

webMethods Adapter Development Kit Installation and User's Guide

Version 9.12

October 2021

This document applies to webMethods Adapter Development Kit 9.12 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.

Copyright © 2008-2022 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <https://softwareag.com/licenses/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <https://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <https://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

Document ID: ADAPTER-WMK-IUG-912-20220323

Table of Contents

About this Guide	7
Document Conventions.....	8
Online Information and Support.....	9
Data Protection.....	10
1 Overview	11
What is the Adapter Development Kit?.....	12
Points of Integration.....	13
Development Time Tasks and Support.....	14
Design Time Tasks.....	19
The Runtime Conceptual Model.....	19
2 Installing and Uninstalling the Sample Adapter	21
Overview.....	22
Requirements.....	22
The Integration Server Home Directory.....	22
Installing Sample Adapter.....	22
Uninstalling Sample Adapter.....	23
3 The Adapter Definition	25
Overview.....	26
Creating an Adapter Package.....	27
Adapter Definition Classes.....	28
Adapter Definition Implementation Classes (Example MyAdapter).....	29
Creating a WmAdapter Implementation Class.....	29
Creating Resource Bundles.....	34
Deploying the adapter.....	40
Package Management.....	49
4 Connections	55
Overview.....	56
Adapter Connection Classes.....	58
Adapter Connection Implementation Classes.....	60
Creating a WmManagedConnection Implementation Class.....	61
Creating a WmManagedConnectionFactory Implementation Class.....	61
Updating the Resource Bundle.....	66
Registering Connection Factories in the Adapter.....	67
Connection Class Interactions.....	67
Configuring and Testing Connection Nodes.....	71
5 Adapter Services	75
Overview.....	76

Adapter Service Classes.....	76
Metadata Model for Adapter Services.....	78
Adapter Service Template Interactions.....	98
Adapter Service Implementation.....	102
Configuring and Testing Adapter Service Nodes.....	119
6 Polling Notifications.....	121
Overview.....	122
Polling Notification Classes.....	123
Polling Notification Callbacks.....	125
Metadata Model for Polling Notifications.....	125
Polling Notification Interactions.....	126
Polling Notification Implementation.....	129
Configuring and Testing Polling Notification Nodes.....	138
Cluster Support for Polling Notifications.....	143
7 Listener Notifications.....	149
Overview.....	150
Listener Classes.....	151
Asynchronous Listener Notification Classes.....	153
Synchronous Listener Notification Classes.....	154
Listener and Listener Notification Interactions.....	156
Listener Implementation.....	159
Listener Notification Implementation.....	167
Configuring and Testing Listener Nodes and Listener Notification Nodes.....	178
8 Runtime Activities.....	183
Overview.....	184
Retry and Recovery Architecture.....	184
Runtime Connection Allocation for Adapter Services.....	187
9 Usage Scenarios.....	195
How to register an adapter with the Integration Server?.....	196
How to create an adapter connection implementation?.....	209
How to create an adapter service implementation?.....	216
How to create a polling notification implementation?.....	253
How to create an adapter listener implementation?.....	278
A Alternative Approaches to Metadata.....	307
Overview.....	308
Implementing Metadata Parameters Using External Classes.....	308
An Alternative Approach to Organizing Resource Domains.....	308
Using Resource Bundles with Resource Domain Values.....	325
B Integration Server Transaction Support.....	329
Overview.....	330
Simple Transactions.....	331

More Complex Transactions.....	331
Implicit Transaction Usage Cases.....	332
Explicit Transaction Usage Cases.....	333
Built-In Services For Explicit Transactions.....	337
Transaction Error Situations.....	340
Specifying Transaction Support in Connections.....	341
C Using the Services for Managing Namespace Nodes.....	343
Overview.....	344
Connection Services.....	392
Adapter Service Services.....	402
Listener Services.....	410
Listener Notification Services.....	418
Polling Notification Services.....	430
D Using the Sample Adapter.....	441
Overview.....	442
The Sample Server.....	442
Banking Services, Queries and Alerts.....	444
Prerequisites for Code Compilation.....	445
Phase 1: Creating an Adapter Definition.....	446
Phase 2: Adding a Connection.....	449
Phase 3: Adding Adapter Services.....	454
Phase 4: Adding Polling Notifications.....	461
Phase 5: Adding Listener Notifications.....	470

About this Guide

■ Document Conventions	8
■ Online Information and Support	9
■ Data Protection	10

This guide describes how to install, upgrade, and uninstall Adapter Development Kit, as well as how to configure and use it. This guide contains information for application developers who want to create adapters that interact with webMethods Integration Server.

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

- Terminology and basic operations of your operating system
- How to perform basic tasks with Integration Server and Software AG Designer

This version of the *webMethods Adapter Development Kit Installation and User's Guide* contains the most up to date information about the Adapter Development Kit. This version obsoletes, replaces, and supersedes all previous versions.

Document Conventions

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

Online Information and Support

Product Documentation

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

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

Product Training

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

Tech Community

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

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

Product Support

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

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

Data Protection

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

1 Overview

■ What is the Adapter Development Kit?	12
■ Points of Integration	13
■ Development Time Tasks and Support	14
■ Design Time Tasks	19
■ The Runtime Conceptual Model	19

What is the Adapter Development Kit?

The Adapter Development Kit (ADK) is a set of public Application Programming Interfaces (APIs) that you can extend to create custom adapters that interact with webMethods Integration Server. The ADK abstracts adapters from Integration Server, thus ensuring that ADK created adapters continue to run with future versions of Integration Server.

Like any Integration Server based adapter, your adapter links your backend system with heterogeneous systems outside your organization through Integration Server, regardless of the technology at either end, and without requiring changes to the existing security infrastructure. An adapter that you create is secure and scalable because it is an add-on, system level software component that you plug into an Integration Server. The adapters use the Integration Server application services to connect to backend systems.

The ADK provides:

- An architecture for creating adapters based on Java EE Connector Architecture (JCA).

This architecture supports the JCA Common Client Interface (CCI) and extends it to provide additional functionality. It includes a standard set of system level contracts between an Integration Server and the backend system to which the adapter connects. These contracts handle aspects of integration such as connections, adapter services, notifications, and system message logging.
- An example adapter package named `WmSampleAdapter` (**Sample Adapter**).

You can use **Sample Adapter** as a model for developing your own adapters. The **Sample Adapter** enables you to exchange data with a simulated adapter resource provided with the example adapter. Configure this adapter to perform a banking application. All of the underlying `SampleAdapter` class files are located in the `Integration Server_directory \ instances\<instance_name>\packages\WmSampleAdapter` directory. For more information about how the Sample Adapter was developed, and how you can configure and use the adapter, see [“Using the Sample Adapter” on page 441](#).
- A set of auxiliary Java services that you can use to:
 1. Replicate namespace nodes programmatically.
 2. Change the nodes' metadata when deploying the adapter to a different Integration Server.
- Online API Reference Javadoc files that provide detailed descriptions and usage information about all public APIs provided in the WmART package.

The WmART package contains the components of the adapter run time as well as the ADK classes you extend to create the adapter implementation.

To create an adapter, you need access to webMethods Integration Server (IS), Software AG Designer, a Java 1.8 compiler, and any Java editor.

Points of Integration

Before you build an adapter, determine which "point of integration" is most appropriate for integrating your backend system. When determining this, consider factors such as the type and volume of information you need to move between systems, and the number of systems you need to integrate. The major "point of integration" types include the following:

Data-Level Integration

An Enterprise Information System (EIS) that is integrated at the data level moves data between data stores. That is, the data stores in your EIS are involved in processing or storing transaction data received from outside the EIS.

For example:

- You might need to expose a catalog and pricing database to customers to enable them to transact with it.
- In addition to the catalog and pricing database, you need to reference or update data located in another database to process a customer transaction. After you extract data from an EIS-to-Integration Server purchase order (such as item, quantity, and price), you:
 - Process that data using another database (perhaps you subtract the quantity ordered from the quantity in inventory, which is located in another database)
 - Store the result (the updated quantity in inventory) in that other database.

Assuming that the catalog and pricing database has a JDBC driver, you can use the database as your point of integration. That is, it would be the mechanism you use to build your adapter. You might have several of your databases involved in an integration scheme.

Application Interface-Level Integration

An EIS that is integrated at the application interface level leverages the exposed interfaces of custom or packaged applications, such as SAP, Siebel, and PeopleSoft applications. Such an interface, which usually consists of a set of the application's APIs, enables applications outside of your EIS to access business processes as well as simple information.

Method-Level Integration

An EIS that is integrated at the method level is similar to one that is integrated at the application interface level. Using this point of integration, you can enable any application to access the methods of any application in your EIS.

User Interface-Level Integration

An EIS that is integrated at the user interface level uses a more primitive, yet viable approach. This approach, also known as "screen scraping", leverages legacy EIS user interfaces as a common

point of integration. For example, mainframe applications that do not provide database or business process level access may be accessed through the user interface of the application.

Development Time Tasks and Support

At development time, you write the Java classes that define the adapter. To do this, you extend the base classes provided by the ADK to produce an adapter definition class and template classes for connections, adapter services, polling notifications, and listener notifications.

At development time, you must:

1. Create the Adapter Definition.
2. Define Connections and Connection Factories.
3. Define Adapter Service Templates.
4. Define Polling Notification Templates.
5. Define Listener Notification Templates.
6. Define Metadata.

Support for other development-time activities includes:

1. Transaction Support.
2. Exception Support.
3. Logging Support.
4. Internationalization Support

Create the Adapter Definition

An adapter definition is the framework of an adapter. An adapter definition is recognized as an adapter by your Integration Server, but it lacks functionality. In later stages of development, you add functionality by defining templates for adapter connections, adapter services, and optionally for polling notifications and listener notifications.

In the adapter definition, you create services and methods that:

- Describe the adapter to Integration Server.
- Describe the adapter's resources to Integration Server, including its connection factories, adapter service templates, polling notification templates, listener notification templates, and its default resource bundle implementation class.

A *resource bundle* contains all display strings and messages used by the adapter at run time and at design time. A resource bundle is specific to particular locale. If you plan to run your adapter in multiple locales, include a resource bundle for each locale. Doing so enables you to internationalize an adapter quickly, without having to change any code in the adapter. Each adapter must provide at least a default resource bundle.

- Initialize properties and resources associated with the adapter, and clean up the resources when the adapter is disabled.
- Load the adapter onto Integration Server when the adapter is enabled, and unload the adapter when the adapter is disabled.

For more information about adapter definition, see [“Overview” on page 26](#).

Define Connections and Connection Factories

A connection class functions as a connection to the resource with which the adapter must communicate (known as the *adapter resource*). The ADK provides a connection management service that dynamically manages connections and connection pools, based on rules that the users of the adapter establish when they configure connections. To create connections, the connection management service uses a connection factory class.

The users of the adapter create one or more connection namespace nodes, using the adapter's administrative interface. The creation of namespace nodes is a design time activity. The users of the adapter also create namespace nodes for the adapter service templates and notification templates.

For more information about connections, see [“Overview” on page 56](#).

Define Adapter Service Templates

An adapter service defines an operation that the adapter performs on a resource. You need to implement one adapter service template class for each resource operation the adapter supports.

For example, you might create a service template that fetches rows from a database based on a key field, and another service template that updates and inserts records into the database.

The users of the adapter define their run time adapter service nodes based on these templates. Designer provides facilities for creating, configuring, and testing adapter service nodes.

For more information, see [“Overview” on page 76](#).

Define Polling Notification Templates

A polling notification is a mechanism that notifies your application when an event occurs in your adapter's resource. For example, your application might need to be notified when data is added, updated, and deleted from the resource. You need to implement one polling notification template class for each polling notification the adapter supports.

A polling notification periodically checks the resource at specified intervals for the occurrence of events, and publishes a document each time an event occurs in the resource.

The users of the adapter define their run time polling notification nodes based on these templates, in a way similar to how they configure adapter service nodes. Designer provides facilities for creating, configuring, and testing polling notification nodes.

For more information, see [“Overview” on page 122](#).

Define Listener Notification Templates

A listener notification works in conjunction with a listener object to create a much more powerful model for detecting and processing events in the adapter resource than is possible with polling notifications. You need to implement one listener notification template class for each listener notification the adapter supports.

A listener object is connected to an adapter resource, waiting for the server to deliver event notifications. The listener object is instantiated and is given a connection when the associated node is enabled. The listener object remains active with the same connection to monitor the resource activity until it is explicitly disabled.

When the listener detects a publishable event in the resource, it passes an object back to the server. The server interrogates a configured list of listener notifications associated with the listener node until it finds a listener notification node that can process the event. The first listener notification to return true from this call is invoked.

The ADK includes both a synchronous and an asynchronous processing model.

- **Asynchronous Listener Notification.** Publishes a document to a webMethods messaging queue, using the `doNotify` method. The users of the adapter may 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 service.
- **Synchronous Listener Notification.** Invokes a specified Integration Server service, and potentially receives a reply from the service and delivers the results back to the adapter resource. In this case, the notification object's `runNotification` method calls `invokeService` (instead of `doNotify`), to process the data produced by the notification. A synchronous listener notification can publish a document and wait for reply.

For more information, see [“Overview” on page 150](#).

Define Metadata

Connection factories, adapter service templates, polling notification templates, and listener notification templates all support a metadata interface. Metadata that you create in your adapter implementation primarily supports design time activities. It describes parameters that the users of the adapter use to create namespace nodes for connections, adapter services, and notifications, and describes the data passed to and from adapter services and notification nodes.

When a user of the adapter creates a namespace node, the server interrogates the adapter for metadata to describe the parameter values supported by the node. The node stores the parameter values selected or entered by the user. These parameters are used to configure the appropriate implementation class when it is instantiated at run time.

Metadata parameter values are derived from the adapter's resource domain. A *resource domain* defines the domain of valid values for metadata parameters, based on rules and/or data that are specific to the adapter resource. A resource domain can have a name, a set of resource domain values, properties that affect the behavior of the resource domain, and associations with metadata parameters.

A resource domain can be either fixed or dynamic. A *fixed* resource domain displays default values that you provide for the resource domain parameters. With a *dynamic* resource domain, you use a method that enables the adapter to look up values for the parameters.

You can use resource domain values to constrain the values of parameters, to enable dynamic validation of user supplied data, and to disable parameters, based on specific sets of values in other parameters. A common use of resource domain values is to create a dropdown list of values for a parameter.

With both fixed and dynamic resource domains, you can allow the users of the adapter to enter their own values. In addition, you can enable the adapter to validate these values using callbacks known as *adapter check values*.

With adapter services and polling notifications, resource domain values can interact with the Adapter Service Editor and the Adapter Notification Editor. As values in one parameter change, callbacks are made to the adapter to update resource domain values. In addition, the adapter can retrieve resource domain values directly from the adapter resource.

A parameter may contain a single value or an array of values. Parameters that hold arrays of values are called *sequence parameters*. The user interface for configuring connections does not allow you to populate more than the first element of a sequence parameter.

The metadata model provides the ability to:

- Define groups of parameters that appear on different pages in the user interface.
- Define *resource domain lookups* that return dropdown lists of possible values for a parameter. Resource domain lookups may return values established at development time and/or retrieved at design time from the resource with which the adapter communicates.
- Establish sophisticated relationships between parameters using field maps, tuples, and resource domain dependencies.
 - *Field maps* are used to place sequence parameters in a grid-style table with each sequence parameter forming a column in the table.
 - *Tuples* are used in conjunction with field maps. When a set of parameters is placed in a tuple, resource domain lookups for those parameters are always made together. This is commonly used when columns are closely related, for example when column 1 contains field names, and column 2 contains data types for those fields.
 - *Resource domain dependencies* indicate that the values to be returned in a resource domain lookup are dependent on the value of one or more other parameters. Whenever the value of one parameter changes, if a second parameter's lookup is dependent on the value of the first, the lookup for the second parameter is performed again, with the new value of the first parameter being passed in as an argument to the lookup.
- Define signatures that define the data that should be passed to, or received from, the node when the service or notification is executed (by calling the execute method of the implementation class).
- Make interactive calls into the adapter to validate the data entered by the user.

- Define tree structures to facilitate input of parameter values or to create logical groupings of the fields in a signature.

Metadata is discussed in more detail throughout this document, in the chapters about connections, adapter services, and notifications.

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 transactions types:

- 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 transaction types, without requiring the users of the adapter 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 the adapter services use connections to create transactions. However, there are cases where the user of the adapter 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, see [“Overview” on page 330](#).

Exception Support

The ADK provides standard exception classes that enable the adapter implementation to inform the server of exception conditions encountered by the adapter implementation. Any exception from an adapter implementation is caught and logged in the Integration Server and error logs. For details about the standard ADK exceptions, see the Javadoc for the `com.wm.adk.error` package.

Logging Support

The ADK provides the adapter with write access to the Integration Server and error logs. It also provides the ability to determine whether a particular message at a particular log level is written to the log. All log messages generated by an adapter are stamped with the current date and time, as well as codes that uniquely identify the adapter generating the message. Logging services are tightly integrated with the resource bundle facilities for internationalization support. For more information, see the Javadoc for `com.wm.adk.log.ARTLogger` package.

Internationalization Support

If you plan to run your adapter in multiple locales, you can internationalize an adapter quickly, without having to change any code in the adapter. To accomplish this, you use resource bundles. A resource bundle contains all display strings and messages used by the adapter at run time and at design time. A resource bundle is specific to a particular locale.

For more information, see [“Creating Resource Bundles” on page 34](#).

Design Time Tasks

At design time, the users of the adapter select the template classes to configure namespace nodes. A *namespace node* is a run time component containing information about how a connection, adapter service, polling notifications, or listener notifications should behave at run time. The users of the adapter configure and initialize the run time components of the adapter. To do this, use the following Integration Server packages:

- The *adapter package*, which contains the adapter run time components as well as the ADK classes that you, the adapter developer, extend to produce the adapter implementation.
- A *namespace node package*, in which the users of the adapter create the adapter's namespace nodes for connections, adapter services, polling notifications, and listener notifications.

A *namespace node* (or *node*) is based on its corresponding Java classes. For example, a connection node is based on the Java connection classes created at development time.

Design time tasks include the following:

- Create one or more namespace node packages.
- Create and initialize namespace nodes for connections, adapter services, polling notifications, listeners, and listener notifications. The users of the adapter can place all the nodes of an adapter in one package, or distribute them among multiple packages.
- Load the adapter package and namespace node packages into Integration Server.

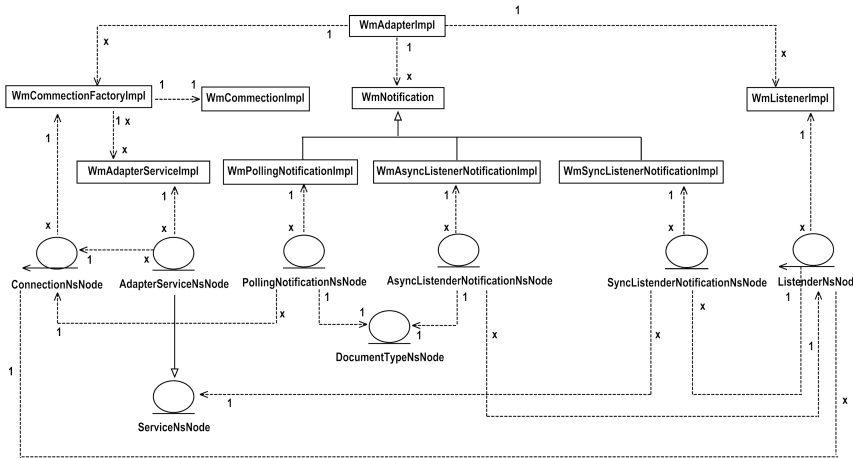
To perform these tasks, the users of the adapter require access to Integration Server, Designer, and a web browser.

The Runtime Conceptual Model

At run time, the adapter services and notifications communicate with the adapter resource to perform the function for which the adapter was created. Designer provides facilities for testing adapter services and notifications.

Although the following diagram is not strictly UML-compliant, it shows the relationships between the implementation classes of an adapter and their corresponding namespace nodes created at design time.

Relationships between adapter implementation classes and namespace nodes



In the diagram, entity icons represent namespace nodes, while standard class icons represent the implementation classes. The (unlabeled) dependency lines show direct references between classes and/or nodes. Thus, the adapter implementation class directly references supported connections or notification templates by directly referencing the connection factory and the notification implementation class. The connection factory for each connection type references the service implementation class of each supported adapter service template.

Each namespace node depends on an implementation class. The implementation class for each node provides metadata that describes the data that is included in the node at design time. The node provides parameter settings that are passed back to the implementation class when that class is executed at run time. Note that service and notification nodes require a reference to a connection node that provides access to the resource.

For more information, see [“Overview” on page 184](#).

2 Installing and Uninstalling the Sample Adapter

■ Overview	22
■ Requirements	22
■ The Integration Server Home Directory	22
■ Installing Sample Adapter	22
■ Uninstalling Sample Adapter	23

Overview

The Adapter Development Kit (ADK) is a framework providing a set of public Application Programming Interfaces (APIs) that you can extend to create custom adapters that interact with Integration Server. Adapter Development Kit(ADK) framework is installed along with webMethods Adapter Runtime(ART) during Integration Server installation. This chapter talks about installing and uninstalling the component **Sample Adapter** using Software AG Installer and Software AG Uninstaller. **Sample Adapter** is a sample to consume ADK. For complete information about other installation methods or installing other webMethods products, see the *Installing Software AG Products* guide for your release.

Requirements

For a list of the operating systems, webMethods Integration Server releases and Software AG Designer releases supported by the Adapter Development Kit, see the *webMethods Adapters System Requirements* .

Sample Adapter has no hardware requirements beyond those of its host Integration Server.

The Integration Server Home Directory

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

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

This guide uses the *packages_directory* as the home directory in Integration Server classpaths. The *packages_directory* is *Integration Server_directory \ instances\<instance_name>\packages* directory.

Installing Sample Adapter

1. Download Software AG Installer from the [Empower Product Support website](#).
2. Run the Software AG Installer, as described in the document *Using Software AG Installer*.
3. Specify the installation directory as follows:
 - If you are installing on an existing webMethods Integration Server, specify the webMethods installation directory that contains the host webMethods Integration Server.
 - If you are installing both, the host webMethods Integration Server and the **Sample Adapter**, specify the installation directory to use.

4. In the product selection list, select **Adapters > webMethods Adapter Development Kit > Sample Adapter 9.12.**

If you are using webMethods Integration Server 9.12 and above, Software AG Installer installs the adapter in both locations, *Integration Server_directory* \packages and your instance's packages directory located in *Integration Server_directory* \instances*<instance_name>* \packages.

Software AG Installer installs the components of the **Sample Adapter** (WmSampleAdapter package and Sample Server for use with the **Sample Adapter**) in *Integration Server_directory* \instances*<instance_name>* \packages\WmSampleAdapter directory.

5. After installation is complete, start the host webMethods Integration Server.

Uninstalling Sample Adapter

1. Run the Software AG Uninstaller, as described in the document *Using Software AG Installer*.
2. Select the webMethods installation directory that contains the host webMethods Integration Server.
3. In the product selection list, select **Adapters > webMethods Adapter Development Kit > Sample Adapter 9.12.**

Software AG Uninstaller removes all **Sample Adapter** related files that were installed into the webMethods Integration Server installation directory. However, Software AG Uninstaller will not delete the following:

- *Integration Server_directory* \instances*<instance_name>* \packages\WmSampleAdapter directory if you added any files to it. You can delete this directory manually.
- Any user-defined Adapter Development Kit components such as connections, adapter services, or adapter notifications. Because these components will not work without the adapter, delete them manually using Software AG Designer, or Integration Server Administrator.

4. Restart the host webMethods Integration Server.

3 The Adapter Definition

- Overview 26
- Creating an Adapter Package 27
- Adapter Definition Classes 28
- Adapter Definition Implementation Classes (Example MyAdapter) 29
- Creating a WmAdapter Implementation Class 29
- Creating Resource Bundles 34
- Deploying the adapter 40
- Package Management 49

Overview

An adapter definition is the framework of an adapter. The adapter definition is recognized as an adapter by Integration Server, but lacks functionality. This chapter describes how to create an adapter definition.

To create an adapter definition, perform the following tasks:

- Create a webMethods package for the adapter definition.
- Create an adapter definition implementation class by extending the `com.wm.adk.WmAdapter` base class. The adapter definition implementation class represents the main class of the adapter. In this class, create services and methods that:
 - Describe the adapter to Integration Server.
 - Describe the adapter's resources to Integration Server, including its connection factories, notification templates, and its default resource bundle implementation class.
 - Initialize resources and properties referenced by the adapter definition when the adapter is enabled, and clean up the resources when the adapter is disabled (optional).
 - Load the adapter onto Integration Server when the adapter is enabled, and unload the adapter when the adapter is disabled.

For more information, see [“Creating a WmAdapter Implementation Class” on page 29](#).

- Create one or more resource bundles.

The resource bundle class must extend the `java.util.ListResourceBundle` base class, and contain all the display strings and messages used by the adapter at run time and at design time. A resource bundle is specific to a particular locale. If you plan to run your adapter in multiple locales, you can include a resource bundle for each locale. Creating a resource bundle for each locale enables you to internationalize an adapter quickly, without having to change any code in the adapter. Each adapter must have one or more resource bundles. For more information, see [“Creating Resource Bundles” on page 34](#).

- Deploy the adapter.

- Create the startup and shutdown Java Services.
- Compile the adapter definition.

Compile your implementation class and construct the Java service nodes for your startup and shutdown services. The ADK provides a sample ANT script that you can run from the packages folder.

- Configure the startup and shutdown Java Service in Designer.

For more information, see [“Deploying the adapter” on page 40](#).

Finally, the users of the adapter use Integration Server Administrator to load the adapter by enabling the adapter package. For more information, see [“Package Management” on page 49](#).

Creating an Adapter Package

Create an adapter package in the same way you create any webMethods package, using Designer. If you need instructions for creating a package, see the *webMethods Service Development Help* for your release.

Designer creates a folder structure in which you can develop your adapter, as shown.

Note:

- You must use the `adapterPackageName\code\source` folder as your source base and create directories corresponding to your Java package structure (for example, `com\mycompany\adapter\myadapter`).
- You must set the package dependency as follows:

Field	Value
Package	WmART
Version	*,*

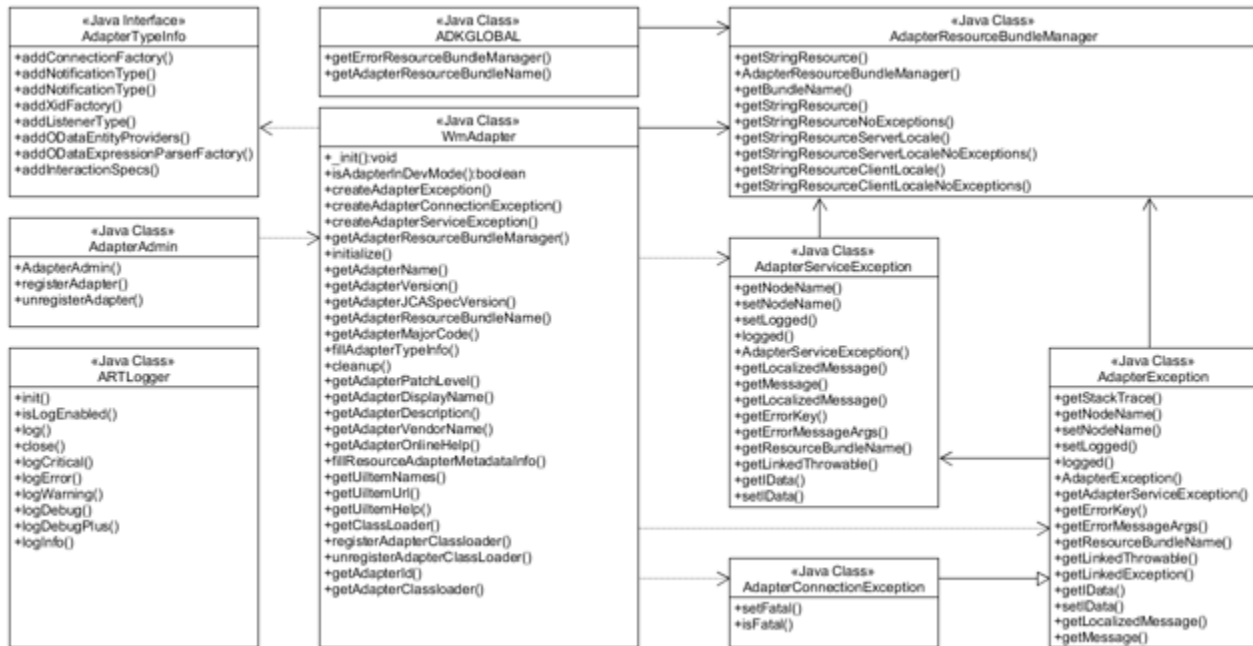
- The WmART package is installed when you install Integration Server.

The following figure shows the webMethods package structure:

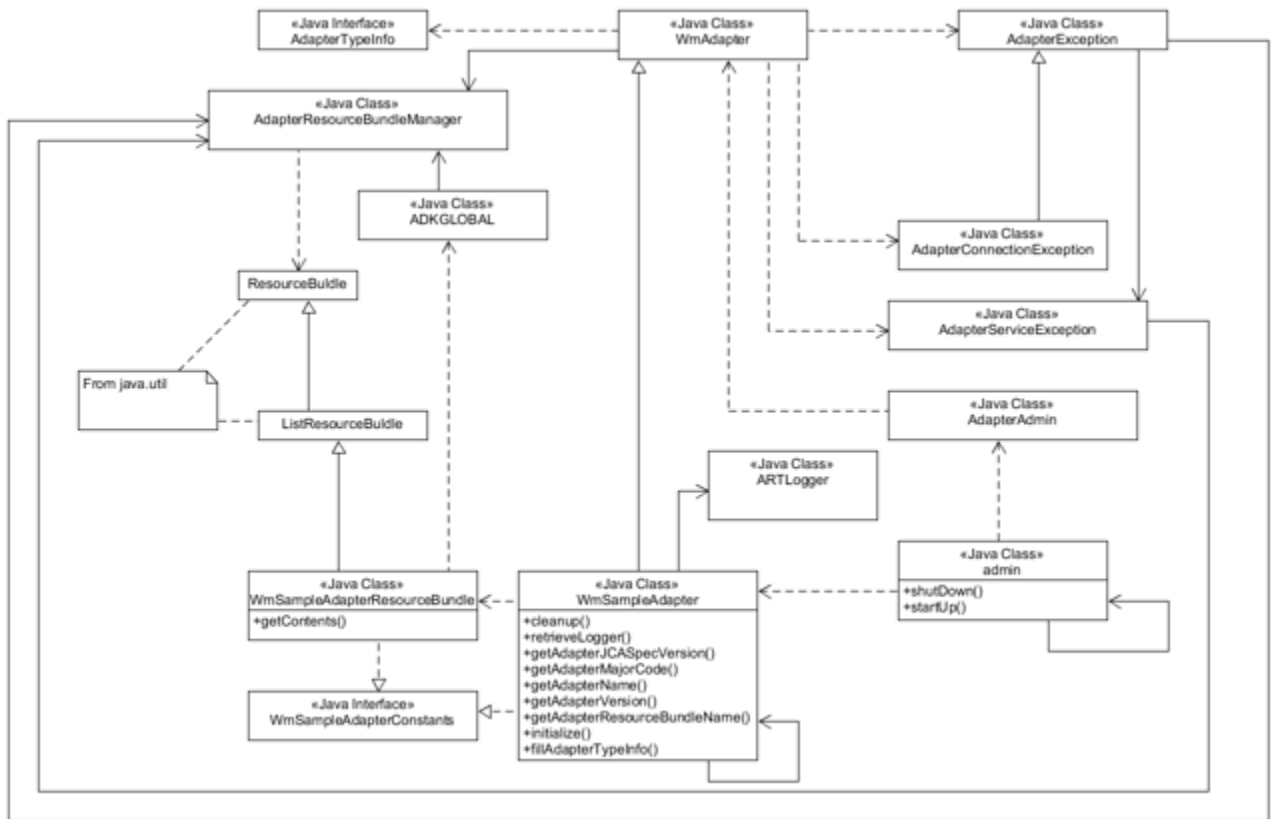
The screenshot shows a package structure for 'MyAdapter'. The tree view on the left lists folders: code, classes, jars, source, config, doc, lib, ns, com, pub, resources, templates, and web. The table view on the right provides details for each item:

Name	Status	Date modified	Type	Size
code	⊙	26-07-2021 15:49	File folder	
config	⊙	26-07-2021 15:49	File folder	
doc	⊙	26-07-2021 15:49	File folder	
lib	⊙	19-07-2021 17:51	File folder	
ns	⊙	26-07-2021 15:49	File folder	
pub	⊙	26-07-2021 15:49	File folder	
resources	⊙	19-07-2021 17:51	File folder	
templates	⊙	19-07-2021 17:51	File folder	
web	⊙	19-07-2021 17:51	File folder	
manifest.bak	⊙	19-07-2021 17:55	BAK File	1 KB
manifest.v3	⊙	20-07-2021 17:52	V3 File	1 KB

Adapter Definition Classes



Adapter Definition Implementation Classes (Example MyAdapter)



Creating a WmAdapter Implementation Class

Create an adapter definition by extending the `com.wm.adk.WmAdapter` base class. This class represents the main class of the adapter. In this class, you create services and methods that:

- Describe the adapter to Integration Server.
- Describe the adapter's resources to Integration Server.
- Initialize and cleanup resources.
- Add custom Dynamic Server Pages (DSPs) to your adapter's administrative interface.

