Explicit Transactions

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

An explicit transaction context starts when the pub.art.transaction:startTransaction service is executed. The transaction context is completed when either the pub.art.transaction:commitTransaction or pub.art.transaction:rollbackTransaction service is executed.

For information about the services, see "Built-In Services for Transaction Management" on page 31.

When you define an explicit transaction, it is nested within the implicit transactions that are controlled by the transaction manager. You can have more than one explicit transaction defined within an implicit transaction. You can also nest explicit transactions within each other.

As with implicit transactions, a single explicit transaction context can contain any number of XA_TRANSACTION connections but no more than one LOCAL_TRANSACTION connection. If you choose to provide dynamic user credentials at run time, all the adapter services using the LOCAL_TRANSACTION connection within a single transaction must use the same user credentials. For example, if you have two adapter services, s1 and s2, configured using the LOCAL_TRANSACTION connection c1 in a single transaction context, both s1 and s2 must either use the same dynamic credentials at run time or the default configured credentials provided at design time. For more information about providing dynamic user credentials for a service's associated connection, see "Changing the User Credentials of a Service's Associated Connection at Run Time".

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

The following example shows 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
```

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

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

Explicit Transaction Usage Cases

To include multiple local transactions in a single flow, you must explicitly start and end each transaction except the first one.

Depending on what the flow needs to accomplish, you may explicitly start and end XAResource transactions as well. This way, you can create a flow that includes multiple local transactions and multiple XAResource transactions.

For example, the following flow includes a local transaction nested within another local transaction:

```
BEGIN FLOW                      // start transaction 1
.
.
.
    INVOKE startTransaction(2)      // start transaction 2
    .
    .
    .
    INVOKE commitTransaction(2)     // commit transaction 2
END FLOW                        // commit transaction 1
```

A nested transaction must adhere to the same rules that apply to container-manager transactions. That is, a nested transaction can contain one of the following:

- One local transaction, interacting with one resource
- One or more XAResource transactions, where each transaction can interact with one or more resources
- One or more XAResource transactions and one local transaction

Following are some examples of explicit transactions.

Two Local Transactions

To make this flow work properly, explicitly start and commit the nested local transaction, using the startTransaction and commitTransaction services as follows:

```
BEGIN FLOW                     // start transaction 1
INVOKE interactWithResourceA   // service for transaction 1
   INVOKE startTransaction(2)     // start transaction 2
   INVOKE interactWithResourceB   // service for transaction 2
   INVOKE commitTransaction(2)    // commit transaction 2
END FLOW                       // commit transaction 1
```

The flow executes as follows:

1. When interactWithResourceA is invoked, Integration Server starts transaction 1 and enlists ResourceA.
2. Transaction 2 executes as follows:
    a. When startTransaction(2) is invoked, Integration Server starts a new, nested transaction.
    b. When interactWithResourceB is invoked, ResourceB is enlisted in transaction 2.
    c. When commitTransaction(2) is invoked, the connection to ResourceB is closed, and transaction 2 is committed. At this point, only the work done on ResourceB is committed; transaction 1 is still open, and the work done with ResourceA is not yet committed.

3. When the flow ends, Integration Server closes the connection for transaction 1 and commits its work to ResourceA.

Note: Each transaction is a separate unit of work. Transaction 1 could be rolled back (or the commit could fail), while transaction 2 remains committed (or vice versa).

Alternatively, to achieve the same result, you can explicitly start transaction 1 before the adapter service is invoked, and explicitly commit it as follows:

```
BEGIN FLOW
INVOKE startTransaction(1)     // start transaction 1
INVOKE interactWithResourceA   // service for transaction 1
INVOKE startTransaction(2)     // start transaction 2
INVOKE interactWithResourceB   // service for transaction 2
INVOKE commitTransaction(2)    // commit transaction 2
INVOKE commitTransaction(1)    // commit transaction 1
END FLOW
```

Two XAResource Transactions

The following flow includes two XAResource transactions: one that interacts with ResourceA, and a nested transaction that interacts with ResourceB and ResourceC.

```
BEGIN FLOW                      // start transaction 1
INVOKE interactWithResourceA    // service for transaction 1
   INVOKE startTransaction(2)      // start transaction 2
   INVOKE interactWithResourceB    // service for transaction 2
   INVOKE interactWithResourceC    // service for transaction 2
   INVOKE commitTransaction(2)     // commit transaction 2
END FLOW                        // commit transaction 1
```

The flow executes as follows:

1. When interactWithResourceA is invoked, Integration Server starts transaction 1 and enlists ResourceA.
2. Transaction 2 executes as follows:
   a. When startTransaction(2) is invoked, Integration Server starts a new, nested transaction.
   b. When interactWithResourceB and interactWithResourceC are invoked, both resources are enlisted in transaction 2.
   c. When commitTransaction(2) is invoked, the connections to ResourceB and ResourceC are closed, and transaction 2 is committed. At this point, only the work done on ResourceB and ResourceC is committed; transaction 1 is still open, and the work done with ResourceA is not yet committed.
3. When the flow ends, Integration Server closes the connection for transaction 1 and commits its work to ResourceA.

One XAResource Transaction and Two Nested Local Transactions

The following flow includes three transactions: one XAResource transaction that interacts with two resources, and two nested local transactions that interact with one resource each.

```
BEGIN FLOW                          // start XAResource transaction 1
INVOKE interactWithXAResourceA      // service for XAResource transaction 1
INVOKE interactWithXAResourceB      // service for XAResource transaction 2
   INVOKE startTransaction(2)         // start local transaction 1
   INVOKE interactWithLocalResourceA  // service for local transaction 1
   INVOKE commitTransaction(2)        // commit local transaction 1
   INVOKE startTransaction(3)         // start local transaction 2
   INVOKE interactWithLocalResourceB  // service for local transaction 2
   INVOKE commitTransaction(3)        // commit local transaction 2
END FLOW                            // commit XAResource transaction 1
```

The flow executes as follows:

1. When interactWithXAResourceA is invoked, Integration Server starts transaction 1 and enlists XAResourceA.
2. When interactWithXAResourceB is invoked, Integration Server enlists XAResourceB in transaction 1.
3. Transaction 2 is executed as follows:
   a. When startTransaction(2) is invoked, Integration Server starts a new, nested transaction.
   b. When interactWithLocalResourceA is invoked, LocalResourceA is enlisted in transaction 2.
   c. When commitTransaction(2) is invoked, the connection to LocalResourceA is closed, and transaction 2 is committed. At this point, only the work done on LocalResourceA is committed; transaction 1 is still open, and the work done with XAResourceA and XAResourceB is not yet committed.
4. Transaction 3 is executed as follows:
   a. When startTransaction(3) is invoked, Integration Server starts a new, nested transaction.
   b. When interactWithLocalResourceB is invoked, LocalResourceB is enlisted in transaction 3.
   c. When commitTransaction(3) is invoked, the connection to LocalResourceB is closed, and transaction 3 is committed. At this point, only the work done on LocalResourceA and LocalResourceB is committed; transaction 1 is still open, and the work done with XAResourceA and XAResourceB is not yet committed.
5. When the flow ends, Integration Server closes the connection for transaction 1 and commits its work to XAResourceA and XAResourceB.

One XAResource Transaction and One Nested Local and XAResource Transaction

The following flow includes two transactions: one XAResource transaction that interacts with two resources, and one nested transaction that interacts with one local resource and one XAResource.