This section describes the basic steps for implementing an adapter definition.

Describing the Adapter to Integration Server

Create an adapter definition by extending the `com.wm.adk.WmAdapter` base class. You must override the following base class methods in your `WmAdapter` implementation class:

Name	Description
getAdapterName	Returns the internal name of the adapter. This name is used to identify the text fields in the resource bundle, and to identify the relationship between the adapter and its associated namespace nodes. For this reason, it is important that the value returned by this method does not change after namespace nodes have been created. This name must be unique within the scope of Integration Server.
getAdapterVersion	Returns the current version of the adapter. This value appears in the adapter's About page. This must not be confused with the package version used when setting package dependencies.
getAdapterJCASpecVersion	Returns the JCA standard version supported by the adapter. This value must always be 1.0.
getAdapterMajorCode	<p>Must return a unique numeric value (if required) that you can obtain from Software AG.</p> <p>Every adapter built using the ADK requires an internal ID called a major code. A <i>major code</i> is an integer ID that Integration Server uses to distinguish journal log information between different adapter types. The major code is a four digit number between 1 and 9999.</p> <p>Each adapter implementation must have a major code that is unique from all other adapters built using the ADK that might be present in the same webMethods environment. Adapters with identical major codes generate an error in the Integration Server log. More importantly, Integration Server log entries from same code adapters is indistinguishable.</p> <p>The major code ranges are reserved as follows:</p>
Major Code Range	Description
1-6999	This range is reserved for Software AG built webMethods commercial adapters.
7000-8999	This range is reserved for adapters built by Software AG Development Partners. If you are a Development Partner, you must register your major code with Software AG.
9000-9999	This range is reserved for adapters you build for use within your own organization. You do not have to register the major codes for these adapters unless you are running them in an environment where the

Name	Description
	adapters' major codes conflict with other adapters in your webMethods environment.
getAdapterResourceBundleName	Returns the name of your ListResourceBundle implementation class. The value must be the fully qualified name of the resource's associated implementation class rather than an object instance.
fillAdapterTypeInfo	Sets the names of all connection factory and notification implementation classes in an AdapterTypeInfo object.
supportsParallelAssetInitialization	Indicates if the adapter supports parallel asset initialization. The default return value is <code>false</code> . Override this method and use the <code>watt</code> property for the adapter to enable or disable parallel asset initialization.

Describing the Adapter's Resources to Integration Server

The adapter resources that you must describe to Integration Server include:

- All connection factories and notification templates that the adapter supports. You set the names of all connection factory and notification implementation classes in an AdapterTypeInfo object in your adapter's implementation of the fillAdapterTypeInfo method. Set these names later, when you implement the connections and notifications.
- The default resource bundle implementation class. Create a default resource bundle by extending the `java.util.ListResourceBundle` base class. Deliver the name of your ListResourceBundle implementation class in your adapter's implementation of the getAdapterResourceBundleName method.

Initializing and Cleaning Up

Initialize and clean up the resources used in your adapter implementation. In your WmAdapter implementation class, you can initialize resources and properties specific to your adapter when the adapter is enabled and release the resources when the adapter is disabled. Resources held by connections, adapter services, and notifications have their own cleanup mechanisms.

For example, you might initialize the ARTLogger resource.

- There can be only one ArtLogger instance per adapter (that is, per major code). You must manage the ArtLogger instance in your WmAdapter implementation class, but it is optional. You must use a static accessor method, which facilitates access to the ArtLogger instance. In the example, this is accomplished using the getLogger method.
- You must release the ArtLogger instance (and the adapter's major code) using the ArtLogger.close method when the adapter is disabled. In the example, this is accomplished using `MyAdapter.cleanup`, which is called from `AdapterAdmin.unregisterAdapter`.

- You can also use an ARTLogger object to generate journal log entries for your adapter.

Adding Custom DSPs to the Adapter Interface

By default, the adapter's administrative interface includes DSPs for connections, polling notifications, listeners, and listener notifications. You may add custom DSPs by overriding the following public methods provided in the `com.wm.adk.WmAdapter` abstract class:

Method	Description
<code>getUiltemNames</code>	Returns a list of the label strings to insert in the left hand panel of the Integration Server Administrator page for the adapter. Each label represents a link to the DSP to be executed.
<code>getUiltemUrl</code> (<i>String itemName</i>)	Returns the URL of the DSP associated with the given <i>itemName</i> . This effectively binds the displayable item link/label name with the DSP to launch when the users of the adapter select that link. The URL is relative to the packages folder of the server installation. For example, if your adapter name is <i>WmFoo</i> and your DSP is <i>bar.dsp</i> , the URL would be <code>\WmFoo\bar.dsp</code> . In the file system, however, the <code>.dsp</code> file actually resides in <code>packages\WmFoo\pub\bar.dsp</code> ; the <code>pub</code> folder is not included in the URL. You may also append arguments to the URL, using the standard notation <code>?=</code> .
<code>getUiltemHelp</code> (<i>String itemName</i>)	Returns the URL of the help file describing the custom DSP page associated with the given <i>itemName</i> . The pathname requirements are the same as for <code>getUiltemUrl()</code> . For example, if your help file is located in <code>\WmFoo\pub\bar_dsp.html</code> , the path returned by this method must be <code>\WmFoo\bar_dsp.html</code> .

The labels for these DSPs appear in the navigation area in the left hand window of the interface, immediately below the default labels (the Connections, Polling Notifications, Listeners, and Listener Notifications labels) but above the About label.

For information about creating DSPs, see *Dynamic Server Pages and Output Templates Developer's Guide*.

Internationalization Considerations for Custom DSPS

You are responsible for implementing these methods (and the associated DSPs) in a manner that takes into consideration the client's locale. In particular, the implementation of `getUiltemNames` method must perform the necessary resource domain lookups in order to return locale specific values. The system does not automatically perform these lookups for you.

Creating WmAdapter Implementation Class with example

1. Create a folder structure for the Java package for adapter implementation. For example: `com\mycompany\adapter\myadapter`. In the example, the Java package created is `com\wm\MyAdapter`.

Note:

You must create your Java package and classes in the `adapterPackageName\code\source` folder in the webMethods package you created using Designer.

2. Create an interface that contains the constants for the adapter implementation.

In the example, create *MyAdapterConstants* interface:

```
package com.wm.MyAdapter;
public interface MyAdapterConstants {

    static final int ADAPTER_MAJOR_CODE = 9001;
    static final String ADAPTER_JCA_VERSION = "1.0";
    static final String ADAPTER_NAME = "MyAdapter";
    static final String ADAPTER_VERSION = "9.12";

    //Using next statement creates cyclic class loading dependency issue
    //therefore, the resource bundle class name is fully spelled out
    //static final String ADAPTER_SOURCE_BUNDLE_NAME =
    MyAdapterResource.class.getName();
    static final String ADAPTER_SOURCE_BUNDLE_NAME =
        "com.wm.MyAdapter.MyAdapterResource";
}
```

3. Create a class by extending the `com.wm.adk.WmAdapter` base class.
 - Your `WmAdapter` implementation class (*MyAdapter*) must call the base class constructor (`super()`). The base class constructor calls several of the implementation class's methods and instantiates your resource bundle.
 - The base class constructor calls several of the implementation class's methods after the first call to `getInstance`. It is vital that they do not invoke another call to `getInstance` which results in an endlessly recursive call to the constructor, and ultimately crashes the JVM and bring down Integration Server. This must be avoided in static initializers in your resource bundle as some of the resource bundles are keyed on the same string that is returned from *MyAdapter*.`getAdapterName`. Do not populate the string in a static initializer. This line of code in a static initializer of a resource bundle produces an undesirable results:

```
{MyAdapter.getInstance().getAdapterName() +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME, "My Adapter"}
```

- When specifying any resource, you must use the fully qualified name of the resource's associated implementation class (rather than an object instance).

In the example, create *MyAdapter* class:

```
package com.wm.MyAdapter;
import java.util.Locale;
import com.wm.adk.WmAdapter;
import com.wm.adk.error.AdapterException;
import com.wm.adk.info.AdapterTypeInfo;
import com.wm.adk.log.ARTLogger;

public class MyAdapter extends WmAdapter implements MyAdapterConstants{
    public static MyAdapter _instance = null;
```

```
public static ARTLogger _logger = null;

public MyAdapter() throws AdapterException { super(); }
public void fillAdapterTypeInfo(AdapterTypeInfo arg0, Locale arg1) {}
public String getAdapterJCASpecVersion() { return ADAPTER_JCA_VERSION; }
public int getAdapterMajorCode() { return ADAPTER_MAJOR_CODE; }
public String getAdapterName() { return ADAPTER_NAME; }
public String getAdapterResourceBundleName() { return ADAPTER_SOURCE_BUNDLE_NAME; }
}
public String getAdapterVersion() { return ADAPTER_VERSION; }
public static ARTLogger getLogger() { return _logger; }
```

```
public void initialize() throws AdapterException {
    // TODO Auto-generated method stub
    _logger = new ARTLogger(getAdapterMajorCode(),
        getAdapterName(),
        getAdapterResourceBundleName());
    _logger.logDebug(9999,"My Adapter Initialized");
}
public void cleanup() {
    if (_logger != null)
        _logger.close();
}
```

```
public static MyAdapter getInstance() {
    // TODO Auto-generated method stub
    if (_instance != null)
        return _instance;
    else {
        synchronized (MyAdapter.class) {
            if (_instance != null) {
                return _instance;
            }
            try {
                _instance = new MyAdapter();
                return _instance;
            } catch (Throwable t) {
                t.printStackTrace();
                return null;
            }
        }
    }
}
```

Creating Resource Bundles

A resource bundle describes the adapter resources to Integration Server. For example, connection factories, notification templates. You set the names of all connection factories and notification implementation classes in an `AdapterTypeInfo` object in your adapter's implementation of the `fillAdapterTypeInfo` method. Set these names when you implement the connections and notifications.

- A resource bundle contains all the display strings and messages used by the adapter at run time and at design time.

- A resource bundle is specific to a particular locale. If you plan to run your adapter in multiple locales, you can include multiple locale specific resource bundles. Creating a resource bundle for each locale enables you to internationalize an adapter quickly, without having to change any code in the adapter. An adapter must have one or more resource bundles.
- A resource bundle consists of lookup keys that provide locale specific objects (normally text strings). A lookup key consists of a constant provided by the `com.wm.adk.ADKGLOBAL` class (a class provided by the ADK's API) combined with your adapter's class name or a parameter name. For a list of these constants, see [“Resource Bundle Lookup Keys” on page 35](#). For example, the ADK uses the following lookup key whenever the display name of the adapter is required. (Assume that the `MyAdapter.getAdapterName` method returns the class name `MyAdapter` where `MyAdapter` is the `WmAdapter` implementation class.)

```
MyAdapter.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME
```

The following lookup key produces a description for a parameter named password, which is used to configure a connection pool:

```
"password" + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION
```

The adapter automatically references lookup keys to provide the following:

- Display names, and property descriptions of the:
 - Adapter definition
 - Connection types
 - Adapter service templates
 - Polling notification templates
- Text, display names, and property descriptions for the following elements of your adapter:
 - Parameters used to configure nodes
 - Metadata group names
 - Log entries
 - Exception text

Note:

You may also make explicit use of a resource bundle to localize other data, such as resource domain values, especially if those values are known at development time (for example, a list of known record status values). In these cases, the strategy for key composition is left to your discretion. For more information, see [“Using Resource Bundles with Resource Domain Values” on page 325](#). Resource domain values are discussed in detail in [“Resource Domains” on page 84](#).

Resource Bundle Lookup Keys

The following table describes the automatic resource bundle lookups performed for each type of adapter element, and describes how the results are used.

Adapter Element	Key Format	Usage
Adapter definition	<i>adapterName</i> + ADKGLOBAL. RESOURCEBUNDLEKEY_DISPLAYNAME	Required. Displays adapter name in Integration Server Administrator and in adapter interface.
	<i>adapterName</i> + ADKGLOBAL. RESOURCEBUNDLEKEY_DESCRIPTION	Required. Displays adapter description in adapter interface's About window.
	<i>adapterName</i> + ADKGLOBAL. RESOURCEBUNDLEKEY_VENDORNAME	Optional. Displays adapter vendor name in adapter interface's About window.
	<i>adapterName</i> + ADKGLOBAL. RESOURCEBUNDLEKEY_THIRDPARTYCOPYRIGHTURL	Optional. Displays adapter's copyright for third parties in the adapter interface's About window.
	<i>adapterName</i> + ADKGLOBAL. RESOURCEBUNDLEKEY_COPYRIGHTENCODING	Optional. Encoding used to display adapter's copyright for third parties in the adapter interface's About window.
Connection type	<i>ConnectionFactoryName.class.getName()</i> + ADKGLOBAL. RESOURCEBUNDLEKEY_DISPLAYNAME	Displays connection type column in adapter interface.
	<i>ConnectionFactoryName.class.getName()</i> + ADKGLOBAL. RESOURCEBUNDLEKEY_DESCRIPTION	Displays description column in connection type listing when configuring connections.
Adapter service template	<i>AdapterServiceName.class.getName()</i> + ADKGLOBAL. RESOURCEBUNDLEKEY_DISPLAYNAME	Displays adapter service template name when selecting a template to create an adapter service in Designer.
	<i>AdapterServiceName.class.getName()</i> + ADKGLOBAL. RESOURCEBUNDLEKEY_DESCRIPTION	Displays adapter service template description when selecting a template to create an adapter service in Designer.
Polling notification template	<i>AdapterNotificationName.class.getName()</i> + ADKGLOBAL. RESOURCEBUNDLEKEY_DISPLAYNAME	Displays template name column when selecting a template to create polling notification in Designer.

Adapter Element	Key Format	Usage
	<code>AdapterNotificationName.class.getName()</code> + <code>ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION</code>	Displays template description column when selecting a template to create polling notification in Designer.
Parameters used to configure nodes	<code>parameterName</code> + <code>ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME</code>	Displays property display name when editing connection, adapter service, and polling notification properties in adapter interface and Designer.
	<code>parameterName</code> + <code>ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION</code>	Displays tool tip when mouse is over parameter name when editing service and polling notification properties in Designer.
Metadata group names	<code>groupName</code> + <code>ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME</code>	Displays property display name when editing parameter group for polling notification in Designer.
	<code>groupName</code> + <code>ADKGLOBAL.RESOURCEBUNDLEKEY_GROUPURL</code>	Overrides Designer help hyperlink on each tab associated with group.
Log entries	<code>new Integer(minorCode).toString()</code>	Text in all log repositories (server locale)
Exception text	<code>new Integer(minorCode).toString()</code>	Text in all log repositories (server locale)

Note:

By default, if you do not specify the *DISPLAYNAME* lookup key in your resource bundle, the parameter or class name is used as the display name. Other lookup keys that are not specified, display nothing.

Considerations for Adapter Definition Lookup Keys

As specified in the preceding table, the lookup keys specifying the adapter's display name, description, and vendor name are required. All other lookup keys are optional, but not specifying the optional lookup keys, can result in generating error messages in the log, thereby making the adapter more difficult to use.

The *adapterName* reference in the Key Format column of the preceding table refers to the name of the adapter returned by the *WmAdapter* implementation class's method *getAdapterName*. You may use the name of your *WmAdapter* implementation class if it is also returned by the *getAdapterName* method.

Note:

The adapter name must be unique within the scope of Integration Server.

Alternatively, you may define a constant that both the lookup key and `getAdapterName` use. However, do not call `WmAdapter` implementation class's `getInstance.getAdapterName` method to retrieve the adapter name from either of these static initializers, or in the resource bundle constructor. Integration Server instantiates the resource bundle data during execution of the `WmAdapter` implementation class's constructor `super()`; calling `getInstance` method from that point produces undesirable results.

Considerations for Specifying URLs in Resource Bundles

Some lookup keys reference documents, such as copyright and other information. If you include these document files with your adapter, place them under `adapterPackageName\pub` (where `adapterPackageName` is the file system folder under the Integration Server's packages folder where your adapter resides). For more information about the Integration Server file system structure, see [“Creating an Adapter Package” on page 27](#).

In your lookup key, identify the resource file using the file path relative to the `pub` subfolder. For example, to specify the location of the copyright file for `MyAdapter`, a resource bundle would include the data value:

```
{ADAPTER_NAME +  
ADKGLOBAL.RESOURCEBUNDLEKEY_THIRDPARTYCOPYRIGHTURL,  
IS_PKG_NAME + "copyright.html"}
```

This relative path is equivalent to the following path:

```
Integration Server_directory\instance\<instance_name>\packages\MyAdapter\pub\copyright.html
```

This relative path scheme must be used for all URL references used by Designer as well as for the `ADKGLOBAL.RESOURCEBUNDLEKEY_THIRDPARTYCOPYRIGHTURL` clause. Other URL references accessed through the Integration Server Administrator or the adapter's administrative interface may use other URL referencing schemes such as absolute paths or valid Internet addresses.

Creating Resource Bundles Class With Example

> To create a resource bundle implementation class

1. Create a class by extending the base class `java.util.ListResourceBundle` in the same Java package that you created for `WmAdapter` implementation class. For example:
`com\mycompany\adapter\myadapter.`

Note:

You must create your class in the same `adapterPackageName\code\source` folder in which you created your `WmAdapter` implementation class in the `webMethods` package you created using Designer.

In the example, the `MyAdapterResource` class in folder `com\wm\MyAdapter`. In the example, the Java package created is `com\wm\MyAdapter`.

```
package com.wm.MyAdapter;  
import java.util.ListResourceBundle;  
import com.wm.adk.ADKGLOBAL;
```

```

public class MyAdapterResource extends ListResourceBundle implements
MyAdapterConstants{
    static final String IS_PKG_NAME = "/MyAdapter/";
    static final Object[][] _contents = {
        // adapter type display name.
        {ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME, "My Adapter"}
        // adapter type descriptions.
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Adapter for MyAdapter Server (a Sample System)"}
        // adapter type vendor.
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_VENDORNAME, "Software AG"}
        //Copyright URL Page
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_THIRDPARTYCOPYRIGHTURL,
        IS_PKG_NAME + "copyright.html"}
        //Copyright Encoding
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_COPYRIGHTENCODING, "UTF-8"}
        //About URL Page
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_ABOUT, IS_PKG_NAME + "About.html"}
        //Release Notes URL Page
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_RELEASENOTEURL, IS_PKG_NAME +
        "ReleaseNotes.html"}
    };
    protected Object[][] getContents() {
        // TODO Auto-generated method stub
        return _contents;
    }
}

```

2. You can create multiple resource bundle classes.

Use the following naming convention to create multiple resource bundle classes:

```
ResourceBundleName ResourceBundleName_locale
```

For example, a default resource bundle and a corresponding locale specific bundle for use in Japan might be named:

```
MyAdapterResourceBundle
MyAdapterResourceBundle_ja
```

For more information about resource bundle naming conventions, see the Javadoc for `java.util.ResourceBundle`.

Note:

For each resource bundle lookup, the adapter uses the default resource bundle if a bundle specific to the target locale is not available.

3. In your subclass, create resource bundle lookup keys.
4. Specify the adapter's default resource bundle in your `WmAdapter` implementation class. You can return the name of your default resource bundle using the `getAdapterResourceBundleName` method in your `WmAdapter` implementation class.

In the example, the *MyAdapter* class's `getAdapterResourceBundleName` method returns the constant `ADAPTER_SOURCE_BUNDLE_NAME`, which is initialized in interface *MyAdapterConstants* interface.

Example of `getAdapterResourceBundleName` method in *MyAdapter* class:

```
@Override
public String getAdapterResourceBundleName() {
    // TODO Auto-generated method stub
    return ADAPTER_SOURCE_BUNDLE_NAME;
}
```

Example of `ADAPTER_SOURCE_BUNDLE_NAME` constant in *MyAdapterConstants* interface:

```
static final String ADAPTER_SOURCE_BUNDLE_NAME =
    "com.wm.MyAdapter.MyAdapterResource";
```

5. Create the reference pages for copyright, and index page for the adapter in `adapterPackageName/pub` folder.
 - In the adapters administrative interface, select **About**, the about page appears with the display name, description and copyright.
 - In the Integration Server Administrator, select **Packages > MyAdapter > Home**, the index page appears.

Note:

You must create your reference pages in the same `adapterPackageName/pub` folder in the webMethods package you created using Designer.

Deploying the adapter

The users of the adapter explicitly load and unload an adapter by enabling and disabling the adapter package, using Integration Server Administrator. This section describes:

- Creating Adapter Startup and Shutdown Java Services.
- Compiling the Adapter.
- Registering the Adapter Startup and Shutdown Java Services in Integration Server.
- Debugging the Adapter.
- Loading the Adapter.
- Unloading the Adapter.

This section describes the basic steps for deploying, and debugging an adapter definition.

Creating Adapter Startup and Shutdown Java Services

The adapter must have one Java Service each for startup and shutdown.

- The startup method must retrieve an instance of your `WmAdapter` implementation class and pass it to `AdapterAdmin.registerAdapter` method.
- The shutdown service must perform the following:
 - Retrieve an instance of your `WmAdapter` implementation class and pass it to `AdapterAdmin.unregisterAdapter` method.
 - Perform the cleanup operations needed by your adapter. In most cases, this is accomplished by calling `MyAdapter.cleanup` method before the call to `AdapterAdmin.unregisterAdapter` method.

You can create the startup and shutdown Java Services in two ways:

1. Create adapter admin Java class, which is used to generate corresponding Java Services using `jcode` utility.
2. Create Java Services using Designer, which is compiled using Integration Server Administrator.

Creating Java Classes for Adapter Startup and Shutdown

1. Create a folder structure for the Java package for adapter admin class. For example: `adapterPackageName\code\source\wm\mycompany\adapteradmin`. In the example, the Java package created is `adapterPackageName\code\source\wm\MyAdapter`.

Note:

You must create your Java package and classes in the `adapterPackageName\code\source` folder in the `webMethods` package you created using Designer.

2. Create adapter admin Java class.

In the example, class `MyAdapterAdmin` is created:

```
package wm.MyAdapter;
//--- <<IS-START-IMPORTS>> ---
import com.wm.MyAdapter.*;
import com.wm.adk.admin.AdapterAdmin;
import com.wm.app.b2b.server.ServiceException;
import com.wm.data.IData;
//--- <<IS-END-IMPORTS>> ---
public class MyAdapterAdmin {

    public static final void startUp (IData pipeline)
        throws ServiceException
    {
        // --- <<IS-START(startUp)>> ---
        AdapterAdmin.registerAdapter(MyAdapter.getInstance());
        // --- <<IS-END>> ---
    }
    public static final void shutDown (IData pipeline)
```

```

        throws ServiceException
    {
        // --- <<IS-START(shutDown)>> ---
        MyAdapter instance = MyAdapter.getInstance();
        instance.cleanup();
        AdapterAdmin.unregisterAdapter(instance);
        // --- <<IS-END>> ---
    }
}

```

Note:

Tags are used to mark the beginning and end of imports and methods. For more information, see *webMethods Service Development Help*.

Creating Java Services for Adapter Startup and Shutdown using Designer

1. Start Designer.
2. Select the webMethods package you created using Designer.
3. Create a folder structure for the Java package for adapter admin Java Services. In the example, the folder structure created is *adapterPackageName\code\source\wm\MyAdapter\MyAdapterAdmin*.

Note:

You must create your Java package and classes in the *adapterPackageName\code\source* folder in the webMethods package you created using Designer.

4. Create a new Java Service for adapter startup. In the example, the Java Service created is *startUp*.
 - a. Add the following Java code.

```

public static final void startUp(IData pipeline) throws ServiceException {
    // --- <<IS-START(startUp)>> ---
    AdapterAdmin.registerAdapter(MyAdapter.getInstance());
    // --- <<IS-END>> ---
}

```

Note:

Tags are used to mark the beginning and end of imports and methods. For more information, see *webMethods Service Development Help*.

5. Create a new Java Service for adapter shutdown. In the example, the Java Service created is *shutDown*.
 - a. Add the following Java code.

```

public static final void shutDown(IData pipeline) throws ServiceException {
    // --- <<IS-START(shutDown)>> ---
    MyAdapter instance = MyAdapter.getInstance();
    instance.cleanup();
}

```

```

    AdapterAdmin.unregisterAdapter(instance);
    // --- <<IS-END>> ---
}

```

Note:

Tags are used to mark the beginning and end of imports and methods. For more information, see *webMethods Service Development Help*.

Compiling the Adapter

Before you load your adapter, you must compile your implementation classes and construct the Java Service nodes for your startup and shutdown services.

1. Create an ANT script to compile the adapter implementation and admin classes, and deploy these classes in Integration Server as Java Services. In the example, the ANT script created is `build.xml` and `build.properties`.

Note:

You must create your ANT script in the `adapterPackageName\code\source` folder in the `webMethods` package you created using Designer.

For example `build.properties`:

```

# The Site Name
debug=on
optimize=off
deprecation=off
webM.home=C:/softwareag/912
server.home=${webM.home}/IntegrationServer
package=MyAdapter
instance_name=default
srcdir=${server.home}/instances/${instance_name}/packages/${package}/code/source
destdir=${server.home}/instances/${instance_name}/packages/${package}/code/classes

```

For example `build.xml`:

```

<?xml version="1.0"?>
<project name="Adapter using ADK" default="deploy" basedir=".">
  <property file="build.properties" />
  <!-- classes belonging to this package -->
  <path id="this.package.classpath">
    <fileset dir="${server.home}/instances/${instance_name}/packages/${package}/">
      <include name="code/classes"/>
    </fileset>
  </path>
  <!-- All classes that need to be found by this script -->
  <path id="total.classpath">
    <pathelement
location="${server.home}/instances/${instance_name}/packages/WmART/code/jars/wmart.jar"/>
    <pathelement location="${server.home}/lib/wm-isserver.jar"/>
    <pathelement location="${webM.home}/common/lib/wm-isclient.jar"/>
    <pathelement location="${webM.home}/common/lib/glassfish/gf.jakarta.resource.jar"/>
    <pathelement
location="${server.home}/instances/${instance_name}/packages/WmART/code/classes"/>

```

```

<path refid="this.package.classpath"/>
</path>

<!-- Compile the java files of this package -->
<target name="createclasses" depends="init">
  <echo>Creating classes</echo>
  <mkdir dir="${destdir}"/>
  <javac debug="${debug}" optimize="${optimize}"
    deprecation="${deprecation}" srcdir="${srcdir}"
    destdir="${destdir}">
    <classpath>
      <path refid="total.classpath"/>
    </classpath>
  </javac>
</target>
<!-- Execute jcode -->
<target name="execjcode" depends="createclasses">
  <echo>Deploying classes</echo>
  <exec executable="${server.home}/instances/${instance_name}/bin/jcode"
    vmlauncher="false" failonerror="true">
    <arg value="fragall" />
    <arg value="${package}" />
  </exec>
</target>
<!-- delete .class files built in this package -->
<target name="cleanclasses">
  <echo>Cleaning classes</echo>
  <mkdir dir="${destdir}"/>
  <delete quiet="false">
    <fileset dir="${destdir}" includes="**/*.class"/>
  </delete>
</target>

<!-- if this package depends on classes found in other packages,
  add targets to build those classes here. -->
<target name="init">
  <tstamp/>
</target>

<target name="packageDependencies" depends="" />
<target name="clean" depends="cleanclasses" />
<target name="classes" depends="cleanclasses, createclasses" />
<target name="deploy" depends="execjcode" />
<target name="all" depends="packageDependencies, cleanclasses, execjcode" />
<target name="remake" depends="packageDependencies, cleanclasses, createclasses"
/>
</project>

```

2. Set the classpath in *total.classpath* in the ANT script.

The JAR files required to compile your source code are as follows:

- *Software AG_directory* \common\lib\wm-isclient.jar
- *Software AG_directory* \common\lib\glassfish\gf.jakarta.resource.jar
- *Integration Server_directory* \lib\wm-issserver.jar

- *Integration Server_directory* \instances*<instance_name>*\packages\WmART\code\jars\wmart.jar

The folder containing the class files required to compile your source code is as follows:

- *Integration Server_directory* \instances*<instance_name>*\packages\WmART\code\classes

Software AG_directory is the folder in which webMethods components are installed and *Integration Server_directory* is the folder in which Integration Server is installed.

Note:

- Add the required folders location to your classpath, or package all folders in a JAR and then add the JAR to your classpath.
- You must specify JDK version 1.8 or higher in your classpath.

3. Run the ANT script to compile the Java classes.

```
ant classes
```

4. Compile the classes and deploy in Integration Server..

- a. Run the ANT script to create classes and deploy in Integration Server as Java Services.

```
ant deploy
```

Note:

If you have created adapter admin Java class for adapter startup and shutdown, then the corresponding Java Services are deployed using jcode utility. The jcode utility is provided with Integration Server. For more information, see *webMethods Service Development Help*.

- b. If you have created Startup and Shutdown Java Services using Designer, you must compile it using Integration Server Administrator.

- Start Integration Server Administrator.
- Select **Settings > Extended > Edit Extended Settings**.
- Set the property `watt.server.compile` to include the path to Java compiler and the classpath to include the `wmart.jar` in *Integration Server_directory* \instances*<instance_name>*\packages\WmART\code\jars\wmart.jar. For example:

```
watt.server.compile=C:\softwareag\912\jvm\jvm\bin\javac
-classpath
{0};C:\softwareag\912\IntegrationServer\instances\default\packages\
WmART\code\jars\wmart.jar; -d {1} {2}
```

5. Restart Integration Server.

If your startup and shutdown Java Services do not appear in the adapter package, there has been an error either in compiling your code or in creating the Java Services.

Registering the Adapter Startup and Shutdown Java Services in Integration Server

Before you register the adapter startup and shutdown Java Services for the adapter package, make sure that you have:

- Created your adapter startup and shutdown Java Service.
 - Successfully compiled the adapter.
1. Start Designer.
 2. In the **Package Navigator**, select the webMethods package you created.
 3. Set the **Startup Services** and **Shutdown Services**.

Perform the following operations:

- In the **Properties > StartUp/Shutdown Services > Startup Services**, add the startup service created.
 - In the **Properties > StartUp/Shutdown Services > Shutdown Services**, add the shutdown service created.
4. Restart Integration Server.
 5. Start Integration Server Administrator.
 6. In Integration Server Administrator, select **Adapters**.

You can see the adapter you have created in the dropdown.

Debugging the Adapter

1. Shut down your Integration Server .
2. Modify the script *Software AG_directory* \profiles\IS_<instance_name>\bin\startDebugMode.bat (or startDebugMode.sh in a UNIX environment).

Set SUSPEND_MODE to n.

```
set SUSPEND_MODE=n
```

3. Start Integration Server in the debug mode by executing startDebugMode.bat (or startDebugMode.sh in a UNIX environment).
4. Create a script to attach the debugger to Integration Server.

For example *debug.bat*:

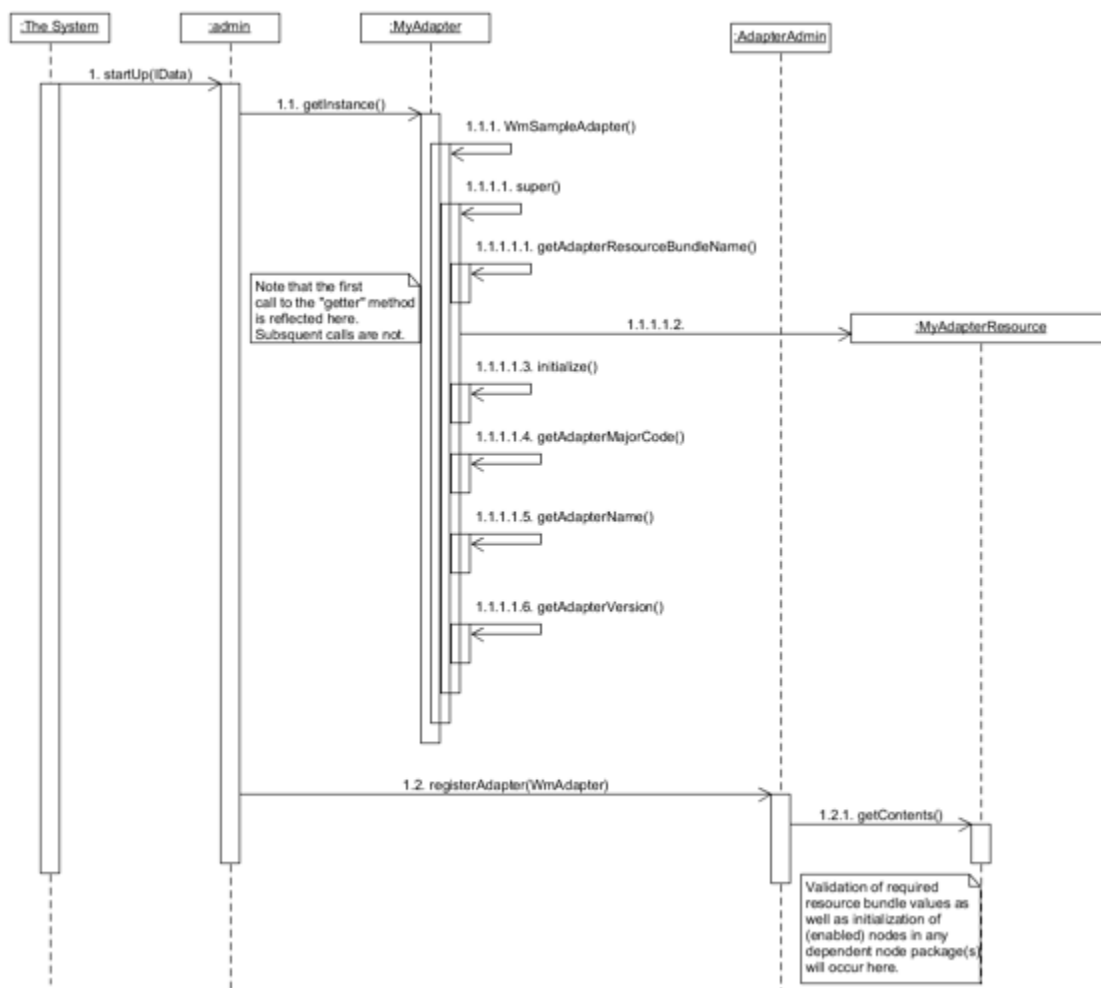
```
rem @echo off
set
TARGET="C:\softwareag\912\IntegrationServer\instances\default\packages\MyAdapter\code\source"
set JAVA_MIN_MEM=64M
set JAVA_MAX_MEM=64M
set JAVA_MEMSET=-ms%JAVA_MIN_MEM% -mx%JAVA_MAX_MEM%
set JAVA_DBG="C:\softwareag\912\jvm\jvm\bin\jdb"
set ARGS=com.sun.jdi.SocketAttach:hostname=localhost,port=10033
%JAVA_DBG% -sourcepath %TARGET% -connect %ARGS%
```

5. Run the debug script.

debug.bat (or *debug.sh* in a UNIX environment).

You can attach to Integration Server and "step" your code (assuming you compiled your code to include debug symbols (for example, `javac -g`) using your favorite debugger.

Adapter Load Process

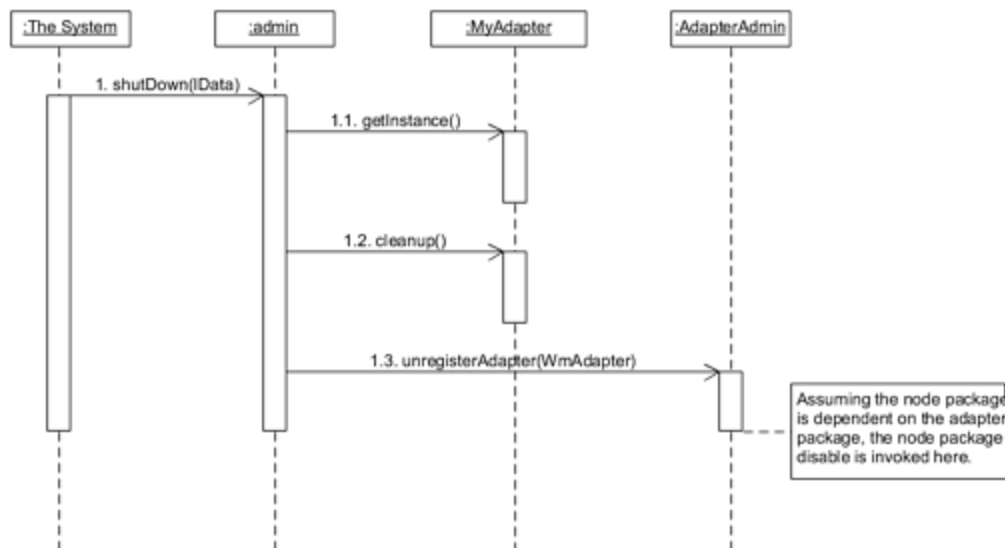


The users of the adapter explicitly load and unload an adapter by enabling and disabling the adapter package, using Integration Server Administrator. This figure shows how *MyAdapter* loads the adapter into Integration Server at run time.

The loading process is described as follows:

1. The system calls `AdapterAdmin.registerAdapter`. This method is the designated package startup service.
2. The implementation of `registerAdapter` instantiates the *MyAdapter* class by calling its `getInstance` method. It then registers the *MyAdapter* object as an adapter by passing it to the `registerAdapter` method of the `AdapterAdmin` class provided by the ADK.
3. During the construction of the *MyAdapter* instance, `super()` calls the base class `com.wm.adk.WmAdapter` constructor, which instantiates the default resource bundle, calls `MyAdapter.initialize`, and retrieves basic information about the adapter. (The `com.wm.adk.WmAdapter` constructor only instantiates the default resource bundle; it instantiates and reads other resource bundles if a request is received using the locale of that supplementary resource bundle.)
4. The adapter load process is completed with a call to the static method `AdapterAdmin.registerAdapter`, which reads and evaluates the adapter's default resource bundle, and updates the Integration Server list of registered adapters. If the resource bundle does not provide the required resource bundle elements, the registration process fails, with an error log entry explaining the failure.
5. Upon completion of the adapter's registration, the adapter loads any dependant node packages. See the chapters about connections, adapter services, polling notifications, and listener notifications for descriptions of their respective load processes.
6. If the load was successful, the adapter name (as specified in the resource bundle) appears in the list of adapters in Integration Server Administrator. If it does not appear, refresh your browser page. If it still does not appear, check the log for errors.

Adapter Unload Process



During server shutdown or package reload, the system disables the adapter's dependant nodes (or their packages) before it disables the adapter. When you explicitly disable an adapter's package using Integration Server Administrator, disable its dependant nodes first. The diagram shows how *MyAdapter* unloads the adapter from Integration Server at run time.

To unload an adapter, the server calls `AdapterAdmin.unregisterAdapter`. This method is the designated package shutdown service. This method performs the following tasks:

1. Retrieves the adapter instance.
2. Calls the adapter's cleanup method (if implemented).
3. Calls `AdapterAdmin.unregisterAdapter`.

Reporting Adapter Fix Levels

After you deploy an adapter, you may need to deliver a fix. You can identify the fix to Integration Server through the adapter's package manifest file. Whenever the adapter package is loaded, the **patch_history** field is read from the file and displayed on the adapter's About page. The management of this field is the responsibility of the adapter developer. The Integration Server facilities for creating partial package installations can be used to deliver adapter fixes. For more information, see the *webMethods Integration Server Administrator's Guide* for your release.

Package Management

A namespace node package is a package in which the user of the adapter creates the adapter's namespace nodes for connections, adapter services, polling notifications, listener notifications, and listeners. The user of the adapter must not create nodes in the adapter package. Keeping

namespace nodes separate from the adapter package simplifies the process of upgrading a deployed adapter.

The procedure for creating a namespace node package is identical to the procedure for creating any webMethods package. For instructions for creating packages, see the *webMethods Service Development Help* for your release.

All the nodes of an adapter may be located in one package, or they may be distributed among multiple packages. **Namespace node packages** management tasks include:

- Setting package dependencies for namespace node packages.
- Setting package dependencies for namespace nodes.
- Using access control lists (ACLs) to control which development group has access to which adapter services
- Enabling and Disabling Packages.
- Loading, Reloading, and Unloading Packages.

To perform these tasks, the user of the adapter must use webMethods Integration Server, Designer, and a web browser.

Note:

You must have Integration Server administrator privileges to access an adapter's management screen. For information about setting user privileges, see the *webMethods Integration Server Administrator's Guide* for your release.

Package Dependency Considerations for Namespace Node Packages

Following are dependency requirements and guidelines for namespace node packages:

- A namespace node package must have a dependency on its associated adapter package, and the adapter package must have a dependency on the WmART package. For more information about setting package dependencies, see the *webMethods Service Development Help* for your release.

Setting these dependencies ensures that at startup Integration Server automatically loads or reloads all packages in the proper order:

- WmART package.
- Adapter package.
- Node package(s).

The WmART package is automatically installed when you install Integration Server. You must not manually reload the WmART package.

- Keep connections for different adapters in separate packages so that you do not create interdependencies between adapters. If a package contains connections for two different

adapters, and you reload one of the adapter packages, the connections for both adapters is reloaded automatically.

- You cannot enable a package if it has a dependency on another package that is disabled. That is, before you can enable your package, you must enable all packages on which your package depends. For information about enabling packages, see [“Enabling and Disabling Packages” on page 51](#).
- You can 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 your adapter package before you disable the adapter package.

If the namespace nodes of an adapter are located in multiple packages, see [“Package Dependency Considerations for Namespace Nodes” on page 51](#). For more information about setting package dependencies, see the *webMethods Service Development Help* for your release.

Package Dependency Considerations for Namespace Nodes

If all the namespace nodes of an adapter are located in the same package, there is no need to set package dependencies.

However, if the nodes of an adapter are located in multiple packages, then:

- A package that contains the connection node(s) must depend on the adapter package.
- Packages that contain other nodes must depend on their associated connection package.

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

Group Access Control

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

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

Enabling and Disabling Packages

All packages are automatically enabled by default. To prevent Integration Server from loading a particular package, you must manually disable that package.

Enabling Packages

To enable the package:

- Start Integration Server Administrator.

- Click **Packages > Management**.
- In the **Management** screen, click **No** in the **Enabled** column for the package. The **No** changes to **Yes** (enabled).

Note:

Enabling an adapter package will not cause its associated node package(s) to be reloaded.

Important:

Before you manually enable a node package, *you must first enable its associated adapter package*. Similarly, if your adapter has multiple node packages, and you want to disable some of them, disable the adapter package first. Otherwise, errors will be issued when you try to access the remaining enabled node packages.

Disabling Packages

To disable the package:

- Start Integration Server Administrator.
- Click **Packages > Management**.
- In the **Management** screen, click **Yes** in the **Enabled** column for the package. The **Yes** changes to **No** (disabled).

Disabled packages are not listed in Designer. A disabled adapter will remain disabled until you explicitly enable it using Integration Server Administrator.

Loading, Reloading, and Unloading Packages

Loading Packages

If the node packages are properly configured with a dependency on the adapter package, at startup Integration Server automatically loads or reloads all packages in the proper order:


- WmART package.
- Adapter package.
- Node package(s).

You must not manually reload the WmART package.

Reloading Packages Manually

You must typically reload the adapter package manually as you make changes to the adapter code. Similarly, the users of the adapter must reload their node packages manually when they modify the nodes. Reloading a node package will not cause its associated adapter package to be reloaded.

You can reload the adapter packages and the node packages by performing either of the following:

- In Integration Server Administrator, in the **Management** screen, click the  reload icon in the **Reload** column.
- In Designer, right click the package and select the **Reload Package** option from the menu.

Unloading Packages

At shutdown, Integration Server unloads the packages in the reverse order in which it loaded them:

- Node package(s).
- Adapter package.
- WmART package (assuming the dependencies are correct).

4 Connections

■ Overview	56
■ Adapter Connection Classes	58
■ Adapter Connection Implementation Classes	60
■ Creating a WmManagedConnection Implementation Class	61
■ Creating a WmManagedConnectionFactory Implementation Class	61
■ Updating the Resource Bundle	66
■ Registering Connection Factories in the Adapter	67
■ Connection Class Interactions	67
■ Configuring and Testing Connection Nodes	71

Overview

An adapter connection connects to an adapter resource. This chapter describes the classes provided by the ADK to support connections, and how to create an adapter connection implementation.

Connection Factories

The ADK's adapter connection model uses a factory method pattern in which a connection factory object is responsible for creating connection objects. In many cases, both the factory and its connections wrap comparable functionality provided in the resource's libraries. For example, an adapter connection factory may wrap a data source class provided by a database vendor, from which it creates the database connections and wraps them in an adapter connection object produced by the factory.

Each connection factory in an adapter implementation constitutes a *connection type* on the adapter's administrative interface. You can define one or more connection types for an adapter. If you need a different set of configuration parameters, create another connection type. For example, if you have a request that requires special security requirements, create a separate connection type for it.

A connection factory is also responsible for defining implementation-specific parameters and for making them available to the connection. The adapter uses these parameters at design time, when the users of the adapter create connection namespace nodes. For example, note the following connection type configuration window of the Sample Adapter.

Configure Connection Type > Sample Adapter	
Package	Default
Folder Name	<input type="text"/>
Connection Name	<input type="text"/>
Connection Properties	
Host Name	<input type="text"/>
Port	4444
User Id	<input type="text"/>
Password	<input type="text"/>
Retype Password	<input type="text"/>
Local Transaction Control	true
Connection Timeout	20000
Connection Management Properties	
Enable Connection Pooling	true
Minimum Pool Size	1
Maximum Pool Size	10
Pool Increment Size	1
Block Timeout (msec)	1000
Expire Timeout (msec)	1000
Startup Retry Count	5
Startup Backoff Timeout (sec)	10

Connection Management Properties

Each field in the Connection Management Properties section shown in the adapter's administrative interface has a corresponding metadata parameter provided by the associated connection factory. At run time, the node passes the user of the adapters' settings for these fields to the connection factory, which is then used for each connection that the factory creates.

The two fields in **Connection Management Properties** section that pertain to initializing the connection pool at startup are as follows:

- **Startup Retry Count** specifies the number of times that the system attempts to initialize the connection pool at startup if the initial attempt fails, before issuing an `AdapterConnectionException`.
- **Startup Backoff Timeout** specifies the number of seconds to wait between each attempt to initialize the connection pool.

These fields provide flexibility in managing connections in environments where network anomalies are commonplace. They are significant in the following situations:

- When a new connection is enabled; when Integration Server starts.
- When the package containing a previously enabled connection node is reloaded.

For information about other connection management parameters, see [“Configuring and Testing Connection Nodes” on page 71](#).

Connection Management

Integration Server includes a connection management service that dynamically manages connections and connection pools based on the settings stored in the connection namespace node (such as the connection pooling and timeout fields specified in the Connection Management Properties section shown in the adapter's administrative interface).

When a connection namespace node is enabled, Integration Server uses the connection factory to initialize the pool, creating a number of connection instances equal to the minimum configured pool size.

- When a connection is needed by an adapter service or notification, the ADK provides a connection from the pool.
- If no connections are available in the pool, and the maximum pool size has not been reached, a new connection is retrieved from the connection factory.
- If the pool is full, the requesting thread blocks the amount indicated in the Block Timeout field until a connection becomes available.

For information about configuring your connection pool, see [“Configuring and Testing Connection Nodes” on page 71](#).

Creating Connection Implementation

To create a connection implementation, you perform the following tasks:

- Create an adapter connection class by extending the `com.wm.adk.connection.WmManagedConnection` base class. In this class, create methods that:
 - Wrap the connection to your resource.
 - Instantiates the class on receiving information from the connection factory.
 - Access the resource. Accessing the resource is your responsibility.
- Create an adapter connection factory class by extending the `com.wm.adk.connection.WmManagedConnectionFactory` base class. In this class, you create services and methods that:
 - Construct a new connection object.
 - Identify all adapter service templates supported by the factory's connections.
 - Specify the transactional capabilities of the factory's connections.
 - Create webMethods metadata for the connection factory.

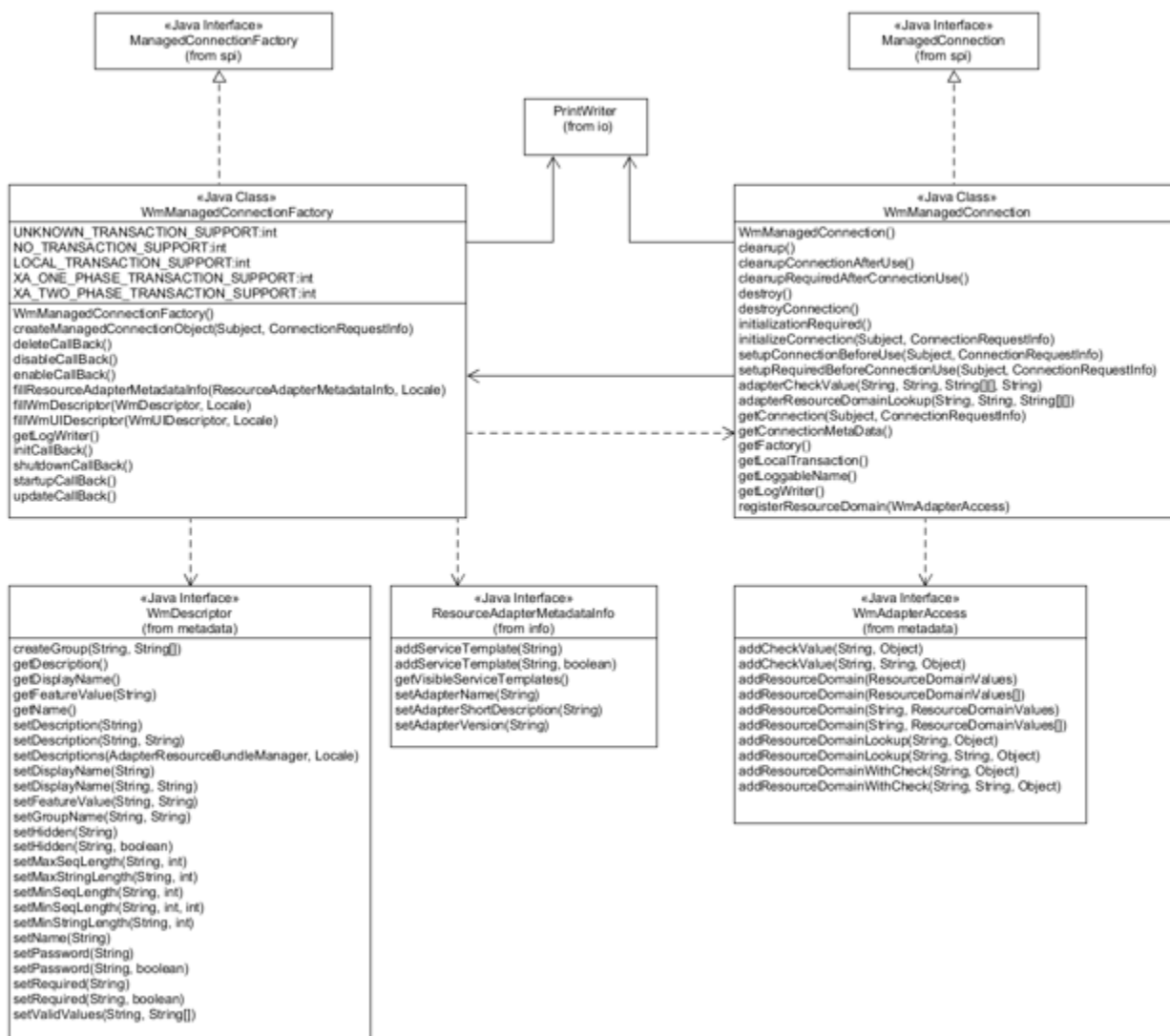
For more information, see [“Creating a WmManagedConnectionFactory Implementation Class” on page 61](#).

- Update the adapter's resource bundle with display names and other display-oriented data for the connection implementation and its parameters.
- Register the connection type in the adapter.
- Compile and reload your adapter.
- Configure and test the connection.

Adapter Connection Classes

ADK Adapter Connection Classes

This diagram displays the ADK classes, class members, and class methods used to create an adapter connection.



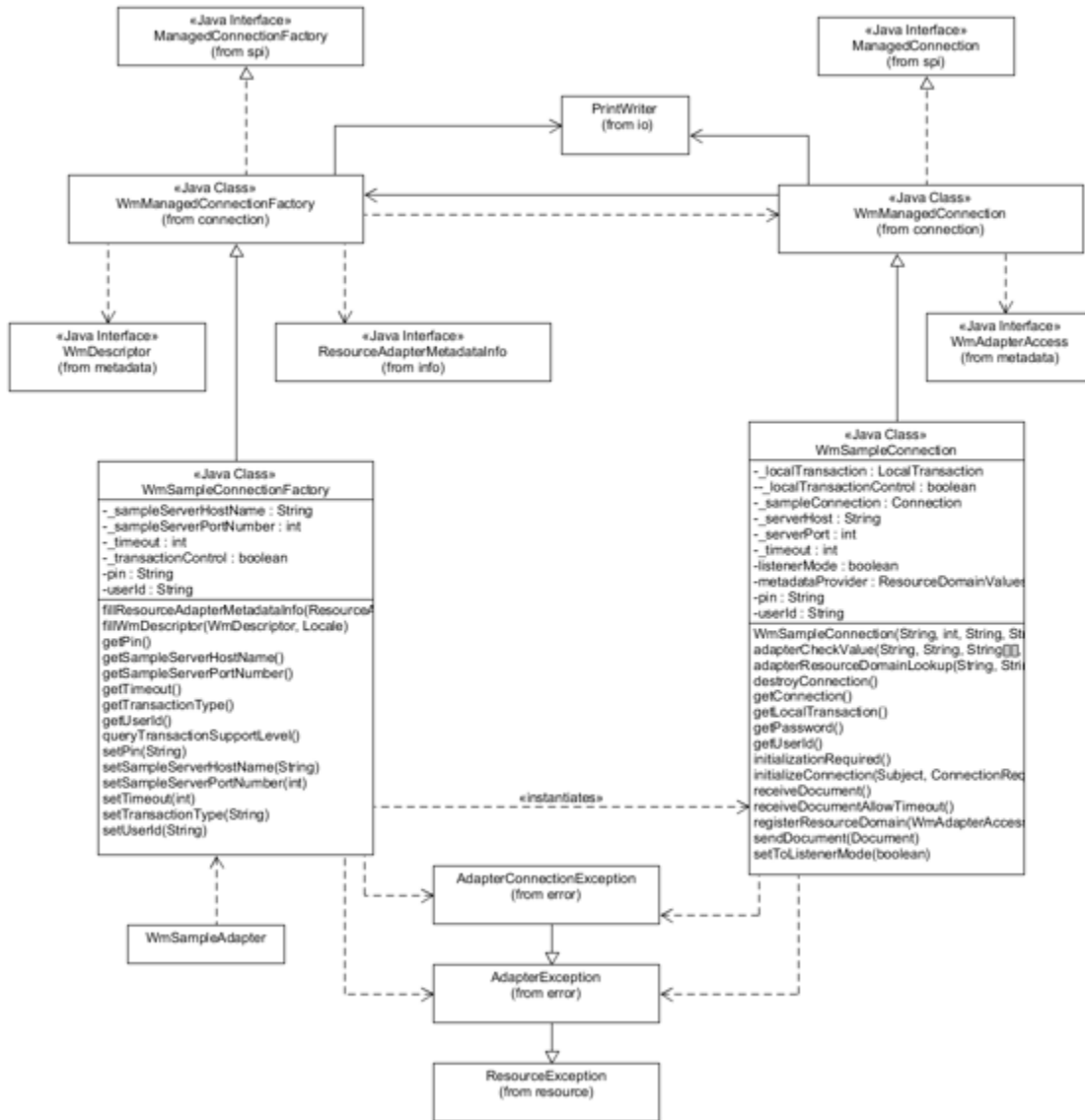
External Interfaces Used in Adapter Connection Classes

The ADK's base connection classes are dependent on the following external interfaces. These interfaces are used as arguments of the abstract methods that you must override in the connection implementation classes:

Interface	Description
com.wm.adk.metadata.WmDescriptor	Creates webMethods metadata.
com.wm.adk.info.ResourceAdapterMetadataInfo	Specifies adapter services supported by the connection. This interface is an extended version of <code>javax.resource.cci.ResourceAdapterMetadata</code> .
com.wm.adk.metadata.WmAdapterAccess	Used in conjunction with the metadata operations for the associated adapter services and notifications.

Adapter Connection Implementation Classes

The diagram shows how a typical adapter implements the connection classes.



At design time, each connection factory in an adapter implementation constitutes a *connection type* on the adapter's administrative interface. For a connection factory to be recognized as part of the adapter, you must specify the class in the `fillAdapterTypeInfo` method in the adapter's `WmAdapter` implementation class (see [“Registering Connection Factories in the Adapter”](#) on page 67).

For more information about how Integration Server uses connection classes at design time to support the creation and management of connection namespace nodes, see [“Connection Class Interactions”](#) on page 67.

Note:

The connection implementation classes often use both the `AdapterException` class and the `AdapterConnectionException` class. The main difference between these classes is the impact on the connection pool (see [“Receiving AdapterConnectionExceptions” on page 71](#)).

Creating a `WmManagedConnection` Implementation Class

Create an adapter connection class by extending the `com.wm.adk.connection.WmManagedConnection` base class. This class is primarily responsible for wrapping the connection to your resource. The method that instantiates the class receives information from the connection factory. Accessing the resource is your responsibility. You must override the following base class methods in your `WmManagedConnection` implementation class:

Method	Description
<code>adapterCheckValue</code>	Validates the user-supplied parameter values in service or notification editor, if <code>ResourceDomainValues.setCanValidate(true)</code> is set. For more information, see “Connection Callbacks” on page 62 .
<code>registerResourceDomain</code>	Registers the resource domains. For more information, see “Registering Resource Domains” on page 84 .
<code>adapterResourceDomainLookup</code>	Determines the new value or values that is applied to the parameter in service or notification editor. For more information, see “Populating Resource Domains with Values” on page 88 .
<code>destroyConnection</code>	Called when the connection is removed from the pool, releasing implementation specific resources.
<code>initializationRequired</code>	Returns <code>true</code> if <code>initializeConnection</code> is present in the implementation class.
<code>initializeConnection</code>	Called once if <code>initializationRequired</code> returns <code>true</code> . Optional.

Creating a `WmManagedConnectionFactory` Implementation Class

Create an adapter connection factory class by extending the `com.wm.adk.connection.WmManagedConnectionFactory` base class. You must override the following base class methods in your `WmManagedConnectionFactory` implementation class:

Method	Description
<code>createManagedConnectionObject</code>	Constructs a new connection object (for example, <code>SimpleConnection</code>).
<code>queryTransactionSupportLevel</code>	Specifies the transactional capabilities of this factory's connections. For more information, see this method's Javadoc.
<code>fillWmDescriptor</code>	Supports webMethods metadata constructs. For more information, see “WmDescriptor Interface” on page 65 .

Method	Description
deleteCallBack	Called when a connection factory is deleted.
disableCallBack	Called when a connection factory is disabled.
enableCallBack	Called when a connection factory is enabled.
initCallBack	Called when a connection factory is initialized.
startupCallBack	Called when a connection factory is started.
shutdownCallBack	Called when a connection factory is stopped.
updateCallBack	Called when a connection factory is updated.

Connection Callbacks

The `WmManagedConnectionFactory` base class defines a set of callback methods that you can override in any connection factory implementation class. These callbacks are called when the state changes on the connection node.

The following table describes which methods are called on a connection for certain operations.

User Actions	Callbacks Received By Enabled Connection
Enables a disabled connection	<ul style="list-style-type: none"> ■ enableCallBack ■ startupCallBack
Disables an enabled connection	<ul style="list-style-type: none"> ■ shutdownCallBack ■ disableCallBack
Enables a package containing an enabled connection	<ul style="list-style-type: none"> ■ initCallBack ■ startupCallBack
Disables a package containing an enabled connection	<ul style="list-style-type: none"> ■ shutdownCallBack
Deletes a package containing an enabled connection	<ul style="list-style-type: none"> ■ shutdownCallBack
Reloads a package containing an enabled connection	<ul style="list-style-type: none"> ■ shutdownCallBack ■ initCallBack ■ startupCallBack
Starts Integration Server	<ul style="list-style-type: none"> ■ initCallBack ■ startupCallBack

User Actions	Callbacks Received By Enabled Connection
	Note: This only occurs if the connections are contained in a package that is enabled.
Shuts down Integration Server	<ul style="list-style-type: none"> ■ shutdownCallBack Note: This only occurs if the connections are contained in a package that is enabled.

The following table describes which methods are called on a disabled connection for certain operations.

User Actions	Callbacks Received By Disabled Connection
Creates a connection	initCallBack
Updates a connection	updateCallBack
Deletes a connection	deleteCallBack
Copies a connection	initCallBack Note: This only occurs if the connection copied creates a new connection.
Enables a package containing a disabled connection	initCallBack
Reloads a package containing a disabled connection	initCallBack
Starts Integration Server	initCallBack Note: This only occurs if the connections are contained in a package that is enabled.

Note:

If more than one callback is listed, the callbacks occur in the specified order from top to bottom. If an action is not listed in the table then no callbacks occur.

Metadata Model for Connection

Each adapter interface field for configuring connection types must have a corresponding metadata parameter, provided by the associated connection factory. The webMethods metadata models are

designed to define, refine, organize, and constrain parameters used to configure namespace nodes for connections, adapter services, and notifications. The metadata model for connections is the foundation for the more complicated metadata model used to configure adapter services and notifications.

Each metadata parameter is identified by a *set* accessor method. For example:

```
public void setHostName(String name);
```

```
public void setPortNumber(int portNumber);
```

Metadata Parameter Names

You derive the name of a parameter from the name of its *set* method. You remove the prefix (*set*) and make the first letter lower case. Thus, the *set* methods above would define metadata parameters named *serverName* and *portNumber*. For example:

```
public void setHostName(String name) {  
}
```

```
public void setPortNumber(int portNumber) {  
}
```

The complete naming convention rules follow the Java bean property naming conventions.

Some naming variations include:

```
setFoo() -> parameter name is "foo"  
setfoo() -> parameter name is "foo"  
setF00() -> parameter name is "F00"  
setF0o() -> parameter name is "Foo"  
set_foo() -> parameter name is "_foo"  
set_Foo() -> parameter name is "_Foo"
```

Metadata Parameter Arguments

A metadata parameter's *set* method must accept a single argument, whose data type defines the data type of the parameter. For connections, this type must be limited to a Java primitive or a `java.lang.String` types. Arrays (or sequence parameters) are allowed, but the adapter's administrative interface only provides widgets to access the first element of the array. Other object types are interpreted as external Java bean classes, as described in [“Implementing Metadata Parameters Using External Classes” on page 308](#).