```
BEGIN FLOW                          // start XAResource transaction 1
INVOKE interactWithXAResourceA      // service for XAResource transaction 1
INVOKE interactWithXAResourceB      // service for XAResource transaction 2
   INVOKE startTransaction(2)           // start transaction 2
   INVOKE interactWithLocalResourceA    // service for transaction 2
   INVOKE interactWithXAResourceC       // service for transaction 2
   INVOKE interactWithLocalResourceA    // service for transaction 2
   INVOKE commitTransaction(2)          // commit transaction 2
END FLOW                            // commit XAResource transaction 1
```

The flow executes as follows:

1. When interactWithResourceA is invoked, Integration Server starts an XAResource transaction 1 and enlists ResourceA.
2. When interactWithResourceB is invoked, Integration Server enlists ResourceB in transaction 1.
3. Transaction 2 is executed as follows:
   a. When startTransaction(2) is invoked, Integration Server starts a new, nested transaction.
   b. When interactWithLocalResourceA is invoked, LocalResourceA is enlisted in transaction 2.
   c. When interactWithXAResourceC is invoked, XAResourceC is enlisted in transaction 2.
   d. When interactWithLocalResourceA is invoked, LocalResourceA is enlisted in transaction 2.
   e. When commitTransaction(2) is invoked, the connection to both resources of transaction 2 is closed, and transaction 2 is committed. At this point, only the work done on LocalResourceA and XAResourceC is committed; transaction 1 is still open, and the work done with XAResourceA and XAResourceB is not yet committed.
4. When the flow ends, Integration Server closes the connection for transaction 1 and commits its work to XAResourceA and XAResourceB.

Built-In Services for Transaction Management

Use the following built-in services to manage explicit transactions and set the transaction timeout interval for implicit and explicit transactions.

pub.art.transaction:commitTransaction

Commits an explicit transaction.

Input Parameters

commitTransactionInput **Document List** Information for each commit request.

Key	Description
transactionName	**String** The name of an explicit transaction that you want to commit. The *transactionName* must have been previously used in a call to pub.art.transaction:startTransaction. 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.

Usage Notes

This service must be used in conjunction with the pub.art.transaction:startTransaction service. If the *transactionName* parameter was not provided in a prior call to pub.art.transaction:startTransaction, a run-time error will be returned.

pub.art.transaction:rollbackTransaction

Rolls back an explicit transaction.

Input Parameters

rollbackTransactionInput **Document List** Information for each rollback request.

Key	Description

transactionName	**String** The name of an explicit transaction that you want to roll back. The *transactionName* must have been previously used in a call to pub.art.transaction:startTransaction. 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.

Usage Notes

This service must be used in conjunction with the pub.art.transaction:startTransaction service. If the given *transactionName* parameter was not provided in a prior call to pub.art.transaction:startTransaction, a run-time error will be returned.

pub.art.transaction:setTransactionTimeout

Manually sets a transaction timeout interval for implicit and explicit transactions.

Input Parameters

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

Output Parameters

None.

Usage Notes

This service is available only if your adapter supports built-in transaction management services, which you can confirm by checking the user guide for the adapter.

When you use this service, you are temporarily overriding the Integration Server transaction timeout interval.

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 transacted operations are rolled back.

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

pub.art.transaction:startTransaction

Starts an explicit transaction.

Input Parameters

startTransactionInput

Document List Information for each start transaction request.

Key	Description
transactionName	**String** Optional. Specifies the name of the transaction to be started. 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 transaction name.

Output Parameters

startTransactionOutput

Document List Information for each start transaction request.

Key	Description
transactionName	**String** The name of the transaction the service just started.

Usage Notes

This service is available only if your adapter supports built-in transaction management services, which you can confirm by checking the user guide for the adapter.

This service is intended for use with the pub.art.transaction:commitTransaction or pub.art.transaction:rollbackTransaction service. The *transactionName* value returned by a call to this service can be provided to pub.art.transaction:commitTransaction (to commit the transaction) or pub.art.transaction:rollbackTransaction (to roll back the transaction).

Changing the Integration Server Transaction Timeout Interval

The Integration Server default transaction timeout is no timeout (NO_TIMEOUT). To change the server's transaction timeout interval, add the watt.art.tmgr.timeout parameter to the server.cnf file.

The watt.art.tmgr.timeout 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 roll back all registered operations regardless of the execution status.

For more information about modifying the server.cnf file, see *webMethods Integration Server Administrator's Guide*.

To change the server's transaction timeout level

1. Shut down Integration Server.
2. Open the server.cnf file in a text editor.
3. Add the following parameter to the server.cnf file:

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

   where *TransactionTimeout* is the number of seconds before transaction timeout.
4. Restart Integration Server.

3 The Adapter Runtime in a Clustered Environment

What is Clustering?

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.

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 adapters use the caching mechanism that is enabled on Integration Server. Adapters do not explicitly implement any clustering or caching beyond what is already provided by 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.

For details on Integration Server clustering, see *webMethods Integration Server Clustering Guide*.

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*.

The following considerations and requirements apply to Adapter Runtime based adapters 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	One Integration Server in the cluster cannot be version 9.0 and another Integration Server in the cluster be version 9.5.
Adapter packages	All adapter packages on one Integration Server should be replicated to all other Integration Servers in the cluster.
Adapter versions	All Integration Servers in the cluster must have the same version of an adapter, with the same fixes (updates and service packs) applied.
Adapter connections	If you configure a connection to the back end, 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 40.
Adapter services	If you configure a specific adapter service, this 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.
Adapter notifications	If you configure a specific adapter notification, this notification must appear on all servers in the cluster. For more information about adapter notifications in a cluster, see "Polling Notification Support in a Cluster" on page 41 and "Adapter Listener Support in a Cluster" on page 46.

Replicating Packages to Integration Servers

Every Integration Server in the cluster should contain an identical set of packages that you define using a specific adapter. You should replicate the adapter services, the connections they use, and the adapter notifications.

To ensure consistency, Software AG recommends creating all packages on one server and replicating 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 more information about replicating packages, see *webMethods Integration Server Administrator's Guide*.

Considerations when Configuring Connections with Connection Pooling Enabled

When you configure a 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 to the back end.

For example, if you have a cluster of two Integration Servers with a connection configured to a back end resource 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 the back end.

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 more information about connection pools, see *webMethods Integration Server Administrator's Guide*.

Disabling the Redirection of Administrative Services

An Integration Server instance that cannot handle a client's service request can automatically redirect the request to another server in the cluster. However, adapters use 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 services for an adapter, 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.
2. Open 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.

Polling Notification Support in a Cluster

The Adapter Runtime enables the coordinated execution of polling notifications within an Integration Server cluster. It provides the ability to enable multiple instances of the same polling notification in your cluster, and to coordinate their schedules and execution. This provides enhanced quality of service by allowing configurations for automated failover between notifications and distributed processing of polling notifications.

Important: Adapters support enabling the same polling notification on multiple Integration Server instances connecting to the same back end to achieve automated failover, *only* when the multiple Integration Servers share the same ISInternal database. If you attempt to use the same polling notification on multiple Integration Servers pointing to the same back end but using separate ISInternal databases, you may encounter abnormal results.

Beginning with Integration Server 9.0, adapters use Integration Server Scheduler to support polling notifications. On enabling a polling notification, a new Integration Server scheduled task is created, which polls the back end resource at the given interval. Do not manually edit or change scheduled tasks. Each polling notification creates an Integration Server scheduled task. When a notification is disabled, the scheduled task in Integration Server is removed.

Considerations for Polling Notifications Executing via Scheduled Tasks

With polling notifications executing via scheduled tasks, ensure that:

- Each notification is present in all cluster nodes at all times.
- The Overlap function for the polling notifications is disabled.
- Polling notifications names do not exceed 400 characters.

- The value of the Integration Server watt.server.scheduler.threadThrottle property should not be lower than the number of total polling notifications and scheduled tasks. By default the value is 75% of the total threads.
- The IS Internal functional alias (specified on the Settings > JDBC Pools screen) is configured with a database.

Note: You can make scheduled notification tasks visible in the Server > Scheduler page in Integration Server Administrator by setting `watt.pkg.art.scheduler.notificationtask.display=true`

If the parameter is not shown, add it.

Configuring this property is required only for debugging or for editing the polling notification schedule interval.

Configuring Polling Notifications in Standby or Distributed Mode on Integration Server 8.2

When using an adapter on Integration Server 8.2, you can configure a polling notification to run in either Standby or Distributed mode. You can also configure additional settings for clustered polling notifications.

Standby Mode and Distributed Mode

In Standby mode, a particular instance of a polling notification will execute the notification according to its configured schedule. When you start the cluster, the polling notification that executes the first scheduled run is considered to be the primary notification. This instance will continue to execute the scheduled run as long as it is enabled and fully functional. If at any time this notification becomes disabled, another notification in the cluster will assume control. The notification that assumes control is arbitrary. After a notification has control, it will continue to execute the schedule for as long as it is enabled and fully functional.

In Distributed mode, any instance of the polling notification can execute the currently scheduled run. The notification that executes the current scheduled run is arbitrary. If a notification does not complete executing within the amount of time specified in the **Max Process Time** field, the system considers that notification to be "dead". Another enabled instance in the cluster will recognize this situation and will attempt to execute the scheduled run. For details about **Max Process Time**, see "Notification-Specific Settings" on page 44.

Configuration Settings

Cluster coordination is controlled by a number of configuration settings to control polling notification behavior and tune failure detection using timeouts. All settings that pertain to clustered polling notifications are ignored or disabled until you include the server in a cluster.

Global Settings

The following settings are set in the server.cnf file and apply globally to all clustered notifications for all adapters. The server uses the settings in an algorithm that determines whether a polling notification instance should be considered “dead”. For more information, see *webMethods Integration Server Clustering Guide*.

Global Setting Name	Values and Description
watt.art.clusteredPollingNotification.keepAliveInterval	The frequency, in milliseconds, with which a secondary instance of a polling notification will check to see if an executing instance is still alive. If the property is not set, the secondary instance will change to the default maxLockDuration value of "180000" for the shared cache.
watt.art.clusteredPollingNotification.keepAliveExpireTimeout	The amount of time, in milliseconds, that an executing node can be late before it is assumed to have failed. In general, this setting should be equal to the amount of drift anticipated on the server clocks. If not set, the secondary instance will change to the default maxLockDuration value of "180000" for the shared cache.

Adapter-Specific Settings

The adapter-specific settings apply to all the polling notifications in your adapter.

Within the configuration directory of the adapter's package, the clusterProperties.cnf file provides settings that specify a callback scheme, and place limits on which coordination modes can be applied to notification nodes for the adapter. The clusterProperties.cnf file is an XML file in which settings may be provided globally for the adapter or specifically to a particular notification template.

The following example includes all of the major constructs of a clusterProperties.cnf file.