In addition to defining the parameter name and data type, the *set* method of a metadata parameter is used at run time to pass the value of the parameter to the connection factory. In the example, the metadata accessor methods are shown as methods of the connection factory implementation class. In fact, any method in the connection factory implementation conforming to the naming convention is interpreted as a metadata parameter accessor method.

Metadata Parameter *get* Accessor Methods

A metadata parameter may have a corresponding *get* accessor method (following the same naming convention) that returns the same data type as the argument of the *set* method. The adapter uses these *get* methods only to retrieve default parameter values when creating a new connection.

Creating a *get* method with a different data type than its corresponding *set* method results in an error. In addition, creating a *get* method without a *set* method produces a parameter whose values are unusable.

Note:

All namespace nodes store parameter settings based on the parameter name. If you delete or change the name of your accessor methods, the parameter names stored in the namespace nodes associated with that class are no longer valid. From that point forward, any use of that node (at design time or at run time) fails. If you no longer need a metadata parameter after upgrading a deployed adapter, hide the parameter (using `WmDescriptor.setHidden`) instead of deleting it.

WmDescriptor Interface

The `com.wm.adk.metadata.WmDescriptor` interface controls how metadata parameters are handled at design time, during data entry.

To create webMethods metadata for your connection factory, use the `WmDescriptor` interface within the `WmManagedConnectionFactory.fillWmDescriptor` method. The table lists some commonly used `WmDescriptor` methods.

Name	Description
<code>createGroup</code>	Specifies the order in which parameters must appear on the adapter's administrative interface. Group names are not displayed.
<code>setDescriptions</code>	Sets the parameter display name, description, group name, and online help links for the current connection type. This call queries your resource bundle for <code>setDescriptions</code> information about the connections, its parameters, and its groups. The <code>ResourceBundleManager</code> argument must be identical to <code>WmAdapter.getResourceBundleManager</code>
<code>setDisplayname</code>	Assigns a user friendly parameter name. Alternatively, use <code>setDescriptions</code> to use a localized display name.
<code>setMinStringLength</code>	Specifies the minimum length of an input value for a parameter of type string.
<code>setMaxStringLength</code>	Specifies the maximum length of an input value for a parameter of type string.
<code>setPassword</code>	Displays asterisks when the users of the adapter enter passwords. This method also displays a confirmation field in which users must re-enter passwords. Only one copy of the parameter value is set in the connection factory.

Note:

Name	Description
	Do not use <code>setPassword</code> on more than one parameter per <code>WmDescriptor</code> .
<code>setValidValues</code>	Specifies the list of valid values for a parameter. These values appear in a dropdown.

Note:

The `WmDescriptor` methods apply to specified metadata parameter. In these cases, the name of the parameter must be passed as a `String`.

Updating the Resource Bundle

In the `WmManagedConnectionFactory` implementation class, the call to `WmDescriptor.setDescriptions` in the `fillWmDescriptor` implementation causes Integration Server to look for display names and other display data in the adapter's resource bundle. Update the `java.util.ListResourceBundle` implementation with entries for the `WmManagedConnectionFactory` implementation and its parameters.

In the example, update `MyAdapterResource` class's `Object[][] _contents` with the following:

```
package com.wm.MyAdapter;
import java.util.ListResourceBundle;
import com.wm.adk.ADKGLOBAL;
import com.wm.MyAdapter.connections.SimpleConnectionFactory;
public class MyAdapterResource extends ListResourceBundle implements MyAdapterConstants{
    static final String IS_PKG_NAME = "/MyAdapter/";
    static final Object[][] _contents = {
        ..
        ..
        ..
        //SimpleConnection
        ,{SimpleConnectionFactory.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Simple Connection"}
        ,{SimpleConnectionFactory.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Simple framework for demonstration purposes"}
        ,{SimpleConnectionFactory.SIMPLE_SERVER_HOST_NAME +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Host Name"}
        ,{SimpleConnectionFactory.SIMPLE_SERVER_PORT_NUMBER +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Port"}
    };

    protected Object[][] getContents() {
        // TODO Auto-generated method stub
        return _contents;
    }
}
```

For more information about resource bundles, see [“Creating Resource Bundles Class With Example” on page 38](#).

Registering Connection Factories in the Adapter

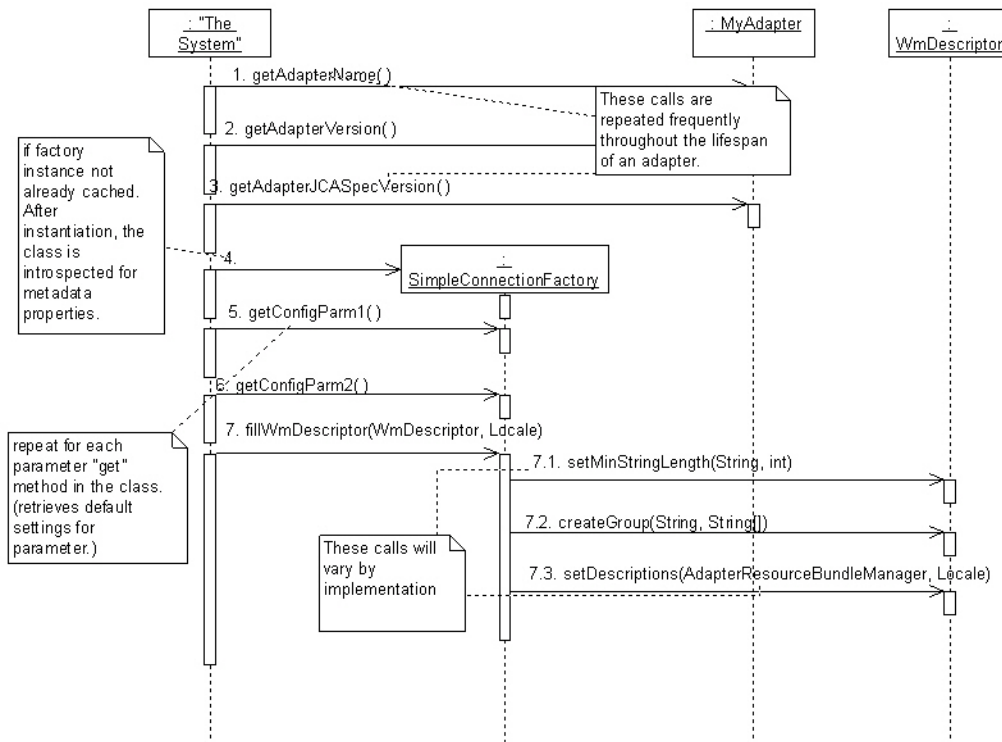
You must register each connection factory class in the `WmAdapter` implementation class. You do this by passing the class name to the `AdapterTypeInfo.addConnectionFactory` method in the `WmAdapter.fillAdapterTypeInfo` method in `WmAdapter` implementation class. In the example the `SimpleConnectionFactory` connection factory class is registered in the `MyAdapter` adapter implementation class:

```
package com.wm.MyAdapter;
..
..
..
import com.wm.MyAdapter.connections.SimpleConnectionFactory;
..
..
..
public class MyAdapter extends WmAdapter implements MyAdapterConstants {
..
..
..
    public void fillAdapterTypeInfo(AdapterTypeInfo info, Locale locale) {
        info.addConnectionFactory(SimpleConnectionFactory.class.getName());
    }
}
```

Connection Class Interactions

Integration Server uses connection classes at design time to support the creation and management of the connection namespace nodes (as well as the adapter service and notification namespace nodes). At run time, the ADK connection manager creates and releases connections as necessary, based on the pool configuration in the connection node and the demand for access to the adapter resource. This section describes the interactions of connection classes during the management of connection namespace nodes, as well as the basic flow used whenever connections are created or destroyed.

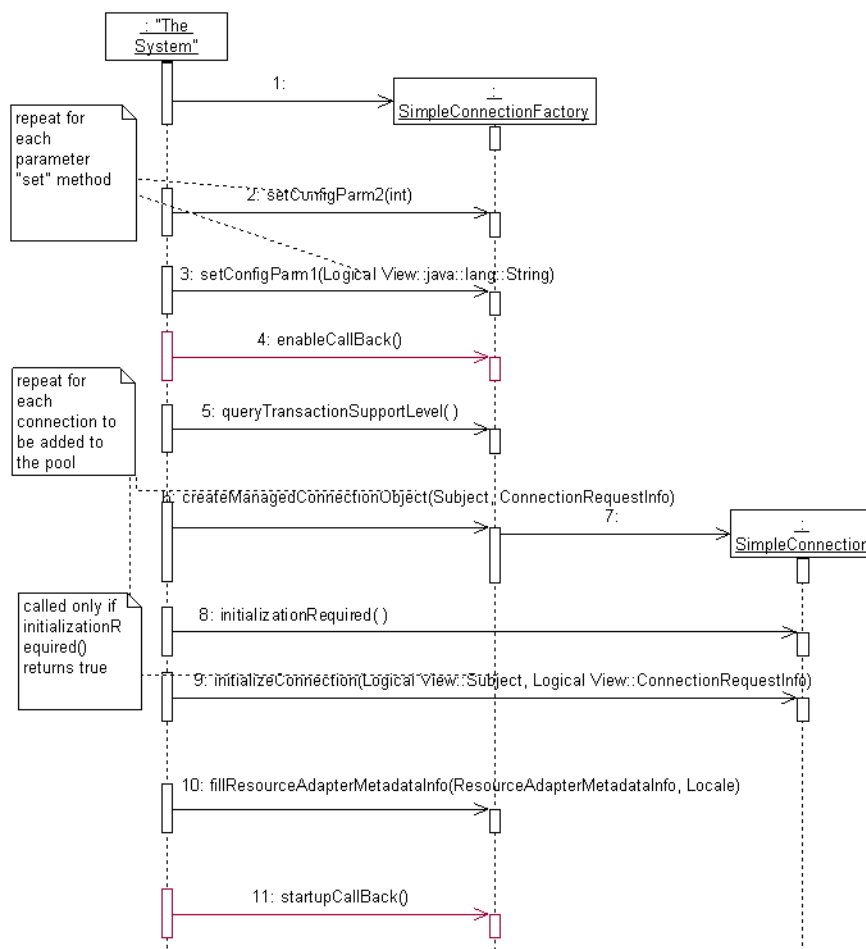
Retrieving Connection Metadata



When a user of the adapter creates or edits a connection node, Integration Server interrogates the connection factory implementation to retrieve the metadata as follows:

1. Gets the adapter name, version, and JCA version, as shown in steps 1 to 3 in the figure.
2. Instantiates the connection factory if it is not already cached, and introspects the connection factory for the metadata parameters, as shown in step 4 in the figure. Integration Server retrieves the *set* accessor methods.
3. Calls the *get* method for each metadata parameter (if present) and retrieves any default values that may be provided for those parameters, as shown in steps 5-6 in the figure. While *get* methods may have other uses in the implementation, this is their only use from the ADK's perspective.
4. Calls the *fillWmDescriptor* method of the connection factory, as shown in step 7 in the figure. The *fillWmDescriptor* method can specify the order in which the parameters must appear on the adapter's administrative interface, set display names, and constrain input of the users of the adapter.

Enabling Connection Nodes



Connection nodes are disabled by default; users must explicitly enable them using the adapter's administrative interface. If a connection node is enabled when Integration Server shuts down, it is also be enabled at Integration Server startup.

Integration Server performs the following actions to enable a connection node:

1. Obtains a connection factory instance.

If the connection manager has already created a connection factory instance (and it is cached for use with this node), that instance is used; otherwise, the manager instantiates a new instance, as shown in step 1 in the figure.

2. Updates the connection factory instance with the metadata parameter settings using the *set* methods.
3. Calls the *enableCallBack* method to call any adapter specific operations for the enable state change, as shown in step 4 in the figure.
4. Calls the *queryTransactionSupportLevel* method to get the transaction support capabilities of connections created by the connection factory instance, as shown in step 5 in the figure.

Transaction support depends on the capabilities of the adapter resource and the current metadata parameter settings. This method determines whether connections from this node can participate in a transaction that may involve other connections, adapters, or resources. For more information about transaction support, see [“Specifying Transaction Support in Connections” on page 341](#).

5. Initializes the connection pool.

For each connection it places in the pool (based on the minimum pool size specified when the node was created, Integration Server calls the `createManagedConnectionObject` on the connection factory and the initialization methods on the resulting connection object, as shown in steps 6 through 9 in the figure. If connection pooling is disabled, a single connection is created, initialized, and then destroyed.

6. Integration Server calls the `fillResourceAdapterMetadataInfo` method on the connection factory to register the types of adapter service templates supported by the connection.
7. Calls the `startupCallBack` method to perform any adapter specific operations for the startup state change, as shown in step 11 in the figure.

Creating Connections

The connection manager requests new connections from an adapter's connection factory when the node is enabled, and whenever there is demand for a connection and the pooling limits defined by the connection node have not been exceeded. If pooling for the node is disabled, the manager creates a connection for each request, and destroys it when the request is completed. The process of creating a connection is shown in steps 5-7 in the figure above.

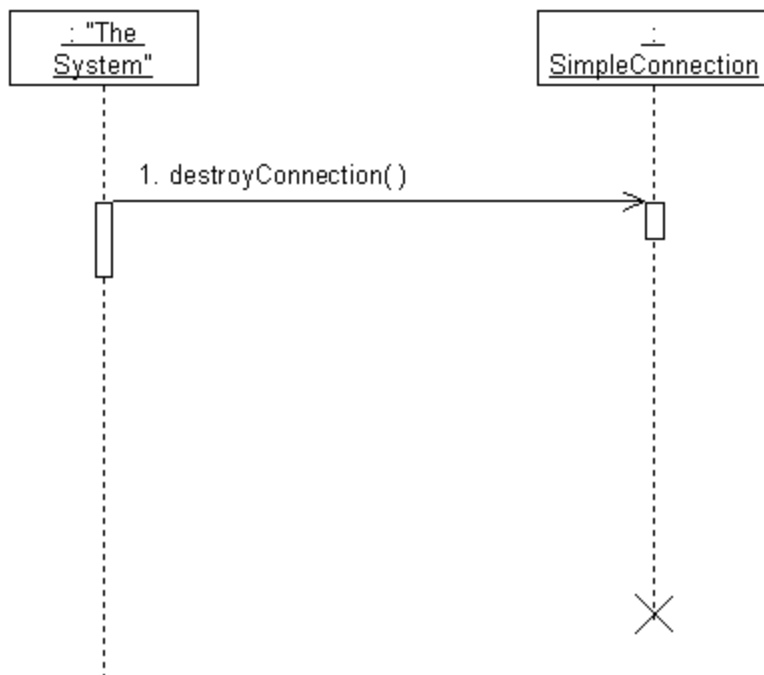
Disabling Connection Nodes

Disabling a connection node causes the connection manager to release all connections in the connection pool, and to reject any further requests for connections from that node. The connection manager destroys connections that are currently being used by an adapter service or notification when the current invocation of the adapter service or notification is completed. To see how a connection is released, see the figure in [“Releasing Connections” on page 71](#).

Note:

A listener instance holds the connection it retrieves during listener initialization for the lifetime of the listener instance. Disabling the connection node has no impact on this connection already held by the listener, but prevents the listener from starting or restarting in the event of an `AdapterConnectionException`. For more information, see [“Listener and Listener Notification Interactions” on page 156](#).

Releasing Connections



When a connection is not in use and is not needed to maintain a connection pool, the connection manager calls the connection implementation's `destroyConnect` method and removes all references to the object, allowing it to be garbage collected.

Receiving AdapterConnectionExceptions

When Integration Server receives an `AdapterConnectionException` thrown from an adapter, Integration Server resets the connection node associated with the exception. This means that the connections in the pool are destroyed and not recreated until the next connection request.

Configuring and Testing Connection Nodes

Now you are ready to configure a connection node and verify that it establishes a connection to your adapter resource as follows:

- Configure and enable a connection node.

Configuring Connection Nodes

You create, manage, and enable connection namespace nodes using the adapter's management screen. When you create a connection node, it is disabled by default. You must enable the connection node after you create it.

➤ **To configure a connection node**

1. Start Designer.
2. In **Adapters** screen, select the name of your adapter.

An adapter management screen opens, displaying any connection nodes that are currently configured for the adapter. There may be one connection type for each connection factory supported by the adapter.

3. Select **Connections** from the navigation area.
4. Select **Configure New Connection**.
5. In the **Connection Types** screen, select a connection type.
6. In the **Configure Connection Type** screen, provide values for the connection's parameters.
 - a. Complete the **Configure Connection Type > Adapter_Name** section as follows:

Field	Description
Package	Namespace node package in which to create the connection. For more information about creating packages, see “Package Management” on page 49 .
Folder Name	Folder name in which to create the connection. If the folder does not already exist in the package, the Integration Server creates it.
Connection Name	Connection name.

- b. Complete the **Connection Properties** section as appropriate for your adapter resource.

Enter values for the connection's metadata properties. For example, the Sample Adapter displays parameters such as **Sample Server Host**, **Sample Server Port Number**, **Local Transaction Control**, and **Timeout**.

- c. Complete the **Connection Management Properties** section as follows:

Field	Description
Enable Connection Pooling	Enables or disables the use of connection pooling for the connection. The default value is true (enabled).
Minimum Pool Size	Number of connections to create when the connection is enabled. The default value is 1.
Maximum Pool Size	Maximum number of connections that can exist at one time in the connection pool. The default value is 10.

Field	Description
Pool Increment Size	Number of connections by which the pool will be incremented if connections are needed, up to the maximum pool size. The default value is 1.
Block Timeout	<p>If connection pooling is enabled, this field specifies the number of milliseconds that Integration Server waits to obtain a connection with the adapter resource before it times out and returns an error.</p> <p>For example, you have a pool with Maximum Pool Size of 20. If you receive 30 simultaneous requests for a connection, 10 requests wait for a connection from the pool. If you set the Block Timeout to 5000, the 10 requests wait for a connection for 5 seconds before they time out and return an error. If the services using the connections require 10 seconds to complete and return connections to the pool, the pending requests fail and return an error message stating that no connections are available.</p> <p>If you set the Block Timeout value too high, you may encounter problems during error conditions. If a request contains errors that delay the response, other requests are not sent. This setting must be tuned in conjunction with the Maximum Pool Size to accommodate such bursts in processing. The default value is 1000.</p>
Expire Timeout	<p>If connection pooling is enabled, this field specifies the number of milliseconds that an inactive connection can remain in the pool before it is closed and removed from the pool.</p> <p>The connection pool removes inactive connections until the number of connections in the pool is equal to the Minimum Pool Size. The inactivity timer for a connection is reset when the connection is used by the adapter.</p> <p>If you set the Expire Timeout value too high, you may have a number of unused inactive connections in the pool. This consumes local memory and a connection on your backend resource. This could have an adverse effect if your resource has a limited number of connections.</p> <p>If you set the Expire Timeout value too low, performance could degrade because of the increased activity of creating and closing connections. This setting should be tuned in conjunction with the Minimum Pool Size to avoid excessive opening/closing of connections during normal processing.</p> <p>The default value is 1000. Enter -1 to specify no timeout.</p>
Startup Retry Count	Number of times that the system should attempt to initialize the connection pool at startup if the initial attempt fails. The default value is 0 (a single attempt).

Field	Description
Startup Backoff Timeout	Number of seconds that the system should wait between attempts to initialize the connection pool. This field is irrelevant if the value of Startup Retry Count is 0. The default value is 10.

7. Click **Test Connection**.

The connection is tested based on the settings provided.

8. Click **Save Connection**.

The connection name is now listed on the adapter's Connections screen and in the Service Browser of Designer.

9. In the adapter's **Connections** screen, enable the connection node by clicking **No** in the **Enabled** column. The **Enabled** column now shows **Yes**.

Integration Server initializes a connection pool based on the provided settings. Enabling and disabling a connection node produces entries in Integration Server log.

Note:

If a connection node is enabled when Integration Server shuts down, it is enabled at Integration Server startup.

5 Adapter Services

■ Overview	76
■ Adapter Service Classes	76
■ Metadata Model for Adapter Services	78
■ Adapter Service Template Interactions	98
■ Adapter Service Implementation	102
■ Configuring and Testing Adapter Service Nodes	119

Overview

An adapter service defines an operation that the adapter performs on an adapter resource. Adapter services operate like Integration Server flow services or Java services. Adapter services have input and output signatures, can be invoked within flow services, and can be audited from the Integration Server's audit system.

Like a connection, an adapter service consists of a Java class component and a namespace node in which design time settings are stored in the metadata parameters. Adapter services support:

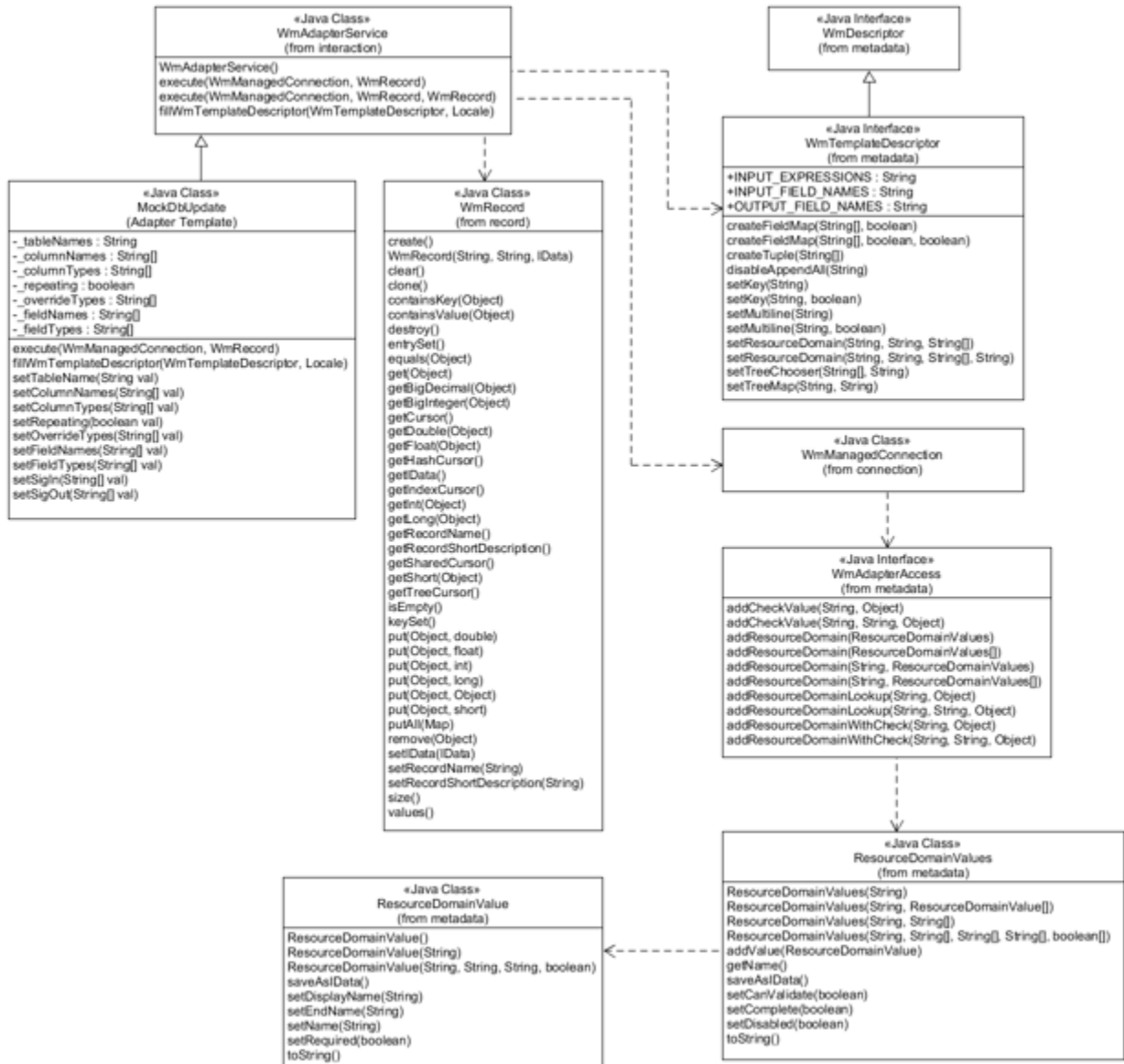
- Basic metadata constructs supported by connections.
- Additional data types.
- More sophisticated widgets.
- Ability to define the signature of the adapter service node.

This means that the users of the adapter can specify what data to search for in the flow service pipeline when the adapter service is called, and what data to place in the pipeline during the execution of the service. Using these signatures, you can link the adapter services to other Integration Server elements as part of a total integration solution.

Designer provides the facilities for the users of the adapter to create, configure, and test adapter service nodes.

Adapter Service Classes

The following figure shows the classes provided by the ADK to support adapter service templates. It also shows the `com.wm.adk.cci.interaction.WmAdapterService` implementation class *MockDbUpdate*.



Adapter Services Implementation Classes

Create an adapter service class by extending the `com.wm.adk.cci.interaction.WmAdapterService` base class. The adapters include several adapter service template classes, so it is common to place them in their own package. You must override the following base class methods in your `WmAdapterService` implementation class:

Method	Description
<code>execute</code>	Receives a <code>WmManagedConnection</code> implementation object from the adapter implementation, and a <code>WmRecord</code> containing the pipeline data.

Method	Description
	<p>Note: This is an abstract method in the base class, so failing to override it results in a compilation error.</p> <p>For more information, see “Adapter Service Execution” on page 96.</p>
fillWmTemplateDescriptor	<p>Receives a WmTemplateDescriptor object and a Locale object. Serves to modify how metadata parameters are handled during data entry.</p> <p>Note: Failing to override this method results in a runtime error.</p> <p>For more information, see “WmTemplateDescriptor Interface” on page 79.</p>
metadataVersion	<p>Returns the current version of the metadata.</p> <p>Note: If the template has multiple metadata versions, override this method.</p>
fieldsToIgnoreInMetadataDefinition	<p>Uses metadata version as input and returns an array of the fields that are not applicable to the metadata version provided.</p> <p>Note: If the template has multiple metadata versions, failing to override this method results in breaking the old services.</p>

Metadata Model for Adapter Services

The following sections describe the basics of the metadata model for adapter services.

Metadata Parameters for Adapter Services

Metadata parameters for adapter services use the same model that connections use.

- The restriction on sequence parameters (arrays) does not apply to adapter services. The adapter service editor supports widgets that allow the users of the adapter to view and manipulate array values in several ways, as described in the next section.
- Providing default values to parameters through a "get" method is not as valuable in the context of adapter services. This functionality is largely replaced by the use of resource domains. Some functionalities, such as the specification of the run time signature of the service, require values to be provided through the resource domain facilities. For more information, see [“Resource Domains” on page 84](#).

For more information, see [“Metadata Model for Connection” on page 63](#).

WmTemplateDescriptor Interface

The WmTemplateDescriptor interface extends the com.wm.adk.metadata.WmDescriptor interface. The WmTemplateDescriptor controls how metadata parameters appear, and defines rules for data entry. Include the method fillWmTemplateDescriptor in your adapter service to populate WmTemplateDescriptor.

Important:

Do not call the base class version of the method (by calling super()).

WmTemplateDescriptor introduces new methods and concepts. In addition, it also modifies the behavior of some of the methods inherited from WmDescriptor.

WmDescriptor Methods

Method	Description
createGroup	<p>Specifies the order in which parameters must appear in Designer. Unlike with connections, you can create multiple groups for adapter services. Each group corresponds to a tab in the adapter service editor, with the group name becoming the key. The key is used to identify the display name. Each group has a resource bundle entry that provides the display name shown on the screen. Group names are only displayed if you do not provide a display name in your resource bundle.</p> <p>Parameters can be assigned to only one group. If some or all of a service's metadata parameters are not assigned to a group, a default group is created and the unassigned parameters are added to it. This is true even if the parameter is hidden. Each group can have a display name and a description specified in the resource bundle.</p>
setValidValues	Use the resource domain facilities (recommended) instead of this method. For more information, see “Resource Domains” on page 84 .
setPassword	Displays asterisks when the users of the adapter enter passwords.
	<p>Note: The adapter service editor does not support the automated password confirmation facilities described for connections.</p>
setDescriptions	Searches the resource bundle for display names, and descriptions for the service, metadata parameters, and any groups that have been created at the point setDescriptions is called. If you want group display fields to be loaded from the resource bundle, call setDescriptions at the end of the fillWmTemplateDescriptor implementation.
setHidden	Designates properties as hidden. A hidden property does not appear in the adapter service editor.

Note:

Some of these methods have implicit order requirements because the methods build on the activities performed in previous method calls.

By default, the adapter service editor displays each metadata parameter in its own widget, based on the parameter's data type. These widgets include:

Widget	Data Type
Text box	String parameter
Table widget	Array
Check box	Boolean
A text box with scrollable values	Numeric

Use the following `WmTemplateDescriptor` methods to modify and enhance the default behavior.

WmTemplateDescriptor Methods

Method	Description
<code>createFieldMap</code>	Organizes parameters into columns of a single table widget.
<code>createTuple</code>	Grouping mechanism that you can use to modify the behavior of a resource domain lookup, and how resource domain values are applied in a field map.
<code>setMultiline</code>	Changes the widget from a standard text box to a multi-line text box. This widget also supports text import from files. The resulting parameter value may include embedded returns.
<code>setResourceDomain</code>	Associates a metadata parameter of an adapter service with a resource domain supported by the service's connection.

For more information about FieldMaps and Tuples, see [“FieldMaps” on page 81](#), [“Tuples” on page 83](#).

For more information about setting resource domains, see [“Resource Domains” on page 84](#) and [“Associating Metadata Parameters with Resource Domains” on page 87](#).

Order of WmTemplateDescriptor and WmDescriptor Methods Called

Some of these methods have implicit order requirements because they build on activities performed in the previous methods calls. For example, call `setDescription` after the final call to `createGroup` so that the resource bundle lookups can include all groups defined in the service. Use the calls in the following order:

1. `createGroup`
2. `createFieldMap`
3. `createTuple`
4. `setResourceDomain`

5. setDescription

Note:

If you create tuples, make `setResourceDomain` calls for tuple parameters in the order in which the parameters are set in the tuple.

Metadata Parameter Groups, FieldMaps and Tuples

Groups

Call the `WmTemplateDescriptor.createGroup` method to perform the following:

1. Organize parameters into different tabs.
2. Specify the order in which the parameters appear.

FieldMaps

Call the `WmTemplateDescriptor.createFieldMap` method to organize various sequence parameters (within the same group) into a single table widget in the adapter service editor. The full signature for `createFieldMap` is:

```
void createFieldMap(String[] members,
                   boolean variable,
                   boolean fillAll);
```

The following table describes the parameters of `WmTemplateDescriptor.createFieldMap`:

Parameter Name	Description
----------------	-------------

<i>members</i>	<p>Specifies a list of the parameter names that make up the field map. A field map can contain one or more member parameters, but a parameter must not be a member of more than one field map.</p> <ul style="list-style-type: none"> ■ The members argument is not an ordered list. ■ The members argument has no impact on the order in which the columns appear in a field map. ■ The order of the columns in a field map is dictated entirely by the order in which the parameters appear in the group. ■ The first member to appear in the group list appears in the first column of the field map. ■ The remaining columns follow the order in which they appear in the group. ■ If there is more than one field map in a group, then the relative positions of the first column parameters in the group dictates the order in which the field map tables appear.
----------------	---

Parameter Name	Description
	<p>Note: Using the parameters from different groups in a field map results in an exception.</p>
<i>variable</i>	<p>Enables the users of the adapter to add rows to the table. Possible values are:</p> <ul style="list-style-type: none"> ■ <code>true</code>. The values are not populated by default. However the users of the adapter must click on the add row icon to populate each value row in the table. ■ <code>false</code>. The values are populated by default, and the add row icon is disabled, thereby disabling the users of the adapter from adding rows to the table. <p>In this case, the field map is considered to be a variable field map because the number of fields that appear in the adapter service editor may vary.</p> <p>By default, each time the user of the adapter adds a row to the field map, each column is populated based on the associated parameter's data type and the contents of the associated resource domain. Typically, the column contains a dropdown list of string values from the resource domain. In other cases, either a check box appears (for Boolean parameters) or the column is empty.</p>
<i>fillAll</i>	<p>Populates the table with all the available data. Possible values are:</p> <ul style="list-style-type: none"> ■ <code>true</code>. Populates the table with all the available data. ■ <code>false</code>. The user of the adapter must add the rows in the table. ■ If the <i>fillAll</i> argument is <code>true</code> and the <i>variable</i> argument is <code>false</code>, then : <ul style="list-style-type: none"> ■ The table is expanded to contain one row for each value provided for the parameter in the first column of the field map. ■ The values for this first column are provided by the associated resource domain, and cannot be changed or manipulated by the user. This is true even when the associated resource domain's <code>setComplete</code> method is <code>false</code>; the users of the adapter cannot directly update this column. The users of the adapter can still make changes to other parameters that might impact the content of the resource domain of the first column's parameter. For more information about dependencies, see “Associating Metadata Parameters with Resource Domains” on page 87. ■ The remaining columns contain the same value and user interface widget that would be employed if the user manually inserted the row.
	<p>Note:</p> <ul style="list-style-type: none"> ■ If <i>variable</i> is <code>false</code>, then <i>fillAll</i> is assumed to be <code>true</code>, regardless of the value passed in the argument. ■ Do not set both <i>fillAll</i> and <i>variable</i> to <code>true</code>; the resulting behavior is unpredictable.