```
<?xml version="1.0"?>
<clusterProps>
  <pollingNotifications>
    <callbackScheme>1</callbackScheme>
    <runtimeModeLimit>distribute</runtimeModeLimit>
    <template className="com.wm.adapter.wmarttest.notification.LatchedPollingNot
    ification">
       <callbackScheme>1</callbackScheme>
       <runtimeModeLimit>standby</runtimeModeLimit>
    </template>
  </pollingNotifications>
```

```
  <listenerNotifications>
    <callbackScheme>1</callbackScheme>
  </listenerNotifications>
  <listeners>
    <runtimeModeLimit>standby</runtimeModeLimit>
  </listeners>
</clusterProps>
```

The <callbackScheme> setting controls how callback coordination is performed, while <runtimeModeLimit> constrains the coordination mode setting that can be set for a notification node. Valid values for these settings are included in the following tables.

Table 1. Values for callbackScheme Setting for Polling Notifications

When callbackScheme is set to...	The following Coordination Modes are set:		
	Enable/Disable	Startup/Shutdown	Resume/Suspend
0	No coordination	No coordination	No coordination
1(default)	Coordinated	No coordination	No coordination
2	No coordination	Coordinated	No coordination
3	Coordinated	Coordinated	No coordination
4	No coordination	No coordination	Coordinated
5	No coordination	Coordinated	Coordinated
6	Coordinated	No coordination	Coordinated
7	Coordinated	Coordinated	Coordinated

Notification-Specific Settings

The notification-specific settings enable you to configure certain scheduling aspects of polling notifications on an individual basis.

Two new fields appear on the Polling Notification Schedule page: **Coordination Mode** and **Max Process Time**. These fields become editable when you add your Integration Server to a cluster.

- The **Coordination Mode** field controls the coordination of the notification schedules across the cluster. Depending on the value you assigned to the **runtimeModeLimit** setting (see "Adapter-Specific Settings" on page 43), the adapter user can select some combination of the following values in the **Coordination Mode** field as follows:

This runtimeModeLimit value...	Displays these values in the Coordination Mode field...
disable	**disable**
standby	**disable** and **standby**
distribute	**disable**, **standby**, and **distribute**

- The **Max Process Time** field enables other notifications to determine whether a currently executing notification should be considered to be "dead". If a notification executes a scheduled run and it fails to complete before the **Max Process Time**, then another notification instance will consider it dead; this other notification will assume control and execute a scheduled run. The default value is equal to the value in the watt.art.clusteredPollingNotification.keepAliveExpireTimeout setting in the server.cnf file.

  If the **Max Process Time** setting is not high enough, you may encounter a situation in which a notification is executing normally, but another notification assumes it is "dead". When the original notification completes, it will recognize that it was prematurely considered "dead". In this case, the system logs an Illegal Overlap exception with message ID [ART.116.3715]. If this exception occurs, increase your **Max Process Time** setting.

  When setting the value of **Max Process Time**, you should allow for "clock drift". For details, see on page 45"Clock Synchronization".

If you want to update the schedule and settings of a notification in a cluster, all notification instances in the cluster must be suspended or disabled for the changes to be saved. If any notification instance in the cluster is enabled, the adapter will not save the updates.

If all instances of a notification in the cluster do not have the same settings, the notification that became active first will have precedence.

Clock Synchronization

To determine whether a notification has failed, notifications use the system clocks of the machines that host the clustered Integration Servers. Synchronizing the clocks of all machines in the cluster is critical for the proper execution of clustered polling notifications.

However, in time these clocks might become un-synchronized. Therefore you should anticipate the effect of "clock drift" when you establish values for the **keepAliveExpireTimeout** server-specific setting and **Max Process Time** notification-specific setting. Clock drift is the time difference between the clocks. As a guideline, add to the **keepAliveExpireTimeout** and the **Max Process Time** settings two times the maximum clock drift you anticipate.

Configuring Adapter Notification Schedules in a Clustered Environment

To enable adapter polling notifications to execute in distributed or standby mode

1. In the cluster, shut down the Integration Server you are configuring.
2. Open the clusterProperties.cnf file for the adapter.
3. Change all `<runtimeModeLimit>` values to `<distribute>` or `<standby>`.

   For an example of a sample clusterProperties.cnf file, see "Adapter-Specific Settings" on page 43.
4. Save the file and restart Integration Server.
5. Start Integration Server.
6. Select **Adapters > *Adapter name***.
7. From the navigation area, select **Polling Notifications**.
8. For each notification:
   a. Disable the notification.
   b. Click the **Edit Schedule** icon.
   c. Set the **Coordination Mode** to **Distributed** or **Standby**, as appropriate for the notification.
   d. Enable the notification.

Important: To maintain duplicate detection and ordering, your polling notification schedules must not run with the **Overlap** option selected. To access the **Overlap** option, click the **Edit Schedule** icon.

After you configure a polling notification, you may propagate all the affected components across your cluster. Changing the polling notification schedule from Integration Server Administrator or changing the polling notification settings in Software AG Designer will require you to propagate the polling notification across the cluster. If you made changes to the settings in server.cnf or to the clusterProperties.cnf file, you must also propagate the changes across the cluster.

Adapter Listener Support in a Cluster

Beginning with Integration Server 9.5, an adapter listener can be active either on multiple nodes or on a single node in an Integration Server cluster.

If the adapter back end supports the non-duplication of messages to which multiple clients are subscribed, the listener is *multi-node*. It is active on all nodes in the Integration Server cluster. In this case, each node retrieves and processes a different message.

If the adapter back end does not support the non-duplication of messages to which multiple clients are subscribed, the listener is *single-node*. It is active on a single node in the cluster - the active or primary node - thus avoiding duplicate messages. If the active node goes down, another node in the cluster becomes active, providing failover support. The active node stores incoming events from the back end in the distributed (shared) cache. In this way, message processing continues when the active node goes down.

Important: If message order is important when the adapter processes events from the back end, use a combination of a single-threaded and single-node listener, or a single-threaded and multi-node listener that is enabled on only one node in the cluster. For information about how multi-threaded listeners work, see "Single-Threaded and Multi-Threaded Listeners" on page 18.

You can enable, disable, or suspend a listener on a single node or on all nodes in a cluster by changing the listener state on one of the nodes. For information about changing listener states in a cluster, see "Enabling Listeners in a Cluster" on page 50 and "Disabling Listeners in a Cluster" on page 50.

Listener States in a Cluster

Both multi-node and single-node listeners have the following states in an Integration Server cluster: enabled, disabled, and suspended. When you enable or disable a listener on a node in the cluster, a listener thread is created or destroyed, respectively, in the node.

Multi-Node Listener States

You can change the state of a multi-node listener on all nodes in an Integration Server cluster from any node in the cluster. When a multi-node listener is enabled, disabled, or suspended on all nodes in the cluster, the action that a particular node takes depends on:

- the selected state change on all nodes
- the listener state on the particular node, that is if the listener thread is running or not

For example, if a multi-node listener has already been suspended on node A, and then on node B you disable the listener on all nodes in the cluster, node A takes no real action. Node A only shows the state as disabled because the listener thread has already stopped running.

The following table shows what change occurs in the state of a multi-node listener on a particular node in the cluster when you enable, disable, or suspend the listener on all nodes.

Listener State on a Particular Node	Enable Listener on All Nodes	Disable Listener on All Nodes	Suspend Listener on All Nodes
Enabled	No action	Disable	Suspend
Disabled	Enable	No action	No action
Suspended	Enable	Show as disabled. No action	No action

Single-Node Listener States

You can change the state of a single-node listener on all nodes in an Integration Server cluster from any node in the cluster. When a single-node listener is enabled, disabled, or suspended on all nodes in the cluster, the action that a particular node takes depends on:

- the selected state change on all nodes
- the listener state on the particular node, that is if the listener thread is running or not
- whether the particular node is the active (primary) node

For example, if a single-node listener has already been suspended on node A, and then on node B you enable the listener on all nodes in the cluster, node A can:

- enable the listener, if node A becomes the active node in the cluster.
- show the listener as enabled without creating a new thread, if another node has already become the active one.

The following table shows what change occurs in the state of a single-node listener on a particular node in the cluster when you enable, disable, or suspend the listener on all nodes.

Listener State on a Particular Node	Enable Listener on All Nodes	Disable Listener on All Nodes	Suspend Listener on All Nodes
Enabled	No action	■ Disable the listener if this is the active node ■ Show the listener as disabled if this is not the active node. No action	■ Suspend the listener if this is the active node ■ Show the listener as suspended if this is not the

Listener State on a Particular Node	Enable Listener on All Nodes	Disable Listener on All Nodes	Suspend Listener on All Nodes
			active node. No action
Disabled	■ Enable the listener if the node acquires the lock on the shared cache and becomes the active node ■ Show the listener as enabled if another node has already become the active node. No action	No action	No action
Suspended	■ Enable the listener if the node acquires the lock on the shared cache and becomes the active node ■ Show the listener as enabled if another node became the active node. No action	Show the listener as disabled. No action	No action

If a single-node listener is active on a particular node and you try to disable it, another Integration Server node's listener will become active if that listener is in an enabled state. If the listener is in a disabled state on all other Integration Server nodes, those nodes will ignore the disable operation.

Enabling, Disabling, and Suspending Listeners in a Cluster

In an Integration Server cluster, you can enable, disable, and suspend adapter listeners using Integration Server Administrator.

For information about what happens in a particular node when you change the state of a multi-node or a single-node listener on all nodes in the cluster, see "Listener States in a Cluster" on page 47.

Enabling Listeners in a Cluster

To enable a multi-node or single-node listener in a cluster

1. In Integration Server Administrator, select **Adapters > *Adapter name***.
2. In the ***Adapter*** menu, select **Listeners**. The Listeners screen appears.
3. Select **Enabled** from the drop-down list in the **State** field for the listener that you want to enable. A dialog box appears prompting you to enable the listener.
4. Do one of the following:
   - Select **Yes for all nodes** to enable the listener on all nodes in the cluster.
   - Select **Yes** to enable the listener on the current node only.
   - Select **Cancel** to end the operation.

Disabling Listeners in a Cluster

To disable a multi-node or single-node listener in a cluster

1. In Integration Server Administrator, select **Adapters > *Adapter name***.
2. In the ***Adapter*** menu, select **Listeners**. The Listeners screen appears.
3. Select **Disabled** from the drop-down list in the **State** field for the listener that you want to disable. A dialog box appears prompting you to disable the listener.
4. Do one of the following:
   - Select **Yes for all nodes** to disable the listener on all nodes in the cluster.
   - Select **Yes** to disable the listener on the current node only.
   - Select **Cancel** to end the operation.

Suspending Listeners in a Cluster

To suspend a multi-node or single-node listener in a cluster

1. In Integration Server Administrator, select **Adapters > *Adapter name***.
2. In the ***Adapter*** menu, select **Listeners**. The Listeners screen appears.
3. Select **Suspended** from the drop-down list in the **State** field for the listener that you want to suspend. A dialog box appears prompting you to suspend the listener.
4. Do one of the following:
   - Select **Yes for all nodes** to suspend the listener on all nodes in the cluster.

- Select **Yes** to suspend the listener on the current node only.
- Select **Cancel** to end the operation.

4 Adapter Runtime Logging and Exception Handling

Overview

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

Adapter Runtime Message Logging

Integration Server maintains several types of logging. However, the Adapter Runtime logs messages only to the error, server, and service logs.

The following table lists the logging types supported by Integration Server for the Adapter Runtime.

Logger	Description
Error	Provides stack trace information about all errors that occur in the Adapter Runtime, including exceptions thrown by services. Adapters automatically post fatal-level and error-level messages to the Adapter Runtime error log.
Server	Provides information about fatal-level through debug-level log messages. Trace-level log messages appear as messages for an individual adapter.
Service	Provides information about adapter services. You can monitor adapter services as you would audit any service in Integration Server. The audit properties for an adapter service are available in each adapter service template in the Properties panel.

The Adapter Runtime log messages appear in the following format, ART.*mmmm .nnnn* , where:

- ART is the facility code indicating that the message is from the Adapter Runtime.
- *mmmm* is the code for one of the following Integration Server log facilities:

Facility		Information and errors related to...
0114	Adapter Runtime	Adapter runtime facilities.

Facility		Information and errors related to...
0115	Adapter Runtime (Listener)	Adapter listeners that use adapter connections to connect to adapter resources.
0116	Adapter Runtime (Notification)	Adapter notifications including polling and listener notifications.
0117	Adapter Runtime (Adapter Service)	Adapter services that define operations that the adapter will perform on adapter resources.
0118	Adapter Runtime (Connection)	Adapter connections that contain parameters that adapter notifications and listeners use to connect to an adapter resource.
0121	Adapter Runtime (SCC Transaction Manager)	Adapter Runtime (JCA System Contract Component Transaction Manager).
0126	Adapter Runtime (SCC Connection Manager)	Adapter Runtime (JCA System Contract Component Connection Manager).

- *nnnn* represents the error's minor code.

Configuring Server Logging Levels for the Adapter Runtime

To specify the amount and type of information to include in the server log for the Adapter Runtime

1. In Integration Server Administrator, go to **Settings > Logging**.
2. In the **Logger List**, select **Server Logger > Edit Server Logger**.

   The **Server Logger Configuration** area lists Integration Server and products that are installed on Integration Server, and the facilities for each of these. To see the Adapter Runtime facilities and their current logging levels, expand the Integration Server node.

   By default, all products inherit the logging level of the Default node. Inherited values are shown in gray text. When you explicitly change the logging level for a product or facility, that level overrides the Default node level.

3. In the Integration Server node, select the level of logging you want to use from the **Logging Level** list for each Adapter Runtime facility.

   For more information about logging levels, see *webMethods Integration Server Administrator's Guide*.

   Important: Recording more information consumes more system resources.

4. Click **Save Changes**.

Adapter Runtime Exception Handling

The Adapter Runtime provides the following exception definitions:

- **AdapterException.** All exceptions thrown by an adapter, excluding exceptions during adapter service execution, belong to this exception type or to a child of this exception type.
- **DetailedException.** This type of exception is a child of AdapterException. It contains detailed information about an exception.
- **AdapterConnectionException.** This exception is thrown when an issue occurs while establishing a connection to the back end or using a connection that has become stale. When the system returns an AdapterConnectionException WmART resets the connection pool.

  All adapter services, notifications, and listeners use the connection pool. If an AdapterConnectionException occurs when an adapter service, notification, or listener tries to retrieve a connection from the connection pool, the Adapter Runtime treats the exception as a fatal one. WmART resets the connection pool and throws a DetailedSystemException.
- **DetailedServiceException.** This type of exception provides detailed information about any exceptions thrown during service execution.

A Adapter Runtime Built-In Services Reference

Summary of Adapter Runtime Built-In Services

Use the built-in services in the WmART package to manage adapter components, including connections, adapter services, listeners, and notifications.

The following services are available in the WmART package:

Element	Description
pub.art:listRegisteredAdapters	Returns the display name and adapter type name of all registered adapters.
pub.art.connection:disableConnection	Disables a connection node.
pub.art.connection:enableConnection	Enables an existing connection node.
pub.art.connection:getConnectionStatistics	Returns current usage statistics for a connection node.
pub.art.connection:listAdapterConnections	Lists connection nodes associated with a specified adapter.
pub.art.connection:queryConnectionState	Returns the current connection state (enabled/disabled) and error status for a connection node.
pub.art.connection:getInterruptedThreadStatus	Returns the status of the threads which are interrupted and not responding to interrupt mechanism.
pub.art.listener:disableListener	Disables a listener.
pub.art.listener:enableListener	Enables an existing listener.
pub.art.listener:listAdapterListeners	Lists listeners associated with a specified adapter.

Element	Description
pub.art.listener:queryListenerState	Returns the current state for a listener.
pub.art.listener:resumeListener	Resumes a specified listener.
pub.art.listener:setListenerNodeConnection	Changes the connection node used by a specified listener.
pub.art.listener:suspendListener	Suspends a specified listener.
pub.art.notification:disableListenerNotification	Disables a listener notification.
pub.art.notification:disablePollingNotification	Disables a polling notification.
pub.art.notification:disablePublishEvents	Disables the publishing of events for an adapter notification.
pub.art.notification:enableListenerNotification	Enables an existing listener notification.
pub.art.notification:enablePollingNotification	Enables an existing polling notification.
pub.art.notification:enablePublishEvents	Enables the publishing of events for an adapter notification.
pub.art.notification:listAdapterListenerNotifications	Lists the listener notifications associated with a specified adapter.
pub.art.notification:listAdapterPollingNotifications	Lists the polling notifications associated with a specified adapter.

Element	Description
pub.art.notification:queryListenerNotificationState	Returns the current state (enabled/disabled) for a listener notification.