Tuples

The `WmTemplateDescriptor.createTuple` method is another grouping mechanism that you can use to modify:

- The behavior of a resource domain lookup.
- How the resource domain values are applied to parameters in the tuple.

Members of a tuple are linked when:

- The resource domain values are retrieved.
- The values are updated on the user interface.

When the adapter service editor performs a resource domain lookup for a parameter in a tuple, it expects the response array to contain a `ResourceDomainValues` object for each parameter in the tuple. Thus, changes resulting from the resource domain lookup is applied simultaneously to each parameter in the tuple. This mechanism is particularly useful when two or more sequence parameters in a field map are closely related. For example, when one parameter contains a column name and the other parameter contains the column format.

Requirements for reliable tuple operation are as follows:

- The parameters of a tuple must be sequence parameters of the same field map. If parameters are in separate field maps, the resource domain lookup functions properly, but the user interface characteristics do not function as described below.

In the user interface, the first parameter in a tuple serves as the *primary* parameter, and all other parameters are *secondary* parameters. In the adapter service editor:

- Users of the adapter can directly manipulate the primary parameter, but not secondary parameters.
- The secondary parameter contains the value from its resource domain that corresponds to the value selected from the primary parameter. For example, if the fourth member of the primary parameter's resource domain is selected, then the fourth member of the secondary parameter's resource domain appears in the secondary parameter's column.
- If a secondary parameter value is not specified in the position corresponding to the primary parameter's value, then the secondary parameter is left blank in that row.
- A tuple must be declared (in the code) before a `setResourceDomain` method.
- The first parameter of a tuple must be assigned to a resource domain before any other member of the tuple.
- Each parameter of a tuple must have the same parameter dependencies listed in the `setResourceDomain` call that you used to assign the metadata parameters of the adapter service to a resource domain.
- For each `ResourceDomainValues` object returned in the lookup, the `setComplete` method must be `true` in which the users of the adapter cannot supply parameter values.

- The first parameter in the tuple must appear first in its group.

For more information, see [“Associating Metadata Parameters with Resource Domains”](#) on page 87, and [“Populating Resource Domains with Values”](#) on page 88.

Resource Domains

A *resource domain* defines the domain of valid values for metadata parameters, based on rules and/or data that are specific to the adapter resource. Resource domains can have properties that affect the behavior of the resource domain, and associations with the metadata parameters.

You can use resource domain values to:

- Assign default values for parameters.
- Enable the adapter to look up parameter values in the adapter resource.
- Enable the users of the adapter to supply their own parameter values, and enable the adapter to validate these values.
- Disable parameters, based on specified sets of values in other parameters.

A common use of resource domain values is to create a dropdown list of data values for a parameter, much like the `WmDescriptor.setValidValues` method in connection factories. However, the resource domain values differ from the `setValidValues` lists in two important ways:

- Resource domain values can interact with the adapter service editor.

As values in one parameter change, callbacks are made to the adapter to update the resource domain values. For example, if parameter A contains a list of database table names, and parameter B contains a list of column names, then when a table is selected in parameter A, the resource domain values used in parameter B can be updated to reflect the columns from the table selected in parameter A.

- Resource domain values can be retrieved directly from the adapter resource, using a `WmManagedConnection` instance.

To create a resource domain:

1. Register the resource domain and its properties in your `WmManagedConnection` implementation.
2. Associate the adapter's metadata parameters with the resource domain.
3. Populate the resource domain with values in your `WmManagedConnection` implementation.

Registering Resource Domains

You must register resource domains in your `WmManagedConnection` implementation using the `WmManagedConnection.registerResourceDomain` method. This method has a single argument of the type `WmAdapterAccess`. For more information, see [“Creating a WmManagedConnection Implementation Class”](#) on page 61.

Registering a resource domain name establishes the definition of the resource domain within a given scope. That is, you can register one resource domain to be used by all adapter services that use the connection, or you can register multiple resource domains on a service-by-service basis. You must register the name of each resource domain supported by the connection (and by extension, each resource domain used by any service supported by the connection).

In the `WmManagedConnection.registerResourceDomain` method, you must perform the following:

- Create a `ResourceDomainValues` object, which represents a list of values for a resource domain.
- Specify whether the resource domain is fixed or dynamic.

Method	Description
<code>WmAdapterAccess.addResourceDomain</code>	<p>Defines a fixed resource domain with one or more values. A <i>fixed</i> resource domain displays default values that you provide for the resource domain parameters. This method expects one or more <code>ResourceDomainValues</code> objects. For more information about resource domain values and their settings, see “Populating Resource Domains with Values” on page 88.</p>
<code>WmAdapterAccess.addResourceDomainLookup</code>	<p>Defines a dynamic resource domain. A <i>dynamic</i> resource domain enables the adapter to look up values for the parameters, based on changes to dependency parameters.</p> <p>This method supplies a reference to an object that the adapter service editor uses to make callbacks when the adapter service node is configured. Resource domain lookups can only be performed against <code>WmManagedConnection</code> objects, so the "this" reference is generally used as the object reference argument. For example:</p> <pre>access.addResourceDomainLookup("aSampleResourceDomainName", this);</pre> <p>For more information, see “Resource Domain Lookups” on page 89.</p>

- Specify whether the users of the adapter can provide their own parameter values, and enable the adapter to validate these values. To do this, you use the following methods:

Method	Description
<code>ResourceDomainValues.setComplete(false)</code>	<ul style="list-style-type: none"> ■ Allows the users of the adapter to supply values. ■ Sets a flag named <i>complete</i>.

Method	Description
	When using both fixed resource domains and dynamic resource domains, you can allow the users of the adapter to enter their own values, allowing them to add to the current list of values for the resource domain.
<code>ResourceDomainValues.setCanValidate(true)</code>	<ul style="list-style-type: none"> Enables the adapter to validate the user supplied values using <i>adapter check values</i> callbacks. The <code>WmManagedConnection.adapterCheckValue</code> method validates the user supplied values. Sets the <i>canValidate</i> flag
<code>WmAdapterAccess.addCheckValue</code>	Calls the <code>WmManagedConnection.adapterCheckValue</code> method.

Note:

`WmAdapterAccess.addCheckValue` must appear after `WmAdapterAccess.addResourceDomain` or `WmAdapterAccess.addResourceDomainLookup`.

The following example registers two resource domains:

```

101. public void registerResourceDomain(WmAdapterAccess access)
102.     throws AdapterException
103. {
104.     ResourceDomainValues tableRdvs = new ResourceDomainValues(
105.         MockDbUpdate.TABLES_RD, mockTableNames);
106.     tableRdvs.setComplete(true);
107.     access.addResourceDomain(tableRdvs);
108.
109.     access.addResourceDomainLookup(MockDbUpdate.COLUMN_NAMES_RD, this);
110.     access.addResourceDomainLookup(MockDbUpdate.COLUMN_TYPES_RD, this);
111.
112.     ResourceDomainValues rdvs = new ResourceDomainValues(
113.         MockDbUpdate.OVERRIDE_TYPES_RD, new String[] {""});
114.     rdvs.setComplete(false);
115.     rdvs.setCanValidate(true);
116.     access.addResourceDomain(rdvs);
117.     access.addCheckValue(MockDbUpdate.OVERRIDE_TYPES_RD, this);
118.     ...
121. }
```

In this example, note that:

- Lines 104-105 create a `ResourceDomainValues` object (identified by the constant `MockDbUpdate.TABLES_RD`).
- Line 106 indicates that the users of the adapter may not supply their own values to the resource domain.
- Line 107 adds a resource domain to this object.

- Lines 109-110 add lookups for this object, indicating that the resource domain is *dynamic*. For these lookups, you must pass an instance of the connection that can satisfy the lookup. This is accomplished by using the "this" reference.
- Lines 112-113 create an empty `ResourceDomainValues` object.
- Line 114-115 indicate that the users of the adapter may supply their own values, and the adapter validates these values. `setCanValidate(true)` calls the `adapterCheckValue` method (line 117) for each value that is not already in the resource domain.
- Line 116 adds a resource domain to this object. This is a fixed resource domain because no lookups are performed.

Associating Metadata Parameters with Resource Domains

The metadata parameters of an adapter service must be assigned to a resource domain supported by the service's connection. The interface `WmTemplateDescriptor`, provides `setResourceDomain` with the following signature:

```
void setResourceDomain(String name,
                      String resourceDomainName,
                      String[] dependencies)
```

Parameter Name	Description
<i>name</i>	Name of the parameter being assigned.
<i>resourceDomainName</i>	Resource domain name that matches the <i>name</i> registered in the connection.
<i>dependencies</i>	<p>List of any other metadata parameter names in the current adapter service upon which the value of the parameter in the first argument depends.</p> <p>Dependencies are important to dynamic resource domain lookups. When the user of an adapter changes the value of a parameter in the dependency list, a lookup retrieves a new set of resource domain values.</p> <p>For example, for the parameters named <i>columns</i> and <i>tables</i>, you might assign the <i>columns</i> parameter to a resource domain called <i>columnsLookup</i>, with a dependency on the <i>tables</i> parameter as follows:</p> <pre>d.setResourceDomain("columns", "columnsLookup", new String[] {"tables"});</pre> <p>When the <i>tables</i> parameter changes, <code>WmManagedConnection.adapterResourceDomainLookup</code> determines the new value or values that can be applied to the <i>columns</i> parameter. Depending on the properties of a resource domain, the lookup may be used to set the value of a parameter, or to provide a list of possibilities from which the user of the adapter may select a value.</p>

For more information, see [“Resource Domain Lookups” on page 89](#).

For more information about the variant forms of `setResourceDomain`, including the concept of *useColumns*, see [“The useParam Argument of setResourceDomain” on page 171](#).

Populating Resource Domains with Values

To populate a fixed or dynamic resource domain with values, implement the `ResourceDomainValues` class in your `WmManagedConnection` implementation.

This class is the primary container for controlling the behavior of the adapter service parameters. It is used during the registration process when you register a fixed resource domain, and it is returned from the `adapterResourceDomainLookup` method in response to a dynamic callback. For more information, see [“Resource Domain Lookups” on page 89](#).

A `ResourceDomainValues` object contains the following:

- Name of the resource domain.
- Current list of values for the resource domain.

The current list of values can contain an array of strings or an array of `ResourceDomainValue` objects. This array constitutes the list of values that appears in the appropriate widgets in the adapter service editor. The displayed values are communicated as strings, regardless of the data type of the parameter associated with the resource domain.

Note:

Using a list of values that cannot be converted to the parameter's data type results in a runtime error. Null and empty strings are not accepted in numeric parameters.

- The following methods:

Method	Description
<code>ResourceDomainValues.setComplete(false)</code>	<ul style="list-style-type: none"> ■ Allows the users of the adapter to supply values. ■ Sets a flag named <i>complete</i>. <p>When using both fixed resource domains and dynamic resource domains, you can allow the users of the adapter to enter their own values, allowing them to add to the current list of values for the resource domain.</p>
<code>ResourceDomainValues.setCanValidate(true)</code>	<ul style="list-style-type: none"> ■ Enables the adapter to validate the user supplied values using <i>adapter check values</i> callbacks. The <code>WmManagedConnection.adapterCheckValue</code> method validates the user supplied values. ■ Sets the <i>canValidate</i> flag
<code>ResourceDomainValues.setDisabled</code>	Disables the parameter in the adapter service editor.

For example, assume that a parameter named *portNumber* has a data type of `int`. When constructing the associated resource domain values, limit the list of values to numeric strings as follows:

```
new ResourceDomainValues("portNumberLookup", new String[]
{"6048", "8088", "9090"});
```

This example shows a `ResourceDomainValues` object with a set list. You have the option to define a range of values, by providing a minimum/maximum value. In this case, a single `ResourceDomainValues` object is used to define the minimum and maximum values of a numeric parameter. The following conditions must be met for defining a range of values:

- The parameter must be of a numeric data type (such as `int` or `long`).
- The parameter cannot be a sequence parameter (array).
- The `ResourceDomainValues` object must contain exactly one value that is constructed from a `ResourceDomainValue` object.
- `ResourceDomainValues.setComplete` must be `false`. You must set this method explicitly if you used the `ResourceDomainValue[]` constructor; otherwise, changes to the parameter are not saved.
- The embedded `ResourceDomainValue` object must have a `name` representing a numeric value, and an `endName` representing a number value greater than the value of the name.

Note:

This is the only defined use for `ResourceDomainValue.endName`. In all other cases, only `ResourceDomainValue.name` is used. All other `ResourceDomainValue` attributes are placeholders, and are not currently implemented. Using a `String[]` to construct `ResourceDomainValues` is equivalent to using a `ResourceDomainValue[]` where only the `ResourceDomainValue.name` attributes are populated.

Resource Domain Lookups

A dynamic resource domain uses resource domain lookups. The method `addResourceDomainLookup` is used to enable the adapter to look up parameter values in the adapter resource.

When the users of the adapter create an adapter service node, they select a connection in which the adapter service executes. That connection also provides data from the resource domain that you registered with the connection.

When a node is created, the adapter service editor initiates a resource domain lookup to the connection class for each of the adapter's metadata parameters that are associated with dynamic resource domains. Additional lookups are made as the users of the adapter modify the values of parameters upon which other parameters depend.

All resource domain lookups invoke the method `adapterResourceDomainLookup` in the `WmManagedConnection` implementation class:

```
ResourceDomainValues[] adapterResourceDomainLookup(String serviceName,
String resourceDomainName,
String[][] values)
```

Parameter Name	Description
<i>serviceName</i>	Class name of the adapter service.
<i>resourceDomainName</i>	Registered name of the resource domain.
<i>values</i>	A multi-dimensional array that is populated with the current value of the parameters upon which the current lookup depends, as specified in the <code>WmTemplateDescriptor.setResourceDomain</code> call.

For more information, see [“Associating Metadata Parameters with Resource Domains”](#) on page 87.

For example:

```
d.setResourceDomain( "columnsArg", "columnsLookup", new String[] {"tablesArg"});
```

This call creates a dependency on *tablesArg*. When the lookup for the *columnsLookup* resource domain is made, the *values* argument contains the current settings for the *tablesArg* parameter. Data in the *values* argument is organized such that:

- The first dimension of the array determines the dependent parameter.
- The second dimension iterates the data in the parameter.

Thus, `values[0][0]` contains the value of the first dependent parameter. If it is a sequence parameter, then `values[0][1]` would contain the next value in the sequence, `values[0][2]` the next, and so on. If there were more than one dependency, the contents of the second parameter on which the lookup depends would be contained in `values[1][0]`.

Note:

Placing a sequence parameter in a field map has several effects on the resource domain lookup process (see [“Tuples”](#) on page 83).

The signature of `adapterResourceDomainLookup` indicates that an array of `ResourceDomainValues` objects must be returned. Unless your implementation includes tuples, there must be exactly one object in the response array, and the name attribute of that `ResourceDomainValues` object must always be the same as the `resourceDomainName` argument passed into the method.

When the adapter service editor receives the lookup response data, it is evaluated against any data in the "current" parameter (the parameter for which the lookup was performed). An entry in the current parameter is considered valid if any one of the following is true:

- The parameter value matches a value in the `ResourceDomainValues` list.
- `ResourceDomainValues.setComplete(false)` and `ResourceDomainValues.setCanValidate(false)`.
- `ResourceDomainValues.setComplete(false)` and `ResourceDomainValues.setCanValidate(true)` and the value was successfully validated through the *adapter check values* facility.

If the current parameter settings are considered valid, then the parameter values remain unchanged. However, any current parameter values are merged with the values in the lookup response when the parameter's dropdown list is opened. (This effectively extends the resource domain to include

values that were entered by the user.) If the current parameter settings are not considered valid, then the invalid value is deleted and replaced by a value in the resource domain.

Adapter Check Value Callbacks

When using both fixed resource domains and dynamic resource domains, you may allow the users of the adapter to enter their own parameter values, allowing them to add to the current list of values for the resource domain. To enable the adapter to validate these values, you use callbacks known as *adapter check values*. Adapter check values are resource domain mechanisms that function very much like resource domain lookups. Any value in a parameter that is not part of the parameter's resource domain list may be validated by an adapter check value. For more information, see [“Registering Resource Domains” on page 84](#).

Adapter check values can also validate resource domain lookups for each unique value that is not part of a resource domain list that is complete (that is, a list that does not accept the values supplied by the users of the adapter). For example, suppose the sequence parameter named colors contains the values "White", "Gray", "Black", and "Red". After a resource domain lookup, the resource domain list contains "Black" and "Gray". Assuming that the resource domain is configured appropriately, the adapter service editor performs an adapter check value callback for "Red" and "White". If it finds either value, it deletes the cell containing that value or overwrites it, if the sequence is in a field map.

To use an adapter check value callback, you must:

- Set the following methods of the ResourceDomainValues class as follows:

Method	Description
ResourceDomainValues.setComplete(false)	<ul style="list-style-type: none"> ■ Allows the users of the adapter to supply values. ■ Sets a flag named <i>complete</i>. <p>When using both fixed resource domains and dynamic resource domains, you can allow the users of the adapter to enter their own values, allowing them to add to the current list of values for the resource domain.</p>
ResourceDomainValues.setCanValidate(true)	<ul style="list-style-type: none"> ■ Enables the adapter to validate the user supplied values using <i>adapter check values</i> callbacks. The WmManagedConnection.adapterCheckValue method validates the user supplied values. ■ Sets the <i>canValidate</i> flag
ResourceDomainValues.setDisabled	Disables the parameter in the adapter service editor.

If both of these flags are set properly, the adapter service editor calls the following method for each value that is not already in the resource domain:

```
Boolean adapterCheckValue(String serviceName,
                          String resourceDomainName,
```

```
String[][] values,
String testValue)
```

Parameter Name	Description
<i>serviceName</i>	Class name of the adapter service.
<i>resourceDomainName</i>	Registered name of the resource domain.
<i>values</i>	A multi-dimensional array that is populated with the current value of the parameters upon which the current lookup depends, as specified in the <code>WmTemplateDescriptor.setResourceDomain</code> call.
<i>testValue</i>	Value being checked.

- Register the adapter check value callback at the same time you register the resource domain name, using the `WmAdapterAccess.addCheckValue` method. For more information, see [“Registering Resource Domains” on page 84](#).

Field Maps with Resource Domain Dependencies

The `addResourceDomainLookup` and `adapterCheckValue` methods have a `values[][]` argument that contains the current value(s) of the parameter(s) on which the resource domain association depends. If the dependency parameter is a sequence parameter, the complete list of values is provided in the `values` argument. However, when a parameter depends on another parameter in the same field map, then by default each row in the field map is handled separately, for resource domain lookup/check value purposes.

For example, assume that with the sequence parameters A and B, the lookup for B depends on A. If the parameters are not in the same field map, then whenever a row in A changes, a lookup is performed for B, to which all values of A are passed. The results list is applied to each row of B, and updates are made to any rows of B that are no longer valid. The rows may be no longer valid because their values are not members of a "complete" resource domain (that is, a resource domain that does not allow the users of the adapter to enter values), or because an adapter check value callback failed to validate the rows.

However, if A and B are in the same field map, then when the value in a row of A changes, only the new value in that row is passed to the resource domain lookup, and the values returned from the lookup apply only to that row of B. For example, if parameter A contains catalog item numbers, and parameter B contains the colors in which the item is available, then for each catalog item number in A, there would be a separate dropdown list of available colors in column B. Also note that if the same catalog item number appeared in multiple rows in column A, then the dropdown list of colors in column B would be the same. There would not be a separate resource domain lookup for each of those rows because the adapter service editor would recognize that it already has the list of colors for that item number. Or, more accurately, that it already has a list of resource domain values based on the given set of dependency parameter values.

If you want to suppress the behavior described above, prefix the name of the parameter in the `setResourceDomain` dependency list with an asterisk (*). For example, the following method causes

Integration Server to treat parameter A like any other parameter, even if it were in a field map with B:

```
setResourceDomain("B", "bLookup", new String[] {"*A"})
```

Adapter Service Node Signatures

In addition to using the metadata model to create parameters for configuring an adapter service node, you can use the model to define the signatures of that node.

An adapter service node has an *input signature* and an *output signature*. 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 in the Input/Output tab of the adapter service editor in Designer. Rules for defining signatures appear in the following procedure.

Once the signature is complete, users may include an adapter service node in flow constructs. They can route, map, and transform the input and output of the adapter service as needed in the integration solution.

The following procedure provides a basic model that you can use to implement a metadata signature. If you deviate from this model, it is important to understand that signature resource domains are only invoked as a result of a value applied to a dependent parameter. Having a resource domain lookup simply change the list of possible values in the resource domain does not impact the signature unless the current value is changed.

➤ To create the signature of an adapter service node

1. Create metadata input parameters for the field names, data types, and signature. Each parameter must have a data type of `String[]`.

For example, assume that *inputNames*, *inputTypes*, and *inputSignature* are created as follows:

```
public void setInputNames(String[] val);
public void setInputTypes(String[] val);
public void setInputSignature(String[] val);
```

2. Add these parameters to a group in the same order as above.

For example:

```
templateDescriptor.addGroup("group name", new String []
{ ..., "inputNames", "inputTypes", "inputSignature"});
```

3. You may hide any or all of the parameters. (Hiding all parameters in a field map hides the map table as well.)

For example:

```
templateDescriptor.setHidden("inputNames");
templateDescriptor.setHidden("inputTypes");
```

```
templateDescriptor.setHidden("inputSignature");
```

4. Create a field map containing the three input parameters. In the `createFieldMap` method, set the `variable` argument to `false`, and set the `fillAll` argument to `true`.

For example:

```
templateDescriptor.createFieldMap(new String [] {"inputNames",  
"inputTypes", "inputSignature"}, false, true);
```

In some cases, you can include other parameters in this field map, but this can sometimes be problematic, particularly if these parameters are hidden.

5. Create a tuple containing the names and types parameters.

For example:

```
templateDescriptor.createTuple(new String [] {"inputNames",  
"inputTypes"});
```

6. In the associated connection class(es), register two resource domains to support name and type lookups.

For example, assume that the resource domains *inputNamesLookup*, and *inputTypesLookup* are created as follows.

```
access.addResourceDomainLookup("inputNamesLookup", this);  
access.addResourceDomainLookup("inputTypesLookup", this);
```

7. Assign the names parameter to the name lookup resource domain, and assign the types parameter to the type lookup resource domain. These assignments must specify the same dependencies because the parameters are in a tuple. If no dependencies are known at this time, specify `null`.

For example:

```
templateDescriptor.setResourceDomain("inputNames","inputNamesLookup",null);  
templateDescriptor.setResourceDomain("inputTypes","inputTypesLookup",null);
```

8. Assign the signature parameter to one of the reserved resource domain names provided in `WmTemplateDescriptor`, specifying the names parameter and the types parameter as dependencies.

For example, *INPUT_FIELD_NAMES* would be used as follows:

```
templateDescriptor.setResourceDomain( "inputSignature",  
WmTemplateDescriptor.INPUT_FIELD_NAMES, new String[]  
{ "inputNames", "inputTypes"}););
```

9. Implement the name and type lookups as described in [“Implementing Resource Domain Lookups for Signature Names and Data Types”](#) on page 95.

10. Create an output signature by repeating this procedure, substituting *OUTPUT_FIELD_NAMES* for *INPUT_FIELD_NAMES* in step 8.

Implementing Resource Domain Lookups for Signature Names and Data Types

You can load a signature's name and data type parameters using resource domain lookups. You implement lookups by including an `adapterResourceDomainLookup` method in your `WmManagedConnection` implementation class.

These parameters are implemented as string arrays, with the corresponding index in each array used to associate a name with a data type. The following subsections describe the values you supply in your resource domains for the name and data type parameters.

Field Name String Values

A field name string can contain a simple value, such as *itemNumber*, or a more complex value such as *customer.orders[].lineItems[].itemNumber*. The complex field name string demonstrates the ability of a signature to specify hierarchy (using a dot ".") and multiplicity (using a pair a square brackets "[]"). Thus, this example shows an aggregate of customer fields containing multiple orders that may contain multiple line items that contain one *itemNumber*.

Follow these rules when creating resource domain values for containing signature names:

- A name can be used as a field (containing data) or an aggregate (containing fields), but not both.
- Each entry must contain a field with any containing aggregates. Do not specify aggregates alone.
- Both fields and aggregates can be arrays, indicated by square brackets. For example: *customer.orders[].lineItems[].itemNumber*.
- Use name restrictions.

For more information, see the *webMethods Service Development Help* for your release.

Data Type String Values

A data type string must contain the data type corresponding to the field name string at the same index in the field name's resource domain list of values. Data types for the adapter services are similar to the data types for Java services. If the signature item is made accessible from an Integration Server flow, its data type must be `java.lang.String`, `java.util.Date`, or one of the "big-letter-primitive" classes (e.g., `java.lang.Integer`). If the data is made to not be accessible from a flow, then any class type is acceptable.

Multiplicity in the data type string uses a pair of square brackets [] appended to the class name. Data type multiplicity represents the multiplicity across the entire signature hierarchy by adding a set of brackets for each set of brackets in the corresponding name string.

Even though there is only one item number in the *lineItems* aggregate, there are many in the signature. In this case, the data type would be `java.lang.Integer[][]` if *itemNumber* is an integer.

For examples, see [“Example 1: WmAdapterService Implementation Class”](#) on page 103. For more information, see *webMethods Service Development Help* for your release and [“Interacting with the Pipeline”](#) on page 96.

Adapter Service Execution

When a flow service, or a trigger invokes an adapter service, Designer invokes an adapter service node. The adapter service node calls the `WmAdapterService.execute` method. The signature is as follows:

```
public WmRecord execute(WmManagedConnection connection,
    WmRecord input) throws ResourceException
```

Input Parameter	Description
<i>connection</i>	The <code>WmManagedConnection</code> object argument delivers a connection instance from the adapter service's connection node. How the adapter uses this connection object to gain access to the adapter resource is determined by the adapter's design, not by the ADK.
<i>input</i>	The inbound <code>WmRecord</code> object contains data based on the input signature as well as any other information that may be in the flow service pipeline at the time the adapter service is invoked. The <code>execute</code> method is responsible for interrogating the inbound <code>WmRecord</code> object to retrieve the data necessary for the adapter service to perform its function. The validation that the fields specified in the signature are actually present in the pipeline is not performed by Integration Server, but the data type for any field present in the pipeline is guaranteed to conform to the data type specified in the signature. The adapter service must determine whether all the required data is present, and how to respond when data elements are missing.

Output Parameter	Description
WmRecord object	When the <code>execute</code> method completes, the outbound <code>WmRecord</code> object must contain data based on the output signature defined in the metadata. This data is added to the pipeline. (An adapter service may not remove the data from the pipeline because it only works with a copy of the pipeline object, not the original pipeline object.) Once again, the adapter implementation has primary responsibility for determining what portions of the signature is populated. A run time exception might occur if a field that was not populated by the adapter service is mapped within a flow service.

Interacting with the Pipeline

During run time, an adapter service has to perform the following:

- Retrieve data from the pipeline at the beginning of its execution.

To retrieve data from the pipeline, the service must interrogate the `WmRecord` argument of the `execute` method. Limit the interrogation to fields identified in the service's signature because often there is other information in the pipeline that is not intended for the service.

- Add data to the pipeline at the end of execution.

At the end of execution, the `execute` method must return a `WmRecord` instance containing the data that must be added to the pipeline. Organize the return data in a way that is consistent with the metadata signature so that other adapter services (or flow or Java services) can access it.

For discussion purposes, assume the following as a sample metadata signature for both input and output of an adapter service:

Field Name	Type
<i>customer.id</i>	<code>java.lang.Integer</code>
<i>customer.name</i>	<code>java.lang.String</code>
<i>customer.orders[].id</i>	<code>java.lang.Integer[]</code>
<i>customer.orders[].date</i>	<code>java.util.Date[]</code>
<i>customer.orders[].lineItems[].itemNumber</i>	<code>java.lang.Integer[][]</code>
<i>customer.orders[].lineItems[].quantity</i>	<code>java.lang.Integer[][]</code>
<i>customer.orders[].lineItems[].description</i>	<code>java.lang.String [][]</code>

The sample signature describes a hierarchal structure that can be expressed as a tree structure, where the actual field names form the leaves, and the elements preceding the field name are nodes. Thus, the names *customer*, *orders*, and *lineItems* are node names, and *id*, *name*, *date*, *itemNumber*, *quantity*, and *description* are leaves in the tree structure.

The `WmRecord` class, which is the primary carrier of data into and out of adapter services, is a JCA based wrapper for an `IData` object. It provides methods that access the `IData` content. However, when dealing with a hierarchal structure (as is the case with the sample), it is necessary to "drill down" into the `IData` structure. Therefore, ignore the `WmRecord` methods except for `getData`, and `putData`, which is used to access the underlying `IData` object.

Note:

The `IData` interface is part of the standard webMethods Integration Server Java API. Its structure is based on key/value pairs, where the key is a `String` and the value is a Java object. For more information, see the Javadoc entries for `IData`, `IDataCursor`, `IDataFactory`, and `IDataUtil`.

getData Method

Returns an `IData` object that contains the entire pipeline at the time the service was invoked. The top-level branch or leaf name(s) in the metadata signature (in this case, *customer*) is the key used to access data intended for use by the service. The value associated with that key is either another

IData object (if the key is a node name) or an object of the type specified by the corresponding type field of the signature. If the name of the branch or leaf includes a pair of square braces "[]", then the value contains an array of the designated object type.

Thus, the fields in the sample would be populated as follows:

- Returns an IData object with an entry, keyed with the name *customer*.
- The value associated with *customer* is another IData object, with three entries: *id*, *name*, and *orders*.
- The value corresponding with *id* is an Integer.
- The value of *name* is a String.
- The value of *orders* would contain an array of IData objects.
- The *orders* IData objects would each contain an *id* of type Integer, a *date* of type `java.util.Date`, and an array of IData objects associated with the *lineItems* key.
- The *lineItems* IData objects would contain entries for *itemNumber*, *quantity*, and *description*, with the data types provided in the signature.

When constructing response data to place in the pipeline at the end of service execution, use the same rules that apply to interpreting the metadata signature. For each node level in the signature, there must be a corresponding layer of IData, keyed with the names from the signature. For each leaf, there must be an IData entry with the corresponding signature name and type.

Note:

Signatures are not enforced by Integration Server or the ADK framework. The validity of a request based on the presence or absence of a given field, or the value given to a field, is determined exclusively by the adapter implementation. Similarly, if the service fails to populate the response `WmRecord` with data organized according to the signature, subsequent services cannot access the data provided by the adapter service.