pub.art.notification:queryPollingNotificationState	Returns the current state for a polling notification.
pub.art.notification:resumePollingNotification	Resumes a specified polling notification node.
pub.art.notification:setListenerNotificationNodeListener	Changes the listener used by a specified listener notification.
pub.art.notification:setPollingNotificationNodeConnection	Changes the connection node used by a specified polling notification.
pub.art.notification:suspendPollingNotification	Suspends a specified polling notification.
pub.art.service:listAdapterServices	Lists adapter services associated with a specified adapter.
pub.art.service:setAdapterServiceNodeConnection	Changes the connection node used by a specified adapter service.
pub.art.transaction:commitTransaction	Commits an explicit transaction.
pub.art.transaction:rollbackTransaction	Rolls back an explicit transaction.
pub.art.transaction:setTransactionTimeout	Manually sets a transaction timeout interval for implicit and explicit transactions.
pub.art.transaction:startTransaction	Starts an explicit transaction.

pub.art:listRegisteredAdapters

Returns the display name and adapter type name of all registered adapters.

Input Parameters

None.

Output Parameters

registeredAdapterList **Document List** Information for each adapter registered with the WmART package.

Key	Description
adapterDisplayName	**String** The localized name that Integration Server Administrator displays.
adapterTypeName	**String** The name of the adapter as registered with the WmART package. This value can be used as input for the inventory services that take *adapterTypeName* as input.

pub.art.connection:disableConnection

Disables a connection node.

Input Parameters

connectionAlias **String** Name of the connection node you want to disable.

Output Parameters

None.

pub.art.connection:enableConnection

Enables an existing connection node.

Input Parameters

connectionAlias	**String** Name of the connection node you want to enable.

Output Parameters

None.

pub.art.connection:getConnectionStatistics

Returns current usage statistics for a connection node.

Input Parameters

aliasName	**String** Name of the connection node for which you want usage statistics returned.

Output Parameters

connectionStatistics **Document List** Information for each connection node.

Key	Description
TotalConnections	**Integer** Current number of connection instances.
BusyConnections	**Integer** Number of connections currently in use by services, notifications, and listeners.
FreeConnections	**Integer** Total number of connections created and available for use.
TotalHits	**Integer** Number of times this connection node successfully provided connections since the last reset.
TotalMisses	**Integer** Number of times this connection node unsuccessfully provided connections since the last reset (when the request timed out).

pub.art.connection:listAdapterConnections

Lists connection nodes associated with a specified adapter.

Input Parameters

adapterTypeName **String** The name of the adapter as registered with the WmART package.

Output Parameters

connectionDataList **Document List** Information for each connection node registered with the specified adapter.

Key	Description
connectionAlias	**String** The name of the connection node.
packageName	**String** The name of the package in which the connection node resides.
connectionState	**String** Current state of the connection node. The state will have one of these values: - `disabled` - Connection node is disabled - `enabled` - Connection node is enabled. - `shuttingdown` - Connection node is in the process of shutting down. - `unknown` - Connection node is registered but has not yet established its state.

pub.art.connection:queryConnectionState

Checks the availability of the underlying resource (for example, database servers) at frequent time intervals, and returns the current connection state (enabled/disabled, shuttingdown, pendingEnabled) and error status for a connection node.

Input Parameters

connectionAlias	**String** Name of the connection node for which you want the connection state and error status returned.

Output Parameters

connectionState	**String** Current connection state (enabled/disabled, shuttingdown, pendingEnabled).
hasError	**Boolean** Flag indicating if any error was detected on connection. The values are: ■ `true` if an error was detected. ■ `false` if no error was detected.
lastErrorTime	**String** The long value of the time stamp for the last error.

Usage Notes

You use the pub.art.connection:queryConnectionState service together with the watt.art.wmConnectionPool.pingRetryInterval and watt.art.wmConnectionPool.pingSafeInterval parameters to monitor the state of the underlying resource.

pub.art.connection:getInterruptedThreadStatus

Returns the list of connection threads which are not responding even after being interrupted by the connection pool interrupter.

Input Parameters

aliasName	**String** Name of the connection node for which you want to get the status of threads that are not responding to the interrupt.

Output Parameter

None.

Usage Notes

We recommend your manual intervention for the server threads which gets hung while creating or destroying the connections and also not responding to the interrupt mechanism.

pub.art.listener:disableListener

Disables a listener.

Input Parameters

listenerName	**String** Name of the listener you want to disable. The listener should have a state of `enabled` or `suspended`.
forceDisable	**String** Optional. Flag to disable the listener regardless of whether it is still waiting for data from a back-end resource. The string may have one of these values: ■ `true` to disable the listener. ■ `false` to keep the listener enabled.

Output Parameters

None.

pub.art.listener:enableListener

Enables an existing listener.

Input Parameters

listenerName	**String** Name of the listener you want to enable.

Output Parameters

None.

Usage Notes

If you do not enable the connection resource associated with the listener, this service will return without performing any action, and the listener will remain disabled. Therefore,

you should invoke on page 61"pub.art.connection:enableConnection" before calling this service to confirm that the listener has been enabled.

pub.art.listener:listAdapterListeners

Lists listeners associated with a specified adapter.

Input Parameters

adapterTypeName **String** The name of the adapter as registered with the WmART package.

Output Parameters

listenerDataList **Document List** Information for each listener registered with the specified adapter.

Key	Description
listenerNodeName	**String** The name of the listener.
packageName	**String** The name of the package in which the listener resides.
listenerEnabled	**String** Current state of the listener. The state will have one of these values: - `disabled` if the listener is disabled. - `enabled` if the listener is enabled. - `enablePending` if the listener is in the process of starting. - `disablePending` if the listener is in the process of disabling. - `suspended` if the listener is suspended. - `suspendPending` if the listener is in the process of suspending.

pub.art.listener:queryListenerState

Returns the current state for a listener.

Input Parameters

listenerName	**String** Name of the listener for which you want the current state returned.

Output Parameters

listenerState	**String** Current state of the listener. The state will have one of these values: ■ `disabled` if the listener is disabled. ■ `enabled` if the listener is enabled. ■ `enablePending` if the listener is in the process of starting. ■ `disablePending` if the listener is in the process of disabling. ■ `suspended` if the listener is suspended. ■ `suspendPending` if the listener is in the process of suspending.

pub.art.listener:resumeListener

Resumes a specified listener.

Input Parameters

listenerName	**String** The name of the suspended listener you want to resume. The service returns an error if you specify an invalid listener.

Output Parameters

None.

Usage Notes

If the requested transition is not valid (for example, trying to resume a disabled listener or a listener that is already resumed), the service ignores the request.

After you use this service, you can use on page 66“pub.art.listener:queryListenerState” to verify pub.art.listener:resumeListener correctly changed the state of the listener.

pub.art.listener:setListenerNodeConnection

Changes the connection node used by a specified listener.

Input Parameters

listenerName	**String** Name of the listener for which you want to change the connection node.
connectionAlias	**String** Name of the new connection node to use with the listener.

Output Parameters

None.

Usage Notes

Calling this service for a listener that is disabled is permitted.

Calling this service for a listener that is suspended changes the state of the listener to `disabled`. The user must enable the listener before using it.

pub.art.listener:suspendListener

Suspends a specified listener.

Input Parameters

listenerName	**String** The name of the listener you want to suspend. The service returns an error if you specify an invalid listener.

Output Parameters

None.

Usage Notes

If the requested transition is not valid (for example, trying to suspend a disabled listener or a listener that is already suspended), the service ignores the request.

After you use this service, you can use pub.art.listener:queryListenerState to verify pub.art.listener:suspendListener correctly changed the state of the listener.

pub.art.notification:disableListenerNotification

Disables a listener notification.

Input Parameters

notificationName **String** The name of the listener notification you want to disable.

Output Parameters

None.

pub.art.notification:disablePollingNotification

Disables a polling notification.

Input Parameters

notificationName **String** The name of the polling notification you want to disable. The polling notification should have a state of `enabled` or `suspended`.

Output Parameters

None.

pub.art.notification:disablePublishEvents

Disables the publishing of events for an adapter notification.

Input Parameters

notificationName **String** The name of the notification for which you want to disable the publishing of events.

Output Parameters

None.

pub.art.notification:enableListenerNotification

Enables an existing listener notification.

Input Parameters