Adapter Service Template Interactions

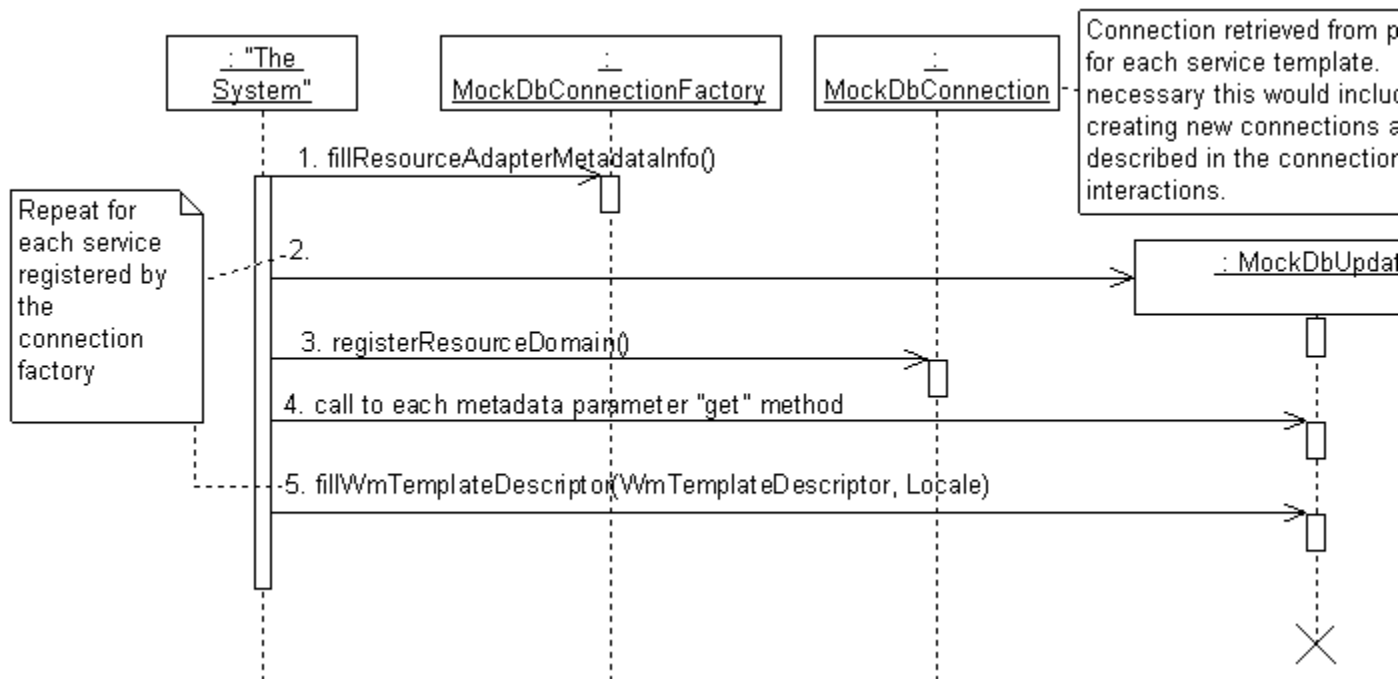
Creating Adapter Service Nodes

Designer provides a wizard to guide the user of the adapter through the process of creating adapter service nodes. During this process there are two significant interactions with the adapter. They occur:

- When the user of the adapter selects the connection node to be used by the service node.
- After the user of the adapter enters the name and folder of the new adapter service node.

Selecting Connection Nodes

The following diagram shows the adapter calls made when the users of the adapter select a connection node and how the metadata cache is loaded in Designer.



When the user of the adapter selects the connection node to be used by the service node, the Integration Server calls the `fillResourceAdapterMetadataInfo` method in the connection for the supported adapter service template class names. Integration Server performs the following:

- Instantiates each of these template classes.
- Retrieves the default metadata parameter values (by calling the parameter "get" methods).
- Calls its `fillWmTemplateDescriptor` method.

All this information is cached in Designer session, and is not requested again for any adapter service activity associated with that connection node. That is, after this information has been gathered from a connection node, the user of the adapter may create multiple adapter service nodes based on any template associated with that connection using the same set of cached information. This is particularly significant during development of metadata-related code.

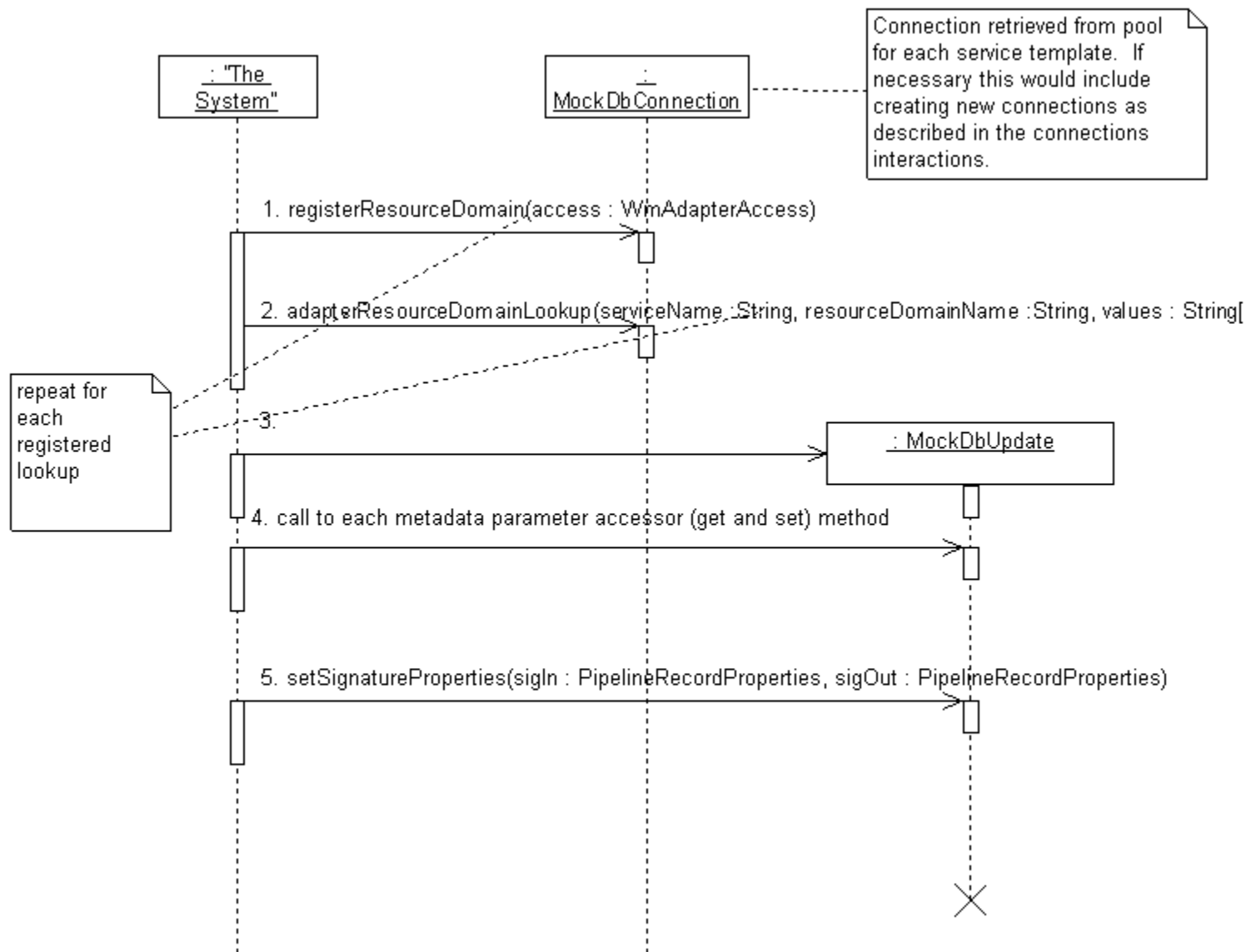
Note:

The cache is not cleared when you recompile the code or reload the package, so it is critical that you refresh the cache manually when loading updated metadata code. Use the **Refresh** button on the Designer toolbar or select **Refresh** from the **Session** menu to refresh the cache.)

Entering Names and Folders for Adapter Service Nodes

The other significant adapter interaction that occurs when creating adapter service nodes occurs after the user of the adapter has entered the name and folder of the new adapter service node.

The following figure describes the parameter interrogation during the creation of an adapter service node:



Before displaying the adapter service editor screens:

- Integration Server invokes the connection's adapterResourceDomainLookup method for each lookup registered with the service.

The values argument in these lookups reflect the dependent parameters' default values that are cached when the user of the adapter selects a connection node. If your default value for a dependent parameter is null, make sure your resource domain lookup code can handle a null value in the values argument.

- After the lookups are complete, Integration Server instantiates the adapter service template class (again), and each of the accessor methods are called.

Values passed to "set" methods come either from the parameter default, or from the result of a resource domain lookup. These accessor method calls merely validate their operation; the

service class instance is not cached. This is the last interaction with the metadata parameter accessor methods during the process of creating an adapter service node. The "set" methods are not called with the final node settings until the service is executed.

Viewing or Editing Adapter Service Nodes

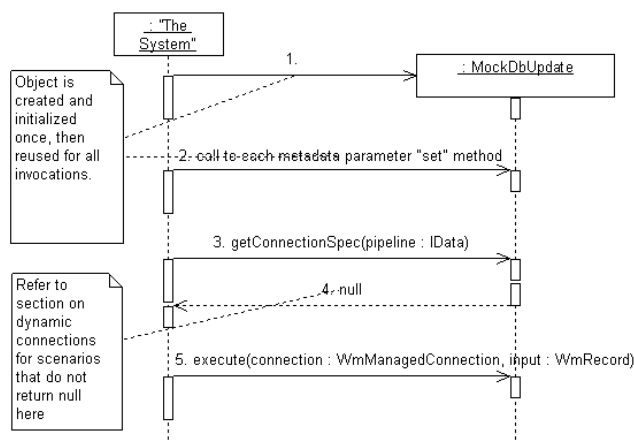
Users may modify the metadata parameter values of an adapter service node using the adapter service editor at any time after the node is created.

When an adapter service node is opened (selected), the adapter service editor performs an `adapterResourceDomainLookup` call for any resource domain values that it has not already cached. This lookup interaction is shown in steps 1 and 2 of the above figure.

Resource domain values are cached in the adapter service editor based on the values of dependent parameters for the adapter service template/connection type combination (that is, for the class, not the node; thus cached values may be used across nodes that are based on the same template and connection type). Whenever the user of the adapter changes the value of a dependency parameter (a parameter upon which a resource domain lookup depends), the adapter service editor checks its cache for a set of resource domain values based on the new value. If an appropriate set of resource domain values is not found in the cache, then the adapter service editor calls the `adapterResourceDomainLookup` method again.

Adapter Check Values operate very much like resource domain lookups. When user of the adapter types a value into a parameter configured with an adapter check value, a call is made to the `adapterCheckValue` method of the corresponding connection class. If the validation succeeds, the adapter service editor caches the checked parameter value as well as the values of any dependent parameters for future use. If a parameter uses the adapter check value feature and is also a dependency parameter for the resource domain lookup of another parameter, the adapter check value validation is performed first. If the validation succeeds, the appropriate lookups are performed. For more information, see [“Adapter Check Value Callbacks” on page 91](#).

Executing Adapter Service Nodes



When an adapter service executes, the Integration Server performs the following:

- Creates a new instance of the corresponding class, unless it is already cached.

- Calls the metadata parameter "set" methods, passing the parameter settings stored in the adapter service node.
- Calls the service's `execute` method, passing a connection instance and a copy of the pipeline wrapped in a `WmRecord` object, as shown in the figure.

Adapter Service Implementation

The example provided in this section demonstrates the mechanics of an adapter service implementation by making full use of design-time and runtime interactions while emulating interactions with an external adapter resource.

The example adapter service simulates a simple database update service, allowing the user of the adapter to select table and column names that create the runtime signature for the resulting adapter service node. Table and column names are provided from hard-coded lists in a mock-connection implementation, as if the data were actually retrieved from the adapter resource. The example also includes interactions with the pipeline based on a dynamic service signature. The example is self-contained; it does not actually interact with any external resource.

The model for adapter services forces syntactic and semantic coupling of code in different methods and classes. Because of this, it might be difficult to understand the process of creating an adapter service by looking at the classes (or even methods) as a unit of work in the development process. For example, adding a metadata parameter can require updating two or more methods in the Adapter Service implementation class, updating up to three methods in the associated connection classes, and adding two entries in the resource bundle. Thus, this section approaches the implementation of a service as a series of activities that you perform, each of which may traverse multiple methods and classes.

This section provides examples of the resulting code, and refers to specific lines.

The tasks for creating an adapter service are as follows:

- Defining a `WmAdapterService` Implementation Class
- Specifying Configuration Metadata for Adapter Service
- Implementing Configuration Resource Domains for Adapter Service
- Manipulating Adapter Service Signature Properties
- Specifying Adapter Service Signature Data
- Specifying Adapter Service Signature Resource Domains
- Implementing the `WmAdapterService.execute` Method
- Updating the Resource Bundle
- Registering Adapter Service in the Connection Factory Implementation Class
- Compiling the adapter
- Reloading Adapter

- Refreshing the Designer cache
- Configuring and Testing Adapter Service Nodes

Example 1: WmAdapterService Implementation Class

```

package com.wm.MyAdapter.services;
import com.wm.adk.cci.interaction.WmAdapterService;
import com.wm.adk.cci.record.WmRecord;
import com.wm.adk.cci.record.WmRecordFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.WmTemplateDescriptor;
import com.wm.data.IData;
import com.wm.data.IDataCursor;
import com.wm.data.IDataFactory;
import com.wm.data.IDataUtil;
import java.util.Hashtable;
import java.util.Locale;
import javax.resource.ResourceException;
import com.wm.MyAdapter.MyAdapter;
public class MockDbUpdate extends WmAdapterService {
    //MockDB Group
    public static final String UPD_SETTINGS_GRP = "Mock Settings";
    public static final String TABLE_NAME_PARM = "tableName";
    public static final String COLUMN_NAMES_PARM = "columnNames";
    public static final String COLUMN_TYPES_PARM = "columnTypes";
    public static final String REPEATING_PARM = "repeating";
    public static final String OVERRIDE_TYPES_PARM = "overrideTypes";
    private String _tableName;
    private String[] _columnNames;
    private String[] _columnTypes;
    private boolean _repeating;
    private String[] _overrideTypes;
    public void setTableName(String val){ _tableName = val;}
    public void setColumnNames(String[] val){ _columnNames = val;}
    public void setColumnType(String[] val){ _columnTypes = val;}
    public void setRepeating(boolean val){ _repeating = val;}
    public void setOverrideTypes(String[] val){_overrideTypes = val;}
    public static final String TABLES_RD = "tablesRD";
    public static final String COLUMN_NAMES_RD = "columnNamesRD";
    public static final String COLUMN_TYPES_RD = "columnTypesRD";
    public static final String OVERRIDE_TYPES_RD = "overrideTypesRD";
    //MockDB Signature Group
    public static final String SIG_SETTINGS_GRP = "Signature";
    public static final String FIELD_NAMES_PARM = "fieldNames";
    public static final String FIELD_TYPES_PARM = "fieldTypes";
    public static final String SIG_IN_PARM = "sigIn";
    public static final String SIG_OUT_PARM = "sigOut";
    private String[] _fieldNames;
    private String[] _fieldTypes;
    public void setFieldNames(String[] val){ _fieldNames = val;}
    public void setFieldTypes(String[] val){ _fieldTypes = val;}
    public void setSigIn(String[] val){}
    public void setSigOut(String[] val){}
    public static final String FIELD_NAMES_RD = "fieldNamesRD";
    public static final String FIELD_TYPES_RD = "fieldTypesRD";
    public void fillWmTemplateDescriptor(WmTemplateDescriptor d,Locale l)
        throws ResourceException

```

```

{
//MockDB Grouping and resource domain setup
d.createGroup(UPD_SETTINGS_GRP, new String [] {
    TABLE_NAME_PARM,
    REPEATING_PARM,
    COLUMN_NAMES_PARM,
    COLUMN_TYPES_PARM,
    OVERRIDE_TYPES_PARM}
);
d.createFieldMap(new String[] {
    COLUMN_NAMES_PARM,
    COLUMN_TYPES_PARM,
    OVERRIDE_TYPES_PARM},
    true);
d.createTuple(new String[]{COLUMN_NAMES_PARM,COLUMN_TYPES_PARM});

d.setResourceDomain(TABLE_NAME_PARM, TABLES_RD, null);
d.setResourceDomain(COLUMN_NAMES_PARM, COLUMN_NAMES_RD,
    new String[]{TABLE_NAME_PARM});
d.setResourceDomain(COLUMN_TYPES_PARM, COLUMN_TYPES_RD,
    new String[]{TABLE_NAME_PARM});
d.setResourceDomain(OVERRIDE_TYPES_PARM, OVERRIDE_TYPES_RD, null);
//MockDB Signature Grouping and resource domain setup
d.createGroup(SIG_SETTINGS_GRP, new String [] {
    FIELD_NAMES_PARM,
    FIELD_TYPES_PARM,
    SIG_IN_PARM,
    SIG_OUT_PARM}
);
d.createFieldMap(new String [] {
    FIELD_NAMES_PARM,
    FIELD_TYPES_PARM,
    SIG_IN_PARM,
    SIG_OUT_PARM},
    false);
d.createTuple(new String[]{FIELD_NAMES_PARM, FIELD_TYPES_PARM});

String [] fieldTupleDependencies = {TABLE_NAME_PARM,
    REPEATING_PARM,
    COLUMN_NAMES_PARM,
    COLUMN_TYPES_PARM,
    OVERRIDE_TYPES_PARM};
d.setResourceDomain(FIELD_NAMES_PARM, FIELD_NAMES_RD, fieldTupleDependencies);
d.setResourceDomain(FIELD_TYPES_PARM, FIELD_TYPES_RD, fieldTupleDependencies);
d.setResourceDomain(SIG_IN_PARM, WmTemplateDescriptor.INPUT_FIELD_NAMES,
    new String[] {FIELD_NAMES_PARM, FIELD_TYPES_PARM});
d.setResourceDomain(SIG_OUT_PARM, WmTemplateDescriptor.OUTPUT_FIELD_NAMES,
    new String[] {FIELD_NAMES_PARM, FIELD_TYPES_PARM});
//Call to setDescription
d.setDescriptions(MyAdapter.getInstance().
    getAdapterResourceBundleManager(), l);
}
public WmRecord execute(WmManagedConnection connection, WmRecord input)
    throws ResourceException
{
    Hashtable[] request = this.unpackRequest(input);
    return this.packResonse(request);
}
private Hashtable[] unpackRequest(WmRecord request) throws ResourceException
{

```



```

Hashtable data[] = null;
IData mainIData = request.getIData();
IDataCursor mainCursor = mainIData.getCursor();
try
{
String tableName = this._tableName;
String[] columnNames = this._columnNames;
if(mainCursor.first(tableName))
{
IData[] recordIData;
if(this._repeating)
{
recordIData = IDataUtil.getIDataArray (mainCursor,tableName);
data = new Hashtable[recordIData.length];
}
else
{
recordIData = new IData[] {IDataUtil.getIData(mainCursor)};
data = new Hashtable[1];
}
for(int rec=0;rec<recordIData.length;rec++)
{
IDataCursor recordCursor = recordIData[rec].getCursor();
data[rec] = new Hashtable();
for(int c = 0; c < columnNames.length;c++)
{
if(recordCursor.first(columnNames[c]))
{
data[rec].put(tableName + "." + columnNames[c],
recordCursor.getValue());
}
}
recordCursor.destroy();
}
}
else
{
throw MyAdapter.getInstance().createAdapterException(9999,
new String[] {"No Request Data"});
}
}
catch (Throwable t)
{
throw MyAdapter.getInstance().createAdapterException(9999,
new String[] {"Error unpacking request data"},t);
}
finally
{
mainCursor.destroy();
}
return data;
}
private WmRecord packResonse(Hashtable[] response) throws ResourceException
{
WmRecord data = null;
try
{
IData[] recordIData = new IData[response.length];
String tableName = this._tableName;
String[] columnNames = this._columnNames;

```

```

for(int rec = 0; rec < response.length; rec++)
{
    recordIData[rec] = IDataFactory.create();
    IDataCursor recordCursor = recordIData[rec].getCursor();
    for(int col = 0; col < columnNames.length;col++)
    {
        IDataUtil.put(recordCursor,columnNames[col],
            response[rec].get(tableName + "." +
                columnNames[col]));
    }
    recordCursor.destroy();
}
IData mainIData = IDataFactory.create();
IDataCursor mainCursor = mainIData.getCursor();
if(this._repeating)
{
    IDataUtil.put(mainCursor,tableName,recordIData);
}
else
{
    IDataUtil.put(mainCursor,tableName,recordIData[0]);
}
mainCursor.destroy();
data = WmRecordFactory.getFactory().createWmRecord("nameNotUsed");
data.setIData(mainIData);
}
catch (Throwable t)
{
    throw MyAdapter.getInstance().createAdapterException(9999,
        new String[] {"Error packing response data"},t);
}
return data;
}
}

```

Example 2: WmManagedConnection Implementation Class Updates

```

package com.wm.MyAdapter.connections;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.*;
import com.wm.adk.error.AdapterException;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.services.MockDbUpdate;
public class SimpleConnection extends WmManagedConnection {
    String hostName;
    int port;
    //Adapter Services variables
    private String[] mockTableNames = { "CUSTOMERS","ORDERS","LINE_ITEMS"};
    private String[][] mockColumnNames = {
        {"name","id", "ssn"},
        {"id","date","customer_id"},
        {"order_id","item_number","quantity","description"}
    };
    private String [][] mockDataTypes = {
        {"java.lang.String","java.lang.Integer", "java.lang.String"},
        {"java.lang.Integer", "java.util.Date", "java.lang.Integer"},
        {"java.lang.Integer", "java.lang.Integer", "java.lang.Integer",

```

```

"java.lang.String"}
};
public SimpleConnection(String hostNameValue, int portValue)
{
    super();
    hostName = hostNameValue;
    port = portValue;
    MyAdapter.getInstance().getLogger().logDebug(9999,
        "Simple Connection created with hostName = "
        + hostName + "and port = " + Integer.toString(port));
}
public void destroyConnection()
{
    MyAdapter.getInstance().getLogger().logDebug(9999,"Simple Connection Destroyed");
}

// The remaining methods support metadata for related services, etc.
// Implement content as needed.
public Boolean adapterCheckValue(String serviceName,
    String resourceDomainName,
    String[][] values,
    String testValue) throws AdapterException
{
    Boolean result = new Boolean(false);
    if(resourceDomainName.equals(MockDbUpdate.OVERRIDE_TYPES_RD))
    {
        try
        {
            Object o = Class.forName(testValue).getConstructor(
                new Class[] {String.class}).newInstance(new Object[]{"0"});
            result = new Boolean(true);
        }
        catch (Throwable t){}
    }
    return result;
}
public ResourceDomainValues[] adapterResourceDomainLookup(String serviceName,
    String resourceDomainName, String[][] values) throws AdapterException
{
    ResourceDomainValues[] results = null;

    //MockDB Group Lookup
    if(resourceDomainName.equals(MockDbUpdate.COLUMN_NAMES_RD) ||
        resourceDomainName.equals(MockDbUpdate.COLUMN_TYPES_RD))
    {
        String tableName = values[0][0];
        for(int x = 0; x < this.mockTableNames.length;x++)
        {
            if(this.mockTableNames[x].equals(tableName))
            {
                ResourceDomainValues columnsRdvs = new ResourceDomainValues(
                    MockDbUpdate.COLUMN_NAMES_RD,this.mockColumnNames[x]);
                columnsRdvs.setComplete(true);
                ResourceDomainValues typesRdvs = new ResourceDomainValues(
                    MockDbUpdate.COLUMN_TYPES_RD, this.mockDataTypes[x]);
                typesRdvs.setComplete(true);
                results = new ResourceDomainValues[] {columnsRdvs,typesRdvs};
                break;
            }
        }
    }
}

```

```

}
//MockDB Signature Group Lookup
else if (resourceDomainName.equals(MockDbUpdate.FIELD_NAMES_RD) ||
resourceDomainName.equals(MockDbUpdate.FIELD_TYPERD))
{
String tableName = values[0][0];
boolean repeating = Boolean.valueOf(values[1][0]).booleanValue();
String[] columnNames = values[2];
String[] columnTypes = values[3];
String[] overrideTypes = values[4];
String[] fieldNames = new String[columnNames.length];
String[] fieldTypes = new String[columnTypes.length];
String optBrackets;
if(repeating)
optBrackets = "[";
else
optBrackets = "";
for (int i = 0; i < fieldNames.length;i++)
{
fieldNames[i] = tableName + optBrackets + "." + columnNames[i];
fieldTypes[i] = columnTypes[i] + optBrackets;
if(overrideTypes.length > i)
{
if (!overrideTypes[i].equals(""))
{
fieldTypes[i] = overrideTypes[i] + optBrackets;
}
}
}
}
results = new ResourceDomainValues[]{
new ResourceDomainValues(MockDbUpdate.FIELD_NAMES_RD,fieldNames),
new ResourceDomainValues(MockDbUpdate.FIELD_TYPERD,fieldTypes)};
}
return results;
}
public void registerResourceDomain(WmAdapterAccess access)
throws AdapterException
{
//MockDB Group Registering Resource Domain
ResourceDomainValues tableRdvs = new ResourceDomainValues(
MockDbUpdate.TABLES_RD,mockTableNames);
tableRdvs.setComplete(true);
access.addResourceDomain(tableRdvs);
access.addResourceDomainLookup(MockDbUpdate.COLUMN_NAMES_RD,this);
access.addResourceDomainLookup(MockDbUpdate.COLUMN_TYPERD,this);
ResourceDomainValues rdvs = new ResourceDomainValues(
MockDbUpdate.OVERRIDE_TYPERD, new String[] {""});
rdvs.setComplete(false);
rdvs.setCanValidate(true);
access.addResourceDomain(rdvs);
access.addCheckValue(MockDbUpdate.OVERRIDE_TYPERD,this);

//MockDB Signature Group Registering Resource Domain
access.addResourceDomainLookup(MockDbUpdate.FIELD_NAMES_RD,this);
access.addResourceDomainLookup(MockDbUpdate.FIELD_TYPERD,this);
}
}

```

Defining WmAdapterService Implementation Class

1. Create a folder structure for the Java package for adapter service implementation. For example: `com\mycompany\adapter\myadapter`. In the example, the Java package created is `com\wm\services`.

Note:

You must create your Java package and classes in the `adapterPackageName\code\source` folder in the webMethods package you created using Designer.

2. Create a class by extending the base class `com.wm.adk.cci.interaction.WmAdapterService`.
In the example, the class created is `MockDBUpdate`.
3. Implement the abstract method `WmAdapterService.execute`.
4. Override the base class implementation of the `WmAdapterService.fillWmTemplateDescriptor` method.

Specifying Configuration Metadata for Adapter Service

The next logical step for implementing an adapter service is to create the metadata that enables the users of the adapter to create adapter service nodes. To do this, you perform the following:

- Create metadata parameters appropriate for the function of the adapter service.
- Describe presentation for those metadata parameters.
- Set the data entry rules for those metadata parameters.

Creating Parameters for Data Entry

The sample implementation includes five metadata parameters that the users of the adapter use for data entry when they create adapter service nodes. Each parameter has:

- An accessor method.
- A variable to hold the configured values.
- A String constant containing the name of the parameter.
- A set of resource bundle entries with a localizable parameter name and description. For information about metadata parameters, see [“Metadata Model for Connection” on page 63](#).

The following table describes the purpose of each of these parameters for data entry:

Parameter	Description
<code>tableName</code>	Enables the users of the adapter to select a table to update.

Parameter	Description
<i>columnNames</i>	Lists the columns in the selected table for updating. The resource domain lookup for this parameter depends on the value in the <i>tableName</i> parameter.
<i>columnTypes</i>	Contains the default type associated with the <i>columnName</i> .
<i>overrideTypes</i>	Enables the users of the adapter to type a different data type for the column value.
<i>repeating</i>	A <code>Boolean</code> flag indicating whether the service is used to update a single row or multiple rows of the table.

Specifying the Display and Data Entry Attributes of the "Data Entry" Parameters

- After creating the parameters, specify the display and data entry attributes by calling various methods of the `WmTemplateDescriptor` interface from the service's `fillWmTemplateDescriptor` method.
- The example code places each data entry parameter into a single group (in display order) referenced by the constant `UPD_SETTINGS_GRP`. A constant instead of a string is used to name the group, because the same value is used in the resource bundle to specify a localizable group name.

Placing the Column Names and Column Data Types Parameters in a Field Map

Next, the example places *columnNames* and the two types parameters in a field map. The use of two data type columns in the fieldMap warrants further discussion. The desired behavior is to:

- Automatically update the data type when the column name changes.
- Allow the users of the adapter to enter an alternative data type for the given field.
- The adapter must convert the data at run time.

To have the type value change with the *columnName*, the resource domain associated with the parameter must be "complete" (as specified by the `setComplete` method being set to `true`). Users are not allowed to type values into fields of a "complete" resource domain. Because of these conflicting constraints, it is necessary to have two parameters with different resource domain associations: one updated by the adapter service editor, and the other by the users of the adapter.

Placing the Column Names and Column Types Parameters in a Tuple

Finally, the *columnNames* and *columnTypes* parameters are placed in a tuple. In a tuple, not only are the resource domain lookups performed together, but the adapter service editor maintains a relationship between the parameter settings such that when *value[n]* is selected from the resource domain value list for *columnNames*, then *value[n]* is automatically selected for the corresponding *columnTypes* value. Alternatively, you can make a resource domain lookup for *columnTypes* that depends on the corresponding value of *columnNames* and then determines the appropriate type in the lookup.

Implementing Configuration Resource Domains for Adapter Service

The next step for implementing an adapter service is to:

- Define and implement the resource domains required for the metadata parameters that you created.
- Identify the values upon which those resource domains depend.

For each parameter that requires either a resource domain to supply a value or that requires a validity check for values supplied by the users of the adapter, you must:

1. In the service's `fillWmTemplateDescriptor` method, call `WmTemplateDescriptor.setResourceDomain` method, passing the name of the parameter, the name of the resource domain, and an array of the names of any parameters on which the resource domain depends.
2. In the associated connection class's `WmManagedConnection.registerResourceDomain` method, call `WmAdapterAccess.addResourceDomain(ResourceDomainValues)` to register the resource domain support.
3. Implement the code to populate the resource domain values and/or the "check values".
4. In the associated connection class's `WmManagedConnection.registerResourceDomain` method, call `WmAdapterAccess.addResourceDomainLookup(Resource_Domain_Name, WmManagedConnection)` method to add the resource domains that have to be looked up.
 - The first parameter, *tableName*, is associated with a new resource domain called `tableNameRD`.
 - The list of available tables in this example does not depend on any other parameters, so the dependency list is left null.
 - Furthermore, since the list does not change once it is retrieved from the resource, it can be implemented as a complete resource domain during registration.
 - When a `ResourceDomainValues` object is provided in `registerResourceDomain`, there is no need to add support for the resource domain in the lookup method.
 - The resource domains `columnNamesRD` and `columnTypesRD` are created to support the `columnNames` and `columnTypes` parameters, respectively. The `columnNamesRD` resource domain depends on the value of the `tableName` parameter. Since `columnNames` and `columnTypes` were placed in a tuple, `columnTypesRD` must also depend on the value of `tableName`.
 - The lookup implementation for these resource domains checks for the name of the resource domain and returns the values for both resource domains in a single response array. The `columnNamesRD` resource domain is always passed as the resource domain name in this lookup. Adding the '||' check decouples the lookup implementation from the order in which the parameters were placed in the tuple.

- The final resource domain, *overrideTypesRD* associated with the *overrideTypes* parameter, is used to validate data entered by the users of the adapter into a Java class that contains the data at run time. The specifics of the check implementation are probably unrealistic for a real-world environment, but all the mechanics of doing a check are demonstrated. Remember to set the `setCanValidate` method to `true` in the `WmManagedConnection.registerResourceDomain` method.