notificationName **String** The name of the listener notification you want to enable.

Output Parameters

None.

pub.art.notification:enablePollingNotification

Enables an existing polling notification.

Input Parameters

notificationName **String** Name of the polling notification you want to enable.

Output Parameters

None.

Usage Notes

You must schedule the polling notification before you can run this service. See your adapter user documentation for instructions to schedule the polling notification.

pub.art.notification:enablePublishEvents

Enables the publishing of events for an adapter notification.

Input Parameters

notificationName **String** The name of the notification for which you want to enable the publishing of events.

Output Parameters

None.

pub.art.notification:listAdapterListenerNotifications

Lists the listener notifications associated with a specified adapter.

Input Parameters

adapterTypeName **String** The name of the adapter as registered with the WmART package.

Output Parameters

notificationDataList **Document List** Information for each listener notification registered with the specified adapter.

Key	Description
notificationNodeName	**String** The name of the listener notification.
packageName	**String** The name of the package in which the listener notification resides.
notificationEnabled	**String** The current state of the listener notification. The state will have one of these values: - `no` if the listener notification is disabled. - `yes` if the listener notification is enabled.

pub.art.notification:listAdapterPollingNotifications

Lists the polling notifications associated with a specified adapter.

Input Parameters

adapterTypeName **String** The name of the adapter as registered with the WmART package.

Output Parameters

notificationDataList **Document List** Information for each polling notification registered with the specified adapter.

Key	Description
notificationNodeName	**String** The name of the polling notification.
packageName	**String** The name of the package in which the polling notification resides.
notificationEnabled	**String** The current state of the polling notification. The state will have one of these values: ■ `no` if the polling notification is disabled. ■ `yes` if the polling notification is enabled. ■ `pending` if the polling notification is in the process of shutting down. ■ `suspended` if the polling notification is suspended.

pub.art.notification:queryListenerNotificationState

Returns the current state (enabled/disabled) for a listener notification.

Input Parameters

notificationName **String** The name of the listener notification for which you want the current state (enabled/disabled) returned.

Output Parameters

notificationState **String** The current state (enabled/disabled) for the listener notification.

pub.art.notification:queryPollingNotificationState

Returns the current state for a polling notification.

Input Parameters

notificationName **String** The name of the polling notification for which you want the current state and schedule settings returned.

Output Parameters

notificationState **String** The current state (enabled, disabled, pending disable, pending suspend, or suspended) for the polling notification.

scheduleSettings **IData** Object that contains the notification's schedule settings as follows:

Key	Description
notificationInterval	**Integer** Polling frequency of the notification.
notificationOverlap	**Boolean** Flags whether the notification can overlap. The values are: ■ `true` if the notification can overlap. ■ `false` if the notification cannot overlap.
notificationImmediate	**Boolean** Flags whether the notification can fire immediately. The values are: ■ `true` if the notification can fire immediately. ■ `false` if the notification cannot fire immediately.

pub.art.notification:resumePollingNotification

Resumes a specified polling notification node.

Input Parameters

notificationName	**String** The name of the polling notification you want to resume. The service returns an error if you specify an invalid polling notification.

Output Parameters

None.

Usage Notes

If the requested transition is not valid (for example, trying to resume a disabled polling notification or a polling notification that is already resumed), the service ignores the request.

After you use this service, you can use pub.art.notification:queryPollingNotificationState to verify pub.art.notification:resumePollingNotification correctly changed the state of the polling notification to `enabled`.

pub.art.notification:setListenerNotificationNodeListener

Changes the listener used by a specified listener notification.

Input Parameters

notificationName	**String** Name of the listener notification for which you want to change the listener.
listenerNode	**String** Name of the new listener to use with the listener notification.

Output Parameters

None.

Usage Notes

This service returns an error if the listener notification is enabled.

You can use this service for synchronous and asynchronous listener notifications.

pub.art.notification:setPollingNotificationNodeConnection

Changes the connection node used by a specified polling notification.

Input Parameters

notificationName **String** Name of the polling notification for which you want to change the connection node.

connectionAlias **String** Name of the new connection node to use with the polling notification.

Output Parameters

None.

Usage Notes

The polling notification must be in a `disabled` or `suspended` state before you call this service. This service returns an error if the polling notification is enabled.

If you use this service on a suspended polling notification, the service changes the state of the polling notification to `disabled`.

pub.art.notification:suspendPollingNotification

WmART. Suspends a specified polling notification.

Input Parameters

notificationName **String** The name of the polling notification you want to suspend. The service returns an error if you specify an invalid polling notification.

Output Parameters

None.

Usage Notes

If the requested transition is not valid (for example, trying to suspend a disabled polling notification or a polling notification that is already suspended), the service ignores the request.

After you use this service, you can use pub.art.notification:queryPollingNotificationState to verify pub.art.notification:suspendPollingNotification correctly changed the state of the polling notification to `suspended`.

pub.art.service:listAdapterServices

Lists adapter services associated with a specified adapter.

Input Parameters

adapterTypeName	**String** The name of the adapter as registered with the WmART package.

Output Parameters

serviceDataList **Document List** Information for each adapter service registered with the specified adapter.

Key	Description
serviceNodeName	**String** The name of the adapter service.
packageName	**String** The name of the package in which the adapter service resides.

pub.art.service:setAdapterServiceNodeConnection

Changes the connection node used by a specified adapter service.

Input Parameters

serviceName	**String** Name of an existing adapter service for which you want to change the connection node.

connectionAlias	**String** Name of the new connection node to use with the adapter service.

Output Parameters

None.

Usage Notes

The new connection node must be enabled before you call this service.

pub.art.transaction:commitTransaction

Commits an explicit transaction.

Input Parameters

commitTransactionInput **Document List** Information for each commit request.

Key	Description
transactionName	**String** The name of an explicit transaction that you want to commit. The *transactionName* must have been previously used in a call to pub.art.transaction:startTransaction. 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.

Usage Notes

This service is available only if your adapter supports built-in transaction management services, which you can confirm by checking the user guide for the adapter.

This service must be used in conjunction with the pub.art.transaction:startTransaction service. If the *transactionName* parameter was not provided in a prior call to pub.art.transaction:startTransaction, a run-time error will be returned.

pub.art.transaction:rollbackTransaction

Rolls back an explicit transaction.

Input Parameters

rollbackTransactionInput **Document List** Information for each rollback request.

Key	Description
transactionName	**String** The name of an explicit transaction that you want to roll back. The *transactionName* must have been previously used in a call to pub.art.transaction:startTransaction. 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.

Usage Notes

This service is available only if your adapter supports built-in transaction management services, which you can confirm by checking the adapter's user guide.

This service must be used in conjunction with the pub.art.transaction:startTransaction service. If the given *transactionName* parameter was not provided in a prior call to pub.art.transaction:startTransaction, a run-time error will be returned.

pub.art.transaction:setTransactionTimeout

Manually sets a transaction timeout interval for implicit and explicit transactions.

Input Parameters

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

Output Parameters

None.

Usage Notes

This service is available only if your adapter supports built-in transaction management services, which you can confirm by checking the user guide for the adapter.

When you use this service, you are temporarily overriding the Integration Server transaction timeout interval.

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 transacted operations are rolled back.

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

pub.art.transaction:startTransaction

Starts an explicit transaction.

Input Parameters

startTransactionInput **Document List** Information for each start transaction request.

Key	Description
transactionName	**String** Optional. Specifies the name of the transaction to be started. 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 transaction name.

Output Parameters

startTransactionOutput **Document List** Information for each start transaction request.

Key	Description
transactionName	**String** The name of the transaction the service just started.

Usage Notes

This service is available only if your adapter supports built-in transaction management services, which you can confirm by checking the user guide for the adapter.