Manipulating Adapter Service Signature Properties

In addition to controlling the names and types of the fields in an adapter service signature, the adapter has control over several other aspects of a service's structure and behavior. These aspects can be broadly divided into two categories:

- Template-based properties which apply to all the adapter service nodes associated with the service implementation class.
- Signature field properties which are specific to the records and fields that make up the signature of a particular node.

Template-Based Signature Properties

In the implementation of the adapter service's `fillWmTemplateDescriptor` method, the following signature-related features may be configured using the following methods:

Method Name	Description
<code>WmDescriptor.setShowConnectionName(boolean)</code>	<p>Includes the reserved string field named \$connectionName in the service input signature. Possible Values are:</p> <ul style="list-style-type: none"> ■ <code>true</code>. Default. Includes the reserved string field named \$connectionName in the service input signature. This field allows the flow adapter developers to control the connection pool used during each invocation of the service. The \$connectionName field is inserted in the top-level signature record, outside the wrapper. ■ <code>false</code>. Excludes the reserved string field named \$connectionName from the service input signature. <p>For more information about the effect of \$connectionName on the run time behavior of the service, see the javadoc for <code>WmDescriptor.showConnectionName()</code>.</p>
<code>WmDescriptor.setSignatureWrapped(boolean)</code>	Wraps the adapter-defined signature fields in a record called <i>xxxInput</i> for input fields and <i>xxxOutput</i> for output fields, where xxx represents the name of the

Method Name	Description
	<p>adapter service node when set to <code>true</code>. Possible Values are:</p> <ul style="list-style-type: none"> ■ <code>true</code>. Default. ■ <code>false</code>.
<code>WmDescriptor.setPassFullPipeline(boolean)</code>	<p>Enables the access to all fields in the pipeline. Possible values are:</p> <ul style="list-style-type: none"> ■ <code>true</code>. Default. The adapter service can access all the fields in the pipeline. ■ <code>false</code>. The adapter service may only read or write to pipeline fields that are part of the service's defined signature.

Signature Field Properties

All the Integration Server services (including adapter services, flow services, and Java services) contain an input and output signature definition that identifies the names and types of the pipeline fields, read or written by the service. For each of these fields, there are also a number of properties that can be used to document the intended use of the field or to create constraints on what data is valid for that field at run time. For more information on these properties, see the *webMethods Service Development Help* for your release.

In the flow services and Java services, signature fields and their properties can be modified by the user. For the adapter services, the adapter defines resource domains that act as callbacks from the developer tool that allows the adapter to specify the name, data type, and structure of the service's signature while the service is being configured. A separate callback mechanism allows the adapter to control a limited subset of signature field constraint properties.

The signature field constraints that are controlled by the adapter include:

Signature Field Constraint Field Behavior	
Required	If <code>true</code> , the field must be present on the pipeline.
Allow null	If <code>false</code> , when the field is present, the field cannot hold a null value.
Allow unspecified fields	Controls whether the document (record) element can hold fields that are not specifically identified in the signature. This constraint has no effect on non-document-type signature elements.

The controls for these constraints are disabled so the user can view the value, but not modify it. The remaining properties (that are not related to name and type) are enabled and may be managed like any other service.

Note:

Like other service types, signature constraints are only enforced at run time if the associated **Validate input** or **Validate output** check box is selected by the user.

Adapters gain access to a signature field's constraints by overriding the `setSignatureProperties()` method in the `WmAdapterService` implementation class. This method is called whenever the adapter service node is saved, and whenever the user views the input/output panel of an unsaved adapter service. Because this call is made while the adapter service changes are still in progress, the call is made against a temporary object that has all metadata parameter settings as existing in the adapter service editor. This is not the same object that is used for runtime service invocations.

The ADK includes three classes that are used to provide access to signature property information.

<code>PipelineVariableProperties</code>	Abstract base class that represents any signature element (field or record). It exposes read access to all common signature element properties (name, data type, comments, etc.) and read/write access to the Required and Allow null constraints.
<code>PipelineFieldProperties</code>	Subclass of <code>PipelineVariableProperties</code> that represents a single non-document-type field in the signature. It adds read only access to properties that are specific to non-document elements (pick list choices, content type, etc.)
<code>PipelineRecordProperties</code>	Subclass of <code>PipelineVariableProperties</code> that represents a single document-type element in the signature. It adds read/write access to the Allow unspecified fields property, and methods for accessing the member elements of the record.

The `WmAdapterService.setSignatureProperties()` method receives two `PipelineRecordProperties` objects as arguments, one for the input signature and the other for the output signature. The adapter must use the accessor/navigation methods available through the two `PipelineRecordProperties` objects to locate the signature elements for which the constraints have to be managed. Within the `setSignatureProperties()` implementation, it is frequently useful to use the `WmAdapterService.inputRecordName()` and `WmAdapterService.outputRecordName()` methods to get the names of the respective signature wrappers (if wrappers are enabled). If a connection is needed to get the constraint information from a connection target, use the `WmAdapterService.retrieveConnection()` method.

Note:

This is the only place `retrieveConnection` must be used; using it from the service's `execute` method causes errors.

The code listing below demonstrates a simple implementation of the signature constraint management.

```
protected void setSignatureProperties(
    PipelineRecordProperties inputSigProps,
    PipelineRecordProperties outputSigProps) throws ResourceException
{
    if(this._tableName != null)
    {
```

```

// need a connection to look up info about table being updated
MockDbConnection conn = (MockDbConnection)retrieveConnection();
TableInfo info = conn.getTableInfo(this._tableName);
updateSignatureRecord(inputSigProps, info, _columnNames,
    inputRecordName());
updateSignatureRecord(outputSigProps, info, _columnNames,
    outputRecordName());
    }
}
private void updateSignatureRecord(PipelineRecordProperties inputSigProps,
    TableInfo tInfo, String[] columnNames, String wrapperName)
{
    PipelineRecordProperties wrapperRec =
        (PipelineRecordProperties)inputSigProps.findByPath(wrapperName);
    wrapperRec.setAllowNull(false);
    wrapperRec.setAllowUnspecifiedFields(false);
    wrapperRec.setRequired(true);
    if(columnNames != null)
    {
        for(int i = 0; i < columnNames.length; i++)
        {
            ColumnInfo cInfo = tInfo.getColumnInfo(columnNames[i]);
            PipelineVariableProperties fieldProps =
                wrapperRec.findByPath(columnNames[i]);
            fieldProps.setAllowNull(false);
            fieldProps.setRequired(cInfo.isRequired());
        }
    }
}
}
}

```

Specifying Adapter Service Signature Data

After implementing the configuration logic for the adapter service template, implement the logic that defines the runtime signature of a configured adapter service node. To do so, create the following additional metadata parameters:

- *sigIn* and *sigOut*.

The reserved signature resource domains uses these parameters.

- *fieldNames* and *fieldTypes*

These parameters are the dependency parameters in which you build the signature data. Because the example update service has the same input and output signature, only one set of name and type parameters is necessary. The relationship between these parameters is established in the `WmTemplateDescriptor`. The mechanics of signature construction is discussed in [“Adapter Service Node Signatures” on page 93](#).

These parameters do not accept input from the users of the adapter; from the user's perspective, they are largely redundant with information provided elsewhere in the adapter service editor. In most implementations, these parameters are included in the same group with the configuration parameters, but are hidden from the users of the adapter. For demonstration purposes, these parameters remain visible in the example, but they are located in a separate group. Except for that, the metadata constructs for these parameters follow the same basic rules for specifying a signature for any service.

Specifying Adapter Service Signature Resource Domains

The resource domain implementation for the signature parameters is a little more complex. The `fieldNamesRD` and `fieldTypesRD` resource domains are designed to depend on the values of each of the configuration parameters as described in [“Specifying Configuration Metadata for Adapter Service” on page 109](#). In the `WmManagedConnection` lookup implementation, both `fieldNamesRD` and `fieldTypesRD` are constructed as "complete" resource domains.

For `fieldNamesRD`, the `tableName` and `columnName` parameter values are used to form a hierarchical signature in which the `columnName` elements are contained in an aggregate named by the `tableName`. If the parameter named `repeating` is set to `true`, then the `tableName` aggregate is converted to an array by inserting square brackets ("`[]`") in the field name.

For `fieldTypesRD`, a value in the `overrideTypes` parameter has precedence over a value in `columnTypes`, and the `repeating` parameter is used to specify whether the type repeats.

Implementing the `WmAdapterService.execute` Method

The final step for implementing an adapter service is to implement its `execute` method. This method is specific to the resource with which the adapter communicates. In most cases, the adapter must interact with the pipeline at the beginning and/or end of the `execute` method. The methods `unpackRequest` and `packResponse` demonstrate an effective method of interacting with the pipeline using the same metadata parameters that were used to create the signature.

Important:

The `unpackRequest` and `packResponse` methods read class fields, but they never write to them. This is important because of the multi-threaded nature of the adapter service execution. At run time, exactly one `WmAdapterService` object corresponds to each adapter service node defined in the namespace. All invocations of a given adapter service node call the `execute` method on the same object. If more than one thread is executing the service at the same time, then updates to the class fields by one thread inevitably collide with those of another thread.

Updating the Resource Bundle

Update the resource bundle with a display name, and description to make the service more usable.

```
package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.services.MockDbUpdate;
..
..
public class MyAdapterResource extends ListResourceBundle implements MyAdapterConstants{
    ..
    ..
    static final Object[][] _contents = {
        ..
        ..
        //MockDB Group Resource Domain Values
        ,{MockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
```

```

    "Mock Update Service"}
, {MockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
  "Simulates a database update service"}
, {MockDbUpdate.UPD_SETTINGS_GRP + ADKGLOBAL.RESOURCEBUNDLEKEY_GROUP,
  MockDbUpdate.UPD_SETTINGS_GRP}
, {MockDbUpdate.TABLE_NAME_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
  "Table Name"}
, {MockDbUpdate.TABLE_NAME_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
  "Select Table Name"}
, {MockDbUpdate.COLUMN_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
  "Column Names"}
, {MockDbUpdate.COLUMN_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
  "Name of column updated by this service"}
, {MockDbUpdate.COLUMN_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
  "Column Types"}
, {MockDbUpdate.COLUMN_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
  "Default data type for column"}
, {MockDbUpdate.OVERRIDE_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
  "Override Data Types"}
, {MockDbUpdate.OVERRIDE_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
  "Type to override column default"}
, {MockDbUpdate.REPEATING_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
  "Update Multiple Rows?"}
, {MockDbUpdate.REPEATING_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
  "Select if input will include multiple rows to update"}
//MockDB Signature Group Resource Domain Values
, {MockDbUpdate.SIG_SETTINGS_GRP + ADKGLOBAL.RESOURCEBUNDLEKEY_GROUP,
  MockDbUpdate.SIG_SETTINGS_GRP}
, {MockDbUpdate.FIELD_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
  "Field Names"}
, {MockDbUpdate.FIELD_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
  "Name of Field"}
, {MockDbUpdate.FIELD_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
  "Field Type"}
, {MockDbUpdate.FIELD_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
  "Type of Field"}
, {MockDbUpdate.SIG_IN_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
  "Input Signature"}
, {MockDbUpdate.SIG_IN_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
  "Input Signature"}
, {MockDbUpdate.SIG_OUT_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
  "Output Signature"}
, {MockDbUpdate.SIG_OUT_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
  "Output Signature"}
}
protected Object[][] getContents() {
    // TODO Auto-generated method stub
    return _contents;
}
}
}

```

Registering Adapter Service in the Connection Factory Implementation Class

You must register each adapter service template class in the `WmManagedConnectionFactory` implementation class. Pass the class name to the `ResourceAdapterMetadataInfo.addServiceTemplate` method in the `SimpleConnectionFactory.fillResourceAdapterMetadataInfo` method in the `WmManagedConnectionFactory`

implementation class. In the example, the *MockDbUpdate* class is registered in the *SimpleConnectionFactory* connection factory implementation class:

For example: *SimpleConnectionFactory* class

```
import com.wm.MyAdapter.services.*;
.
.
public class SimpleConnectionFactory extends WmManagedConnectionFactory implements
MyAdapterConstants {
.
.
    public void fillResourceAdapterMetadataInfo(
        ResourceAdapterMetadataInfo info, Locale locale)
    {
        info.addServiceTemplate(MockDbUpdate.class.getName());
    }
}
```

Compiling Adapter

Compile your adapter as described in [“Compiling the Adapter”](#) on page 43.

Reloading Adapter

Reload your adapter as described in [“Loading, Reloading, and Unloading Packages”](#) on page 52.

Refreshing the Designer cache

Refresh the Designer cache.

Configuring and Testing Adapter Service Nodes

Before you configure adapter service nodes, ensure that you have configured a connection node as described in [“Configuring and Testing Connection Nodes”](#) on page 71. You can use Designer to configure adapter service nodes.

➤ To configure the adapter service nodes

1. Start Designer.

Note:

Make sure that the Integration Server, with which you want to use Designer, is running.

2. Select a namespace node package where you want to create the adapter service node.
3. Create a folder in the selected package and navigate to that folder in the Package Navigator section.

4. Select **File > New**.
5. Select **Adapter Service** from the list of elements.
6. In the **Create a New Adapter Service** screen, type a name for your service in the **Element name** field and click **Next**.
7. In the **Select Adapter Type** screen, select the name of your adapter and click **Next**.
8. In the **Select an Adapter Connection Alias** screen, select the appropriate adapter connection name and click **Next**.
9. In the **Select a Template** screen, select an adapter service template and click **Next**.
10. Click **Finish**.
11. Specify values for the tab that is specific for your adapter resource (such as Query, Update, Add, or Delete tab).
12. Specify values for the **Input/Output** tab and the **Settings** tab. For more information, see the *webMethods Service Development Help* for your release.
13. Select **File > Save**.

Configuring and Testing Adapter Service Nodes

Before you configure adapter service nodes, ensure that you have configured a connection node as described in [“Configuring and Testing Connection Nodes” on page 71](#). You can use Designer to configure adapter service nodes.

➤ To configure the adapter service nodes

1. Start Designer.

Note:

Make sure that the Integration Server, with which you want to use Designer, is running.

2. Select a namespace node package where you want to create the adapter service node.
3. Create a folder in the selected package and navigate to that folder in the Package Navigator section.
4. Select **File > New**.
5. Select **Adapter Service** from the list of elements.

6. In the **Create a New Adapter Service** screen, type a name for your service in the **Element name** field and click **Next**.
7. In the **Select Adapter Type** screen, select the name of your adapter and click **Next**.
8. In the **Select an Adapter Connection Alias** screen, select the appropriate adapter connection name and click **Next**.
9. In the **Select a Template** screen, select an adapter service template and click **Next**.
10. Click **Finish**.
11. Specify values for the tab that is specific for your adapter resource (such as Query, Update, Add, or Delete tab).
12. Specify values for the **Input/Output** tab and the **Settings** tab. For more information, see the *webMethods Service Development Help* for your release.
13. Select **File > Save**.

6 Polling Notifications

■ Overview	122
■ Polling Notification Classes	123
■ Polling Notification Callbacks	125
■ Metadata Model for Polling Notifications	125
■ Polling Notification Interactions	126
■ Polling Notification Implementation	129
■ Configuring and Testing Polling Notification Nodes	138
■ Cluster Support for Polling Notifications	143

Overview

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.

The users of the adapter can perform the following:

- Create a polling notification using Designer. An adapter connection node created earlier is assigned to the notification. At the same time, Designer creates a Document Type that describes the data generated by the polling notification when it executes. The notification publishes this document to Integration Server. For more information on Integration Server publishable documents, see the *Publish-Subscribe Developer's Guide* for your release.
- Create a Integration Server trigger to process a document published by the notification. When Integration Server receives a document, the trigger invokes the flow or Java service registered with the trigger. The service then processes the data contained in the notification's document.
- Configure the notification scheduling parameters that specify the interval at which Integration Server must invoke the notification, and then enable the notification, using Integration Server Administrator. For instructions on creating and using polling notification nodes, see [“Configuring and Testing Polling Notification Nodes” on page 138](#).

For example, when a record is inserted in the database table, which is monitored by a polling notification:

- The polling notification publishes the polling notification document to Integration Server messaging system.
- Integration Server receiving the published document, triggers a flow or Java service that processes the data contained in the document.

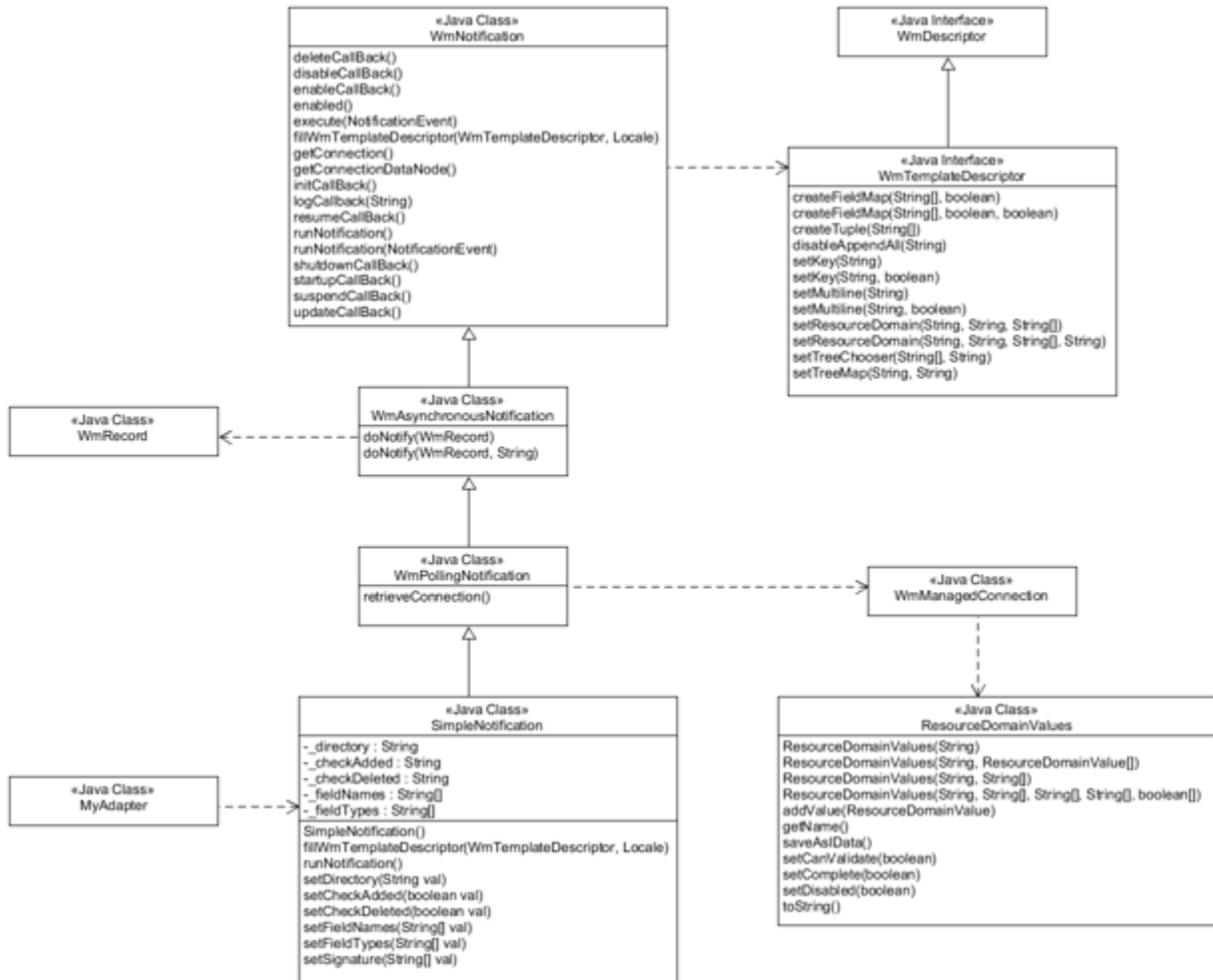
Implementing Polling Notifications

The implementation of a polling notification is similar to the implementation of an adapter service. Each implementation includes a Java class extending an ADK base class, and a namespace node in which design-time configuration data is stored. In the Java class, the metadata model for polling notifications is nearly identical to that of adapter services. The parameters to configure a polling notification are built from the polling notifications' metadata.

The primary difference between adapter services and polling notifications is the runtime behavior of polling notifications. Polling notifications cannot be directly invoked from a flow service or from Designer. Instead, Integration Server automatically invokes a polling notification in 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.

Polling Notification Classes

The following figure shows the classes provided by the ADK to support polling notifications and the `com.wm.adk.notification.WmPollingNotifications`' implementation class `SimpleNotification`.



Polling Notification Implementation Classes

Create a polling notification service by extending `com.wm.adk.notification.WmPollingNotification` base class. You must override the following base class methods in your `WmPollingNotification` implementation class:

Method	Description
<code>fillWmTemplateDescriptor</code>	Modifies how metadata parameters are handled during data entry similar to the <code>WmAdapterService.fillWmTemplateDescriptor</code> method. Failing to override this method results in a runtime error. For more information, see “WmTemplateDescriptor Interface” on page 79.

Method	Description
runNotification	<p>Receives no input and returns results by calling the <code>WmAsynchronousNotification.doNotify</code> method, and passing it a <code>WmRecord</code> instance that must conform to the output signature of the polling notification node. For more information, see “Specifying Notification Signatures (Document Type)” on page 135. The <code>WmRecord</code> is constructed in exactly the same way it is constructed for adapter services. For more information, see “Adapter Service Execution” on page 96.</p>
doNotify	<p>The <code>WmAsynchronousNotification</code> class provides two forms of <code>doNotify</code> method:</p> <ol style="list-style-type: none"> <p><i>Process the notification Exactly Once</i></p> <pre>public void doNotify(WmRecord rec, String msgID)</pre> <p>Receives two input parameters: <code>WmRecord</code> object, and <code>String</code> object for message ID. Provides a resource specific <code>msgID</code> value with each notification record:</p> <ul style="list-style-type: none"> ■ The adapter implementation must guarantee that the value of <code>msgID</code> is unique and constant for each notification event. ■ A <i>notification event</i> is defined as any activity on the adapter resource that causes the <code>WmPollingNotification.runNotification</code> implementation to call <code>WmAsynchronousNotification.doNotify</code>. ■ The <code>msgID</code> is never duplicated for different notification events, but the <code>msgID</code> is the same if the same notification event is retrieved multiple times from the adapter resource, even in a failure-recovery scenario. <p>Integration Server guarantees that <code>msgID</code> values generated by different notification nodes are unique. This is accomplished by combining the <code>msgID</code> value provided by the adapter with a <code>GUID</code> created by Integration Server, and associated with the notification node when it is created.</p> <div style="background-color: #f0f0f0; padding: 10px;"> <p>Note: A fixed number of characters are available in Integration Server to hold a notification ID. Of these, the <code>WmART</code> package reserves a certain number to hold a unique ID that is inserted prior to dispatching a notification. The remaining characters are available to you when calling <code>WmAsynchronousNotification.doNotify(WmRecord rec, String msgId)</code>. The length (number of characters) of the value in <code>msgId</code> must not exceed a particular limit. Call the <code>WmAsynchronousNotification.adapterMaxMessageIdLen</code> method to determine this limit. For more information, see the Javadoc.</p> </div> <p><i>Process the notification</i></p>

Method	Description
	<code>public void doNotify(WmRecord rec)</code>
	Receives one input parameters: WmRecord object.

Polling Notification Callbacks

The base class `WmPollingNotification` defines a set of callback methods that you can override in the notification implementation class. The following table describes when these methods are called, and the impact of an exception thrown from the method. For complete details, see the Javadoc for the `WmPollingNotification` class.

User Actions	Callbacks Received
Notification node is deleted or renamed.	<code>deleteCallBack</code>
Notification node is disabled.	<code>disableCallBack</code>
Disabled notification node is enabled.	<code>enableCallBack</code>
Notification node is created, the package is enabled.	<code>initCallBack</code>
Suspended notification node is enabled.	<code>resumeCallBack</code>
Notification node is disabled or suspended, the Integration Server is shutdown, the package is disabled.	<code>shutdownCallBack</code>
Notification node is enabled or resumed, the Integration Server starts.	<code>startupCallBack</code>
Enabled notification node is suspended.	<code>suspendCallBack</code>
Notification node is modified, but not called when notification node is created.	<code>updateCallBack</code>

Note:

In all cases, an `AdapterException` will cause the associated connection to be destroyed and removed from the pool.

Metadata Model for Polling Notifications

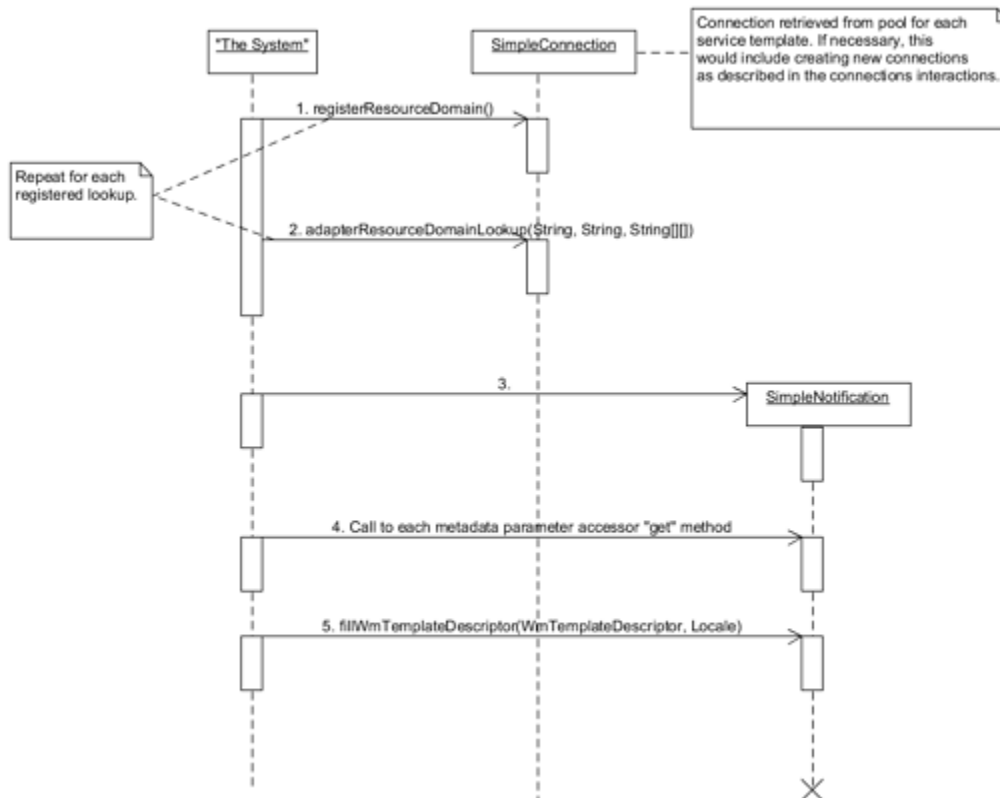
The metadata model for polling notifications is identical to the model for adapter services, except for the following:

- A polling notification has no input signature. The output signature is constructed in the same way, but the server uses it to generate a Document Type node that enables triggers to identify notification data in the node.
- You register polling notifications in the `WmAdapter.fillAdapterTypeInfo` method instead of the `WmManagedConnectionFactory.fillResourceAdapterMetadataInfo` method.

Polling Notification Interactions

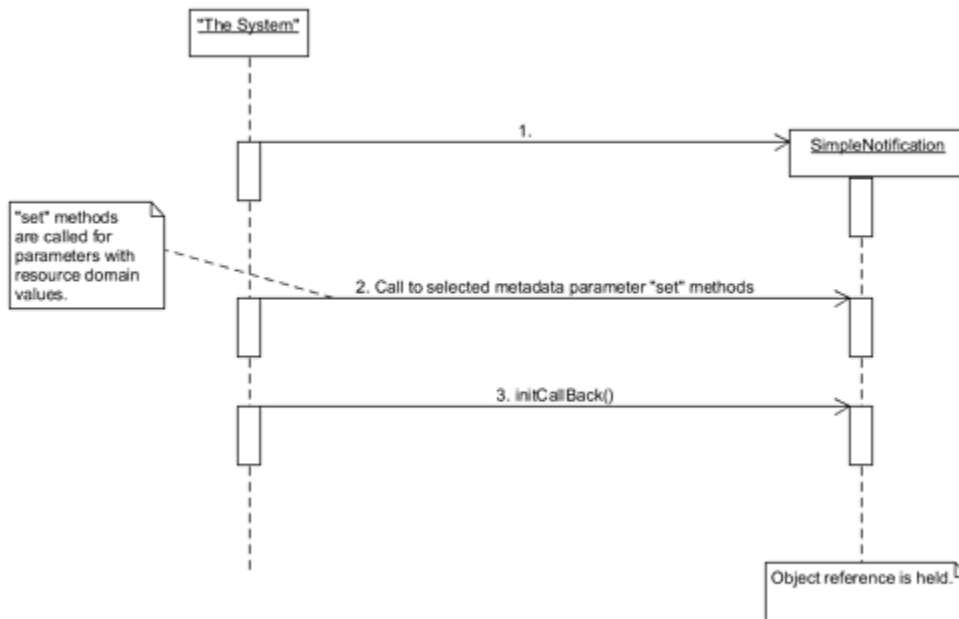
Although polling notifications are structurally similar to adapter services, the dynamic model is similar only in the way in which metadata is initialized, which is described as follows

Loading Polling Notification Templates



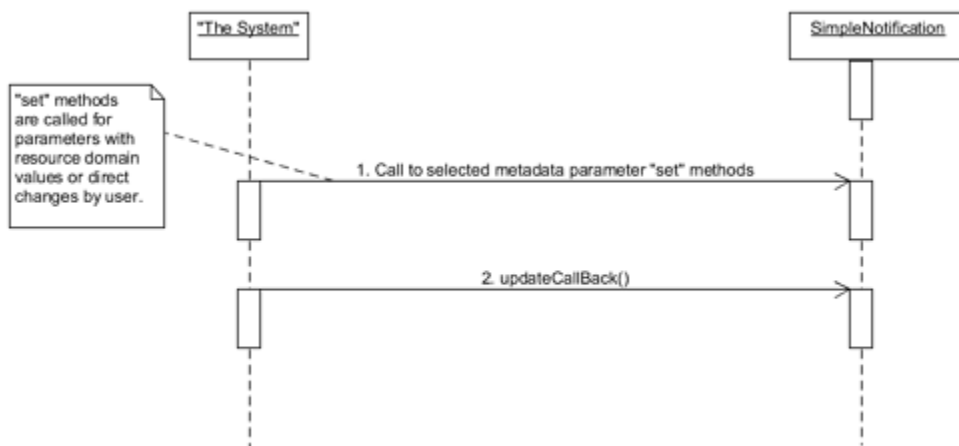
As with adapter services, Designer caches metadata values for polling notifications. These values include resource domain values and template descriptor information. The following figure shows the interactions within the adapter as Designer loads its cache for a polling notification. This interaction occurs either when a new polling notification node is created, or an existing one is viewed (if the data is not already held in the Designer cache).

Creating and Loading Polling Notification Nodes



When a user of the adapter creates a new polling notification node, or loads an existing node during server startup or package startup, the server instantiates the appropriate class and executes the `initCallBack` method. The package containing the polling notification node, holds the object reference of the polling notification node for the lifetime of the package. If this interaction is initiated by a package load, and the polling notification node is enabled, the enable/startup interaction occurs immediately afterwards.