This service is intended for use with the pub.art.transaction:commitTransaction or pub.art.transaction:rollbackTransaction service. The *transactionName* value returned by a call to this service can be provided to pub.art.transaction:commitTransaction (to commit the transaction) or pub.art.transaction:rollbackTransaction (to roll back the transaction).

B Adapter Runtime Configuration Parameter Appendix

Overview

This appendix contains a description of the Adapter Runtime parameters you can specify in the server configuration file (server.cnf), which is located in the *Integration Server_directory*\config directory. Typically you use the Settings > Extended screen in Integration Server Administrator to update this file, but there might be times when you need to edit the file directly using a text editor. If you edit the file directly, you should first shut down Integration Server before updating the file. After you make the changes, restart the server. If you are using the Settings > Extended screen to update the server configuration file (server.cnf), a server restart is not required unless otherwise specified. The server uses default values for the parameters. If a parameter has a default, it is listed with the description of the parameter.

watt.art.analysis

Specifies whether or not to enable logging to analyze adapter listeners and their linked notifications.

watt.adk.adapterService.disable.errorlogging

Specifies whether or not the Adapter Development Kit (ADK) creates an entry in the error logs for exceptions in adapter services. When the parameter is set to `true`, ADK does not create an entry in the error logs for exceptions in adapter services. When the parameter is set to `false`, ADK creates an entry in the error logs for exceptions in adapter services. The default is `false`.

watt.art.adapterService.disable.errorlogging

Specifies whether or not the Adapter Runtime creates an entry in the error logs for exceptions in adapter services. When the parameter is set to `true`, Adapter Runtime does not create an entry in the error logs for exceptions in adapter services. When the parameter is set to `false`, Adapter Runtime creates an entry in the error logs for exceptions in adapter services. The default is `false`.

watt.art.clusteredPollingNotification.keepAliveExpireTimeout

This parameter is not supported with Integration Server 9.0 and higher. Specifies the amount of time, in milliseconds, that a node executing a clustered polling notification can be late before it is assumed to have failed. In general, this setting should be equal to the amount of drift anticipated on the server clocks. If not set, the secondary instance of the polling notification will change to the default maxLockDuration value of `180000` for the shared cache.

watt.art.clusteredPollingNotification.keepAliveInterval

This parameter is not supported with Integration Server 9.0 and higher. Specifies the frequency, in milliseconds, with which a secondary instance of a clustered polling notification will check to see if an executing instance is still alive. If you do not set the parameter, the secondary instance of the polling notification will change to the default maxLockDuration value of `180000` for the shared cache.

watt.art.concurrent.ConnectionPool

Specifies whether Adapter Runtime uses the concurrent connection pooling feature to concurrently create and release connections from a connection pool. When the parameter is set to `true`, Adapter Runtime can create and release multiple connections from a connection pool at the same time. When the parameter is set to `false`, Adapter Runtime can either create or release one connection at a time from a connection pool. The default is `false`.

watt.art.connection.nodeVersion

Specifies whether the adapter connection stores the password in the passman store and the password handle in the connection node. When the watt.art.connection.nodeVersion parameter is set to `1`, the password is embedded in the adapter connection. When the parameter is set to `2`, the password handle is stored in the adapter connection. The default is `2`. Software AG recommends using the default value. Every time you set a new value for this parameter, you must restart Integration Server or reload the WmART package.

When the value of the watt.art.connection.nodeVersion parameter is `2`, during run time-based deployment with webMethods Deployer you must perform variable substitution for the password field to deploy the password to the target system.

watt.art.deploy.listener.disable.waitTime

Specifies the time interval in milliseconds for which the Adapter Runtime waits for the listener to be disabled. The default time interval is 60000ms.

watt.art.notification.eventBus.retries

Specifies the number of retries to publish adapter polling and listener notifications to the event bus. The default number of retries is 5.

watt.art.notification.eventBus.retryInterval

Specifies the time interval in milliseconds between each retry. The default time interval is 30000ms.

watt.art.notifications.disableImplicitUpdate

Specifies whether the Adapter Runtime updates an adapter listener with the list of registered listener notifications when creating a new listener notification. When the parameter is set to `true`, the adapter disables the implicit update of registered notifications. When the parameter is set to `false`, the adapter does not disable the implicit update of registered notifications. The default is `false`.

watt.art.page.size

Specifies the maximum number of items to be displayed on an adapter's Connections screen, Listeners screen, and Notifications screen. The default is `10`. For more information about controlling pagination, see on page 22"Controlling Pagination".

watt.art.synchronousNotification.selectExecuteUser

Specifies WmArt-based adapters that are to include a **Run as User** column on the Listener Notifications screen. With this column in place, you can assign a user to a notification. Then, when the listener notification invokes a service, it runs as the specified user. You can specify one or more adapters. If you specify multiple adapters, separate the names with semicolons (;), for example:
`watt.art.synchronousNotification.selectExecuteUser=WmMQAdapter;WmSAP`

watt.art.service.pipeline.hidden

Specifies whether the adapter service pipeline is logged in the Integration Server log file. When the watt.art.service.pipeline.hidden parameter is set to `true`, the service pipeline is not logged in the Integration Server log file. When the parameter is set to `false`, the service pipeline is logged in the Integration Server log file. The default is `false`.

watt.art.tmgr.timeout

Specifies Integration Server's transaction timeout interval in number of seconds. If you do not set this parameter, Integration Server's default transaction timeout is no timeout (NO_TIMEOUT). The transaction timeout parameter does not halt the execution of a flow service. It is the maximum number of seconds that a transaction can remain open and still be considered valid. For example, if a current transaction has a timeout value of

60 seconds and a flow takes 120 seconds to complete, the transaction manager will roll back all registered operations regardless of the execution status.

watt.art.wmConnectionPool.pingRetryInterval

Specifies the time interval of an Adapter Runtime ping to the underlying resource. The default value is `120` seconds.

You use this parameter together with the watt.art.wmConnectionPool.pingSafeInterval parameter and the pub.art.connection:queryConnectionState service to monitor the state of the underlying resource at frequent intervals.

Note: The Adapter Runtime ping functionality uses database/back end credentials (user id and password) to create a connection and then destroy it to check the connectivity. Because creating and destroying a connection causes overhead, the ping interval should not be very small.

watt.art.wmConnectionPool.pingSafeInterval

Specifies a safe time interval for an Adapter Runtime ping to the underlying resource. The safe time interval is calculated based on the last connection provided by the connection pool. The default value is `5` seconds.

You use this parameter together with the watt.art.wmConnectionPool.pingRetryInterval parameter and the pub.art.connection:queryConnectionState service to monitor the state of the underlying resource at frequent intervals.

Note: The Adapter Runtime ping functionality uses database/back end credentials (user id and password) to create a connection and then destroy it to check the connectivity. Because creating and destroying a connection causes overhead, the ping interval should not be very small.

watt.pkg.art.pollingnotification.scheduler

This parameter is not supported with Integration Server 9.0 and higher. Specifies whether Integration Server executes adapter polling notifications using scheduled tasks or the shared cache. When this parameter is set to `false` (the default), Integration Server uses the shared cache to execute polling notifications. When this parameter is set to `true`, Integration Server uses scheduled tasks for the execution, scheduling, and cluster coordination of adapter polling notifications. When a notification is enabled, Integration Server creates a scheduled task that polls the back end resource at a specified interval. When a notification is disabled, Integration Server deletes the scheduled task.

When the parameter is set to `true`, you must also:

- Set the watt.pkg.art.pollingnotification.scheduler.adapters parameter to specify the adapters that will use the scheduled task functionality.

- Decide whether to display the scheduled tasks for adapter polling notifications in Integration Server Administrator, by setting the watt.pkg.art.scheduler.notificationtask.display parameter.

For complete information about configuring adapter polling notifications, see the adapter's documentation.

watt.pkg.art.pollingnotification.scheduler.adapters

This parameter is not supported with Integration Server 9.0 and higher. Specifies package names of adapters whose polling notifications are to execute using Integration Server scheduled tasks. To specify multiple package names, separate each entry with a semicolon (;). For example, to have polling notifications for webMethods Adapter for JDBC and webMethods Oracle Applications Adapter execute as scheduled tasks, specify the following:

```
watt.pkg.art.pollingnotification.scheduler.adapters=WmJDBCAdapter;WmOAAdapter
```

This parameter has no effect unless the watt.pkg.art.pollingnotification.scheduler is set to `true`. For complete information about configuring adapter polling notifications, see the adapter's documentation.

watt.pkg.art.scheduler.notificationtask.display

Specifies whether scheduled adapter polling notification tasks are shown on the Scheduler screen. When the parameter is set to `true`, the scheduled tasks are displayed. When this parameter is set to `false`, the tasks are hidden. The default is `true`. For complete information about configuring adapter polling notifications, see the adapter's documentation.

watt.art.connection.byPassConnValidation

Specifies whether or not the Integration Server validates the WmART based adapter connection parameters. When the parameter is set to `true`, the server does not validate adapter connection parameters, and all currently enabled adapter connections are maintained. Even if the backend resource is not available at restart, enabled connections remain enabled. When this parameter is set to `false`, the server verifies all enabled adapter connection parameters by trying to connect to its associated backend resource.

watt.server.jca.connectionPool.createConnection.interrupt.waitTime

Specifies the wait time interval in milliseconds, which elapses before Integration Server interrupts a connection creation thread that is in a wait state. The parameter does not require a default value.

watt.server.jca.connectionPool.threadInterrupter.sleepTime

Specifies the sleep time of the pool interrupter thread. The default value of the property is set to 2000msec, which is the sleep time for the pool interrupter thread.

watt.server.jca.connectionPool.threadInterrupt.waitTime

Specifies the wait time, measured in milliseconds, that elapses beforeIntegration Server Connection pool interrupts a connection creating thread or connection closing thread. The pool interrupter thread will start monitoring the server threads, only if this property is set. There is no default value. You must restart Integration Sever for changes to this parameter to take effect.