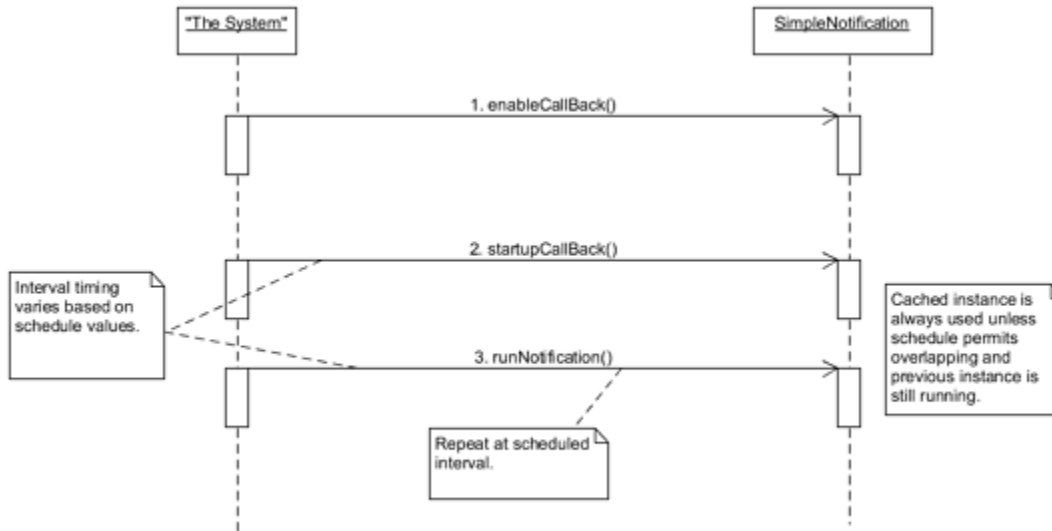
Updating Polling Notifications



Unlike with adapter services, polling notification parameter values are updated each time a user of the adapter saves the values in Designer. After the "set" methods pass the modified values to

the object instance, the notification calls the `updateCallBack` method. If that method throws an exception, it prevents the values from being persisted in the notification node.

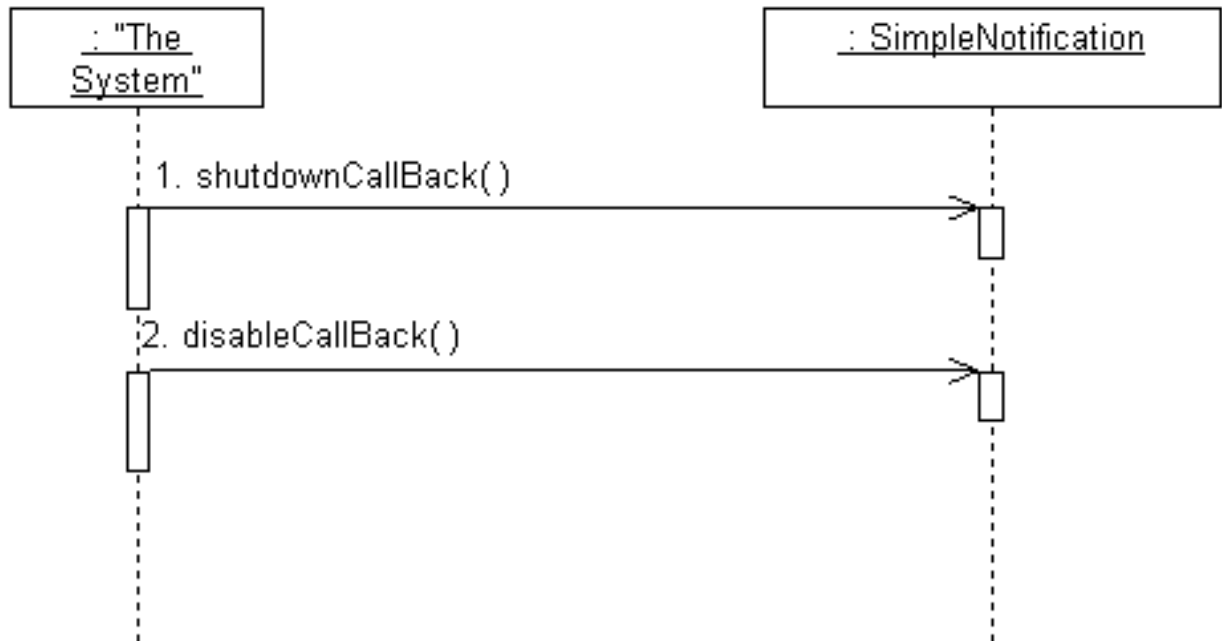
Enabling Polling Notifications



When a user of the adapter enables a polling notification using Integration Server Administrator, the notification calls the `enableCallBack` method before the `startupCallBack` method. If the node was previously enabled, and the user is simply starting up the notification after the package loads, then the `enableCallBack` call is skipped. An exception from either method call disables the notification node.

The server calls the `runNotification` method at regular intervals, based on the scheduling parameters that specify the interval at which Integration Server must invoke the notification. The same object instance is always used unless the schedule is configured to allow overlapping, and the previous call to `runNotification` has not completed.

Disabling Polling Notifications



The figure shows the interactions that occur when a user of the adapter explicitly disables a polling notification node. If a node is shut down by any other means, the `disableCallBack` is skipped.

Polling Notification Implementation

The tasks for implementing a polling notification are as follows:

- Defining a `WmPollingNotification` Implementation Class
- Specifying Configuration Metadata for Polling Notifications
- Implementing Configuration Resource Domains for Polling Notifications
- Specifying Notification Signatures (Document Type)
- Manipulating Adapter Notification Document Properties
- Implementing Signature Resource Domains
- Implementing the `WmPollingNotification.runNotification` Method and Callbacks
- Updating the Resource Bundle
- Registering Polling Notifications in the Adapter
- Compiling the Adapter
- Reloading Adapter

- Refreshing the Designer cache
- Configuring and Testing Polling Notification Nodes

The example polling notification implementation monitors the contents of a directory and sends notifications when files are added to, or removed from the directory. Although this example is very simple, it demonstrates most of the notification capabilities, except for callbacks.

Note:

This example implements a design strategy that enables you to encapsulate the resource domain support inside an adapter service or notification. This strategy is discussed in . You do not need to fully understand this strategy to understand the example code. However, if you are uncomfortable with this strategy, you may implement those methods in your connection implementation. You will have to adjust the method signatures and the "this" references appropriately. The "this" reference refers to the notification. If you move the methods to the connection, then the "this" refers to the connection.

Example Polling Notification Class

```
package com.wm.MyAdapter.notifications;
import com.wm.adk.cci.record.WmRecord;
import com.wm.adk.cci.record.WmRecordFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.error.AdapterException;
import com.wm.adk.metadata.ResourceDomainValues;
import com.wm.adk.metadata.WmAdapterAccess;
import com.wm.adk.metadata.WmTemplateDescriptor;
import com.wm.adk.notification.WmPollingNotification;
import java.io.File;
import java.util.ArrayList;
import java.util.Locale;
import javax.resource.ResourceException;
import com.wm.MyAdapter.MyAdapter;
public class SimpleNotification extends WmPollingNotification
{
    public static final String NOTIFICATION_SETUP_GROUP = "SimpleNotification";
    public static final String DIRECTORY_PARM = "directory";
    public static final String CHECK_ADDED_PARM = "checkAdded";
    public static final String CHECK_DELETED_PARM = "checkDeleted";
    public static final String SIG_FIELD_NAMES_PARM = "fieldNames";
    public static final String SIG_FIELD_TYPES_PARM = "fieldTypes";
    public static final String SIG_PARM = "signature";
    public static final String DIRECTORIES_RD =
        "SimpleNotification.directories.rd";
    public static final String FIELD_NAMES_RD =
        "SimpleNotification.fieldNames.rd";
    public static final String FIELD_TYPES_RD =
        "SimpleNotification.fieldTypes.rd";
    private String _directory;
    private boolean _checkAdded;
    private boolean _checkDeleted;
    private String[] _fieldNames;
    private String[] _fieldTypes;
    public void setDirectory(String val){_directory = val;}
    public void setCheckAdded(boolean val){_checkAdded = val;}
    public void setCheckDeleted(boolean val){_checkDeleted = val;}
}
```

```

public void setFieldNames(String[] val){_fieldNames = val;}
public void setFieldTypes(String[] val){_fieldTypes = val;}
public void setSignature(String[] val){}
private ArrayList _fileList = new ArrayList();
public SimpleNotification(){}
public void fillWmTemplateDescriptor(WmTemplateDescriptor descriptor, Locale l)
    throws ResourceException
{
    descriptor.createGroup(NOTIFICATION_SETUP_GROUP,
        new String[]{DIRECTORY_PARM, CHECK_ADDED_PARM, CHECK_DELETED_PARM,
            SIG_FIELD_NAMES_PARM, SIG_FIELD_TYPES_PARM, SIG_PARM});
    descriptor.createFieldMap(
        new String[]{SIG_FIELD_NAMES_PARM, SIG_FIELD_TYPES_PARM, SIG_PARM},
        false);
    descriptor.setHidden(SIG_FIELD_NAMES_PARM);
    descriptor.setHidden(SIG_FIELD_TYPES_PARM);
    descriptor.setHidden(SIG_PARM);
    descriptor.createTuple(
        new String[]{SIG_FIELD_NAMES_PARM, SIG_FIELD_TYPES_PARM});
    descriptor.setResourceDomain(DIRECTORY_PARM, DIRECTORIES_RD, null);
    descriptor.setResourceDomain(SIG_FIELD_NAMES_PARM, FIELD_NAMES_RD, null);
    descriptor.setResourceDomain(SIG_FIELD_TYPES_PARM, FIELD_TYPES_RD, null);
    descriptor.setResourceDomain(SIG_PARM, WmTemplateDescriptor.OUTPUT_FIELD_NAMES,
        new String[]{SIG_FIELD_NAMES_PARM, SIG_FIELD_TYPES_PARM});
    descriptor.setDescriptions(
        MyAdapter.getInstance().getAdapterResourceBundleManager(),l);
}
public void runNotification() throws ResourceException
{
    File thisDir = new File(_directory);
    File [] newList = thisDir.listFiles();
    ArrayList scratchCopy = new ArrayList(this._fileList);
    for (int nlIndex = 0;nlIndex < newList.length;nlIndex++)
    {
        String name = newList[nlIndex].getName();
        if(newList[nlIndex].isFile())
        {
            if(scratchCopy.contains(name))
            {
                scratchCopy.remove(name);
            }
            else
            {
                this._fileList.add(name);
                if(this._checkAdded)
                {
                    this.doNotify(createNotice(name,_directory,true,false));
                }
            }
        }
        else
        {
            scratchCopy.remove(name);
        }
    }
    // now anything left in the scratch copy is missing from the directory
    String[] deadList = new String[scratchCopy.size()];
    scratchCopy.toArray(deadList);
    for(int dlIndex = 0; dlIndex < deadList.length;dlIndex++)
    {

```

```

        this._fileList.remove(deadList[dlIndex]);
        if(this._checkDeleted)
        {
            this.doNotify(createNotice(deadList[dlIndex], _directory,false,true));
        }
    }
}
public Boolean adapterCheckValue(WmManagedConnection connection,
String resourceDomainName, String[][] values, String testValue)
throws AdapterException
{
    boolean result = true;
    if(resourceDomainName.equals(DIRECTORIES_RD))
    {
        File testDir = new File(testValue);
        if (!testDir.exists())
        {
            result = false;
        }
        else if(!testDir.isDirectory())
        {
            result = false;
        }
    }
    return new Boolean(result);
}
public ResourceDomainValues[] adapterResourceDomainLookup(
WmManagedConnection connection, String resourceDomainName,
String[][] values) throws AdapterException
{
    ResourceDomainValues[] results = null;
    if (resourceDomainName.equals(FIELD_NAMES_RD) ||
        resourceDomainName.equals(FIELD_TYPES_RD))
    {
        ResourceDomainValues names =
            new ResourceDomainValues(FIELD_NAMES_RD,new String[] {
                "FileName", "Path","isAdded","isDeleted"});
        ResourceDomainValues types =
            new ResourceDomainValues(FIELD_TYPES_RD,new String[] {
                "java.lang.String", "java.lang.String",
                "java.lang.Boolean","java.lang.Boolean"});
        results = new ResourceDomainValues[] {names,types};
    }
    return results;
}
public void registerResourceDomain( WmManagedConnection connection,
WmAdapterAccess access) throws AdapterException
{
    access.addResourceDomainLookup(this.getClass().getName(),
        FIELD_NAMES_RD,connection);
    access.addResourceDomainLookup(this.getClass().getName(),
        FIELD_TYPES_RD,connection);
    ResourceDomainValues rd = new ResourceDomainValues(DIRECTORIES_RD,
        new String[] {""});
    rd.setComplete(false);
    rd.setCanValidate(true);
    access.addResourceDomain(rd);
    access.addCheckValue(DIRECTORIES_RD, connection);
}
private WmRecord createNotice(String file, String dir, boolean isAdded,

```

```

boolean isDeleted)
{
    WmRecord notice =
        WmRecordFactory.getFactory().createWmRecord("notUsed");
    notice.put("FileName", file);
    notice.put("Path", dir);
    notice.put("isAdded", new Boolean(isAdded));
    notice.put("isDeleted", new Boolean(isDeleted));
    return notice;
}
}

```

Defining a WmPollingNotification Implementation Class

1. Create a directory structure for the Java package for adapter polling notification implementation. For example: `com\mycompany\adapter\myadapter\notifications`. In the example, the Java package created is `com\wm\MyAdapter\notifications`.

Note:

You must create your Java package and classes in the `adapterPackageName\code\source` directory in the webMethods package you created using Designer.

2. Create a class by extending `com.wm.adk.notification.WmPollingNotification` base class.

In the example, the class created is *SimpleNotification*.

Important:

You can make a callback to the connection factory using `WmManagedConnection.getFactory`. If you do this, do not call the *set* methods on the connection factory which produces unpredictable results.

3. Implement the `runNotification` method.
4. Override the base class implementation of the `fillWmTemplateDescriptor` method.

Specifying Configuration Metadata for Polling Notifications

The next step for implementing a polling notification is to create the metadata constructs that the users of the adapter use for entering data when they create polling notification nodes. To do this, you perform the following:

- Create metadata parameters appropriate for the function of the polling notification. Each parameter has:
 - A variable to hold the configured values.
 - A String constant containing the name of the parameter.
 - An accessor method.

- A set of resource bundle entries with a localizable parameter name and description as shown in [“Updating the Resource Bundle” on page 136](#).

For more information on metadata parameters, see [“Metadata Model for Connection” on page 63](#).

- Describe presentation for those metadata parameters.
- Set the data entry rules for those metadata parameters.

The example implementation includes three metadata parameters that the users of the adapter use to create polling notification nodes. The following table describes the purpose of each of these parameters for data entry:

Parameter	Description
directory	Directory to monitor.
checkAdded	Indicates whether notifications must be sent when files are added to the specified directory.
checkDeleted	Indicates whether notifications must be sent when files are deleted from the specified directory.

Specifying the Display and Data Entry Attributes of the Data Entry Parameters

- After creating the parameters, specify the display and data entry attributes by calling various methods of the `WmTemplateDescriptor` interface from the service's `fillWmTemplateDescriptor` method.
- The example code places each data entry parameter into a single group (in display order) referenced by the `NOTIFICATION_SETUP_GROUP` constant. A constant instead of a string is used to name the group, because the same value is used in the resource bundle to specify a localizable group name.

Implementing Configuration Resource Domains for Polling Notifications

The next steps for implementing a polling notification are:

- Define and implement the resource domains required for the metadata parameters that you created.
- Identify the values on which those resource domains depend.

For each parameter that requires a resource domain to supply a value, or that requires a validity check for values supplied by the users of the adapter, you must perform the following:

1. In the `WmPollingNotification.fillWmTemplateDescriptor` method, call `WmTemplateDescriptor.setResourceDomain` method, passing the name of the parameter, the

name of the resource domain, and an array of the names of any parameters on which the resource domain depends.

2. In the `WmPollingNotification.registerResourceDomain` method, call `WmAdapterAccess.addResourceDomain(ResourceDomainValues)` method to register the resource domain support.
3. In the `WmPollingNotification.registerResourceDomain` method, implement code to populate the resource domain values and/or the adapter check values. For more information about adapter check values, see [“Adapter Check Value Callbacks” on page 91](#).
4. In the associated connection class's `WmManagedConnection.registerResourceDomain` method, call `WmAdapterAccess.addResourceDomainLookup` method to add the polling notification metadata parameters with resource domain lookup.

```
package com.wm.MyAdapter.connection;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.*;
import com.wm.adk.error.AdapterException;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.services.MockDbUpdate;
import com.wm.MyAdapter.notifications.SimpleNotification;
public class SimpleConnection extends WmManagedConnection {
    ..
    ..
    public void registerResourceDomain(WmAdapterAccess access)
        throws AdapterException
    {
        ..
        ..
        ..
        //Simple Polling Notification Registering Resource Domain
        access.addResourceDomainLookup(SimpleNotification.DIRECTORIES_RD, this);
        access.addResourceDomainLookup(SimpleNotification.FIELD_NAMES_RD, this);
        access.addResourceDomainLookup(SimpleNotification.FIELD_TYPES_RD, this);
    }
}
```

In this example, there are no preset values, but adapter's user supplied data for the directory parameter is validated to ensure that the directory exists.

Specifying Notification Signatures (Document Type)

After you implement the configuration logic for the polling notification, you implement the logic that defines the signature of the polling notification node. It is used to create a document type node that enables triggers to identify notification data in the node.

Note:

A polling notification only has an output signature.

To define an output signature, you create additional metadata parameters as follows:

Parameter	Description
<code>signature</code>	Used with the reserved signature resource domain.
<code>fieldName</code>	Dependency parameters in which you build the signature data. The relationship between these parameters is established in the <code>WmTemplateDescriptor</code> interface. For more information about the mechanics of signature construction, see “Adapter Service Node Signatures” on page 93.
<code>fieldType</code>	

Manipulating Adapter Notification Document Properties

With few exceptions, the document properties for an adapter's polling and listener notifications are managed and manipulated the same way as they are for an adapter service's signature. For the three template based features (signature wrapping, override connection name, and pass full pipeline), only the pass full pipeline feature applies to notification documents, and then, only for synchronous notifications. When the pass full pipeline option is enabled for a synchronous listener notification, then the notification can pass fields to the invoked service that are not defined in its request document, and receive fields from that service that are not defined in the notification's reply document. Template-based signature manipulation features have no other effect on notification documents.

Document field properties are managed in exactly the same way as the signature field properties as described earlier. In the case of notifications, the `setSignatureProperties` method is called `setDocumentProperties` method. For asynchronous notifications, only one `PipelineRecordProperties` argument exists.

Implementing Signature Resource Domains

The resource domain implementation for the signature parameters in this example is straightforward. The signature in this case is static, but it is implemented as a lookup, to facilitate maintenance of the class.

Implementing the `WmPollingNotification.runNotification` Method and Callbacks

The final task for implementing a polling notification is to add the `runNotification` method and any callback methods. This example implements some very basic logic as previously described. It relies on the fact that the object instance is reused between `runNotification` calls. This may not be a good technique if the `runNotification` call runs for a long time or if overlapping calls occur. A more robust model would probably use a persistent store instead of an instance variable to track the current directory snapshot.

Updating the Resource Bundle

Update the resource bundle with display names, and descriptions to make the polling notification more usable, as follows:


```

package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.notifications.SimpleNotification;
public class MyAdapterResource extends ListResourceBundle implements MyAdapterConstants{
    ..
    ..
    static final Object[][] _contents = {
        ..
        ..
        //Polling Notifications
        ,{SimpleNotification.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Simple Polling Notification"}
        ,{SimpleNotification.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Looks for file updates to a specified directory"}
        ,{SimpleNotification.NOTIFICATION_SETUP_GROUP + ADKGLOBAL.RESOURCEBUNDLEKEY_GROUP,

        "Simple Notification Settings"}
        ,{SimpleNotification.DIRECTORY_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Directory Path"}
        ,{SimpleNotification.DIRECTORY_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Directory to monitor"}
        ,{SimpleNotification.CHECK_ADDED_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Notify on Add"}
        ,{SimpleNotification.CHECK_ADDED_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Check if notification must be generated when file added"}
        ,{SimpleNotification.CHECK_DELETED_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Notify on Delete"}
        ,{SimpleNotification.CHECK_DELETED_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Check if notification must be generated when file deleted"}
    }
    protected Object[][] getContents() {
        // TODO Auto-generated method stub
        return _contents;
    }
}

```

Registering Polling Notifications in the Adapter

You must register each polling notification class in the `WmAdapter` implementation class. You do this by passing the class name to the `AdapterTypeInfo.addNotificationType` method in the `WmAdapter.fillAdapterTypeInfo` method in `WmAdapter` implementation class. In the example, the polling notification `SimpleNotification` class is registered in the `MyAdapter` adapter implementation class:

For example:

```

package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.notifications.*;
..
..
public class MyAdapter extends WmAdapter implements MyAdapterConstants {
    ..
    ..
    public void fillAdapterTypeInfo(AdapterTypeInfo info, Locale locale)
    {

```

```
..  
..  
    info.addNotificationType(SimpleNotification.class.getName());  
}  
}
```

Configuring and Testing Polling Notification Nodes

Now you are ready to configure a polling notification node as follows:

- Configuring Polling Notification Nodes
- Scheduling and Enabling Polling Notification Nodes
- Testing Polling Notification Nodes

Before you configure a polling notification node, ensure that you have configured a connection node as described in [“Configuring and Testing Connection Nodes” on page 71](#). This section provides procedures for:

Configuring Polling Notification Nodes

Perform the following procedure to configure a polling notification node. You can use Designer to configure a polling notification node.

➤ To configure a polling notification node

1. Start Designer.

Note:

Ensure that the Integration Server connected to the Designer is running.

2. Select a namespace node package where you want to create the polling notification node.
3. Create a folder in the selected package, and navigate to that folder in **Package Navigator**.
4. Select **File > New**.
5. Select **Adapter Notification** from the list of elements.
6. In the **Create a New Adapter Notification** screen, type a name for your polling notification in the **Element name** field, and click **Next**.
7. In the **Select Adapter Type** screen, select the name of your adapter, and click **Next**.
8. In the **Select a Template** screen, select an adapter notification template, and click **Next**.

9. In the **Select an Adapter Connection Alias** screen, select the appropriate adapter connection name, and click **Next**.
10. In the **Publish Document Name**, select **Finish**.
11. Select **File > Save**.

The adapter creates the notification node and a document named *notificationNamePublishDocument*. This document contains the data of the affected portion of the adapter resource such as a database row, and is used to inform Integration Server of the changes.
12. In the Adapter Notification Editor's **notificationNameSetting** tab, select a polling event and polling location to monitor. Configure the Adapter Notification Editor fields as applicable to your adapter resource. For information about using the Adapter Notification Editor, see the *webMethods Service Development Help* for your release.
13. Select **File > Save**.
14. Schedule and enable the notification at runtime.

Scheduling and Enabling Polling Notification Nodes

Before you can use a notification, you must schedule and enable it.

➤ To schedule and enable a polling notification node

1. Start Integration Server Administrator.
2. In **Adapters** screen, select the name of your adapter.
3. Select **Polling Notifications**.
4. In the **Polling Notifications** screen, use the following options to schedule and enable each polling notification:

Note:

If you use an XA-Transaction connection, you cannot enable a notification.

Option	Description/Action
Notification Name	Name of the notification.
Package Name	Name of the package for the notification.

Option	Description/Action
State	<p>Note: Before you can enable a polling notification, you must schedule it. Click on the icon in the Edit Schedule column to edit the schedule.</p> <p>Once you schedule a polling notification, you can select the option Enabled to enable, and Disabled to disable a polling notification. Click the current value in this field to change its value.</p> <p>Use the Edit Schedule icon to create a polling notification as described in step 5.</p>
Edit Schedule	<p>Note: You must disable a polling notification before you can edit it.</p> <p>Click on the icon in the Edit Schedule column to create or modify polling notification parameters.</p>
View Schedule	Click on the icon in the View Schedule column to review the scheduled parameters for the selected polling notification. Click Return to Notifications to go back to the main polling notification page.

5. To create or modify schedule parameters for the selected notification, click the **Edit Schedule** icon and set the following options:

Option	Description/Action
Interval (seconds)	Type the polling interval time in seconds.
Overlap	<p>This option determines when the scheduled interval time you set in the Interval field begins. Enable this option to allow for executions of the scheduled notification to overlap. With the Overlap option enabled, the next scheduled execution does not wait for the current execution to end.</p> <p>Note:</p> <ul style="list-style-type: none"> ■ If your notification requires the preservation of the notification ordering, do not enable this option. ■ This option is enabled for clustered notifications.
Immediate	Enable this option to start polling immediately.

6. Click **Save Schedule**.

Testing Polling Notification Nodes

Creating the Flow Service for the Polling Notification Node

Perform the following procedure to create the flow service for the Polling notification node.

➤ **To create the flow service for the Polling notification node**

1. Start Designer.
2. In the Package Navigator, select the folder where you want to create the flow service.
3. Select **File > New**.
4. Select **Flow Service** from the list of elements.
5. In the **Create a New Flow Service** screen, type *TestMyAdapterFlowService* in the **Element name** field, and click **Next**.
6. On the **Select the Source Type** screen, select **Empty Flow**, and click **Finish**.
7. Click ➤ to insert a flow step.
8. Navigate to the **pub.flow:savePipelineToFile** service in the **WmPublic** package, and click **OK**.

Note:

The `savePipelineToFile` service saves the contents of the pipeline (from the polling notification event) to the file that you specify in the **fileName** parameter.

9. Click the **Pipeline** tab.
10. Open the **fileName** parameter in the pipeline and set its value to *MonitorPollingNotificationPipeline.log*.



Click **OK**.

Creating the Trigger for the Polling Notification Node

Perform the following procedure to create the trigger for the Polling notification node.

➤ **To create the trigger for the Polling notification node**

1. Start Designer.
2. In the Package Navigator, select the folder where you want to create the trigger.
3. Select **File > New**.
4. Select **webMethods Messaging Trigger** from the list of elements.

5. On the **webMethods Messaging Trigger** screen, type *TestMyAdapterMsgTrigger* in the **Element name** field and click **Finish**.
6. In the trigger editor, in the Conditions section, accept the default **Condition1**.
7. In the Condition detail section, in the **Service** field, select or type the flow service name *TestMyAdapterFlowService*.
8. Click  to insert document types. Select **TestMyPollingNotificationPublishDocument** and click **OK**.
9. Click  to save your trigger.

Scheduling and Enabling the Polling Notification Node

Perform the following procedure to schedule and enable the Polling notification node.

> To schedule and enable the Polling notification node

1. Start Integration Server Administrator.
2. In **Adapters** screen, select the name of your adapter.
3. Select **Polling Notifications**.
4. Click **Edit Schedule**.
5. Set the **Interval** to 10 and click the **Save Schedule** button.
6. Enable the node by selecting **Enabled** in the **State** column.

Testing the Polling Notification Node

Perform the following procedure to test the polling notification node.

> To test the Polling notification node

1. Add a file to the directory that is monitored. In this example, the directory is *C:\Monitor* and file added is *Testing-1.txt*.

The file *MonitorPollingNotificationPipeline.log* is created in *Integration Server_directory / instances/<instance_name>/pipeline*. This file contains the following entry for the file added in *C:\Monitor*:

```
<?xml version="1.0" encoding="UTF-8"?>  
<IDataXMLCoder version="1.0">
```

```

<record javaclass="com.wm.data.ISMemDataImpl">
  <value name="fileName">MonitorPollingNotificationPipeline.log</value>
  <record name="TestMyAdapter:TestMyAdapterNotificationPublishDocument"
javaclass="com.wm.data.ISMemDataImpl">
    <value name="FileName">Testing-1.txt</value>
    <value name="Path">C:\Monitor</value>
    <jboolean name="isAdded">>true</jboolean>
    <jboolean name="isDeleted">>false</jboolean>
    <record name="_env" javaclass="com.wm.data.ISMemDataImpl">
      <value name="locale"></value>
      <value name="activation">wm6bfd23a95-1c84-4e3e-b687-5858453777bc</value>
      <value name="businessContext">wm6:bfd23a95-1c84-4e3e-b687-5858453777bc\snul\
snul:wm6bfd23a95-1c84-4e3e-b687-5858453777bc:null:IS_61:null</value>
      <value name="uuid">wm:c3fb4d30-2f23-11ec-8723-000000000002</value>
      <value name="trackId">wm:c3fb4d30-2f23-11ec-8723-000000000002</value>
      <value name="pubId">islocalpubid</value>
      <Date name="enqueueTime" type="java.util.Date">Sun Oct 17 13:55:20 IST
2021</Date>
      <Date name="recvTime" type="java.util.Date">Sun Oct 17 13:55:20 IST 2021</Date>
      <number name="age" type="java.lang.Integer">0</number>
    </record>
  </record>
</record>
</IDataXMLCoder>

```

Cluster Support for Polling Notifications

When Integration Servers are deployed in a cluster, the servers in that cluster automatically share information about the registered polling notifications. The adapter runtime automatically coordinates polling and some callbacks among instances of the same polling notification node on different servers in a cluster.

This coordination of clustered polling notifications requires no special coding by the adapter developers. However, there are design considerations for adapters that is used in a cluster. There are also global, adapter-specific, and node-specific configuration options that control how coordination is performed. At a minimum, adapter developers must specify configuration values that are appropriate for the adapter.

Callback Coordination

When a callback is coordinated across the cluster, that callback is only executed on one instance of the polling notification in the cluster. For example, if the `enableCallBack` is coordinated, the first polling notification in the cluster that is enabled, executes the `enableCallBack`. When subsequent instances of that notification are enabled, the `enableCallBack` call is suppressed. If the callbacks were not coordinated, each instance executes the `enableCallBack` when the node instance is enabled.

The purpose of callback coordination is to prevent redundant updates to the backend associated with starting or stopping a notification. For example, when the `disableCallBack` is called on a polling notification for the Adapter for JDBC, a database trigger that gathers information for the notification is removed. Without coordination, all instances of that notification would be effectively disabled as soon as the first instance is disabled. With coordination, the `disableCallBack` is not executed until the last instance of that notification in the cluster is disabled.

From a design standpoint, it is important to segregate management of resources on the backend in separate callbacks from management of resources that are local to the notification instance. Callback coordination is configured so that related pairs of callbacks are either coordinated, or not. The following pairs of callbacks may be coordinated:

Coordinated Callback Pairs	Description
enableCallback and disableCallback	This occurs when the persistent node state is changed from <code>disabled</code> or changed to <code>disabled</code> . When coordinated, the first instance to go from <code>disabled</code> to <code>enabled</code> executes the <code>enableCallback</code> method and the last instance to go from either <code>suspended</code> or <code>enabled</code> to <code>disabled</code> executes the <code>disableCallback</code> method.
startupCallback and shutdownCallback	This occurs when the node becomes active or inactive for any reason, including when the node is enabled, disabled, suspended, or resumed (if the node is enabled, it also includes Integration Server and package startup or shutdown). When coordinated, the first instance to become active will execute the <code>startupCallback</code> and the last instance to become inactive executes the <code>shutdownCallback</code> .
suspendCallback and resumeCallback	This occurs when the node is suspended or resumed. When coordinated, the first instance to go from <code>enabled</code> to <code>suspended</code> executes the <code>suspendCallback</code> and the last instance to go from either <code>suspended</code> to <code>enabled</code> executes the <code>resumeCallback</code> .

For information on how to configure callback coordination, see [“Configuration Settings” on page 146](#).

Polling Coordination

When a polling notification is started, it executes polls in an interval according to its schedule configuration. When polling is coordinated across a cluster, a poll is executed on only one of the active instances at each scheduled interval. Which instance executes depends on the configuration and timing.

Integration Server supports a coordination mode configuration setting for each polling notification node. This configuration is normally set from Integration Server Administrator on the same page where the schedule is set. The following coordination mode values are supported:

Coordination Mode Setting	Description
Disabled	Disables all cluster coordination for this node (both polling and callback coordination). Instances of this node act independently without considering the cluster.
Standby	The first instance of this notification to start executes all polls until that instance either shuts down or fails. When that occurs, the first active

Coordination Mode Setting	Description
---------------------------	-------------

	notification instance that detects that the original instance is no longer polling take its place.
--	--

Distributed	Behaves much like standby, except that at the end of each interval, the instance that first detects that the time to poll has arrived executes the poll. When the Integration Server clocks are properly synchronized, generally the server with the lightest load executes the poll.
--------------------	---

The coordination information for clustered polling notifications is stored in the cluster's shared cache.

All coordination of clustered polling notifications is done through an entry specific to the node in the cluster's shared cache. When the first instance of a notification (with coordination enabled) is introduced into a cluster, a new shared cache session is created and populated with the name of the server hosting the node, and the state of the node. As that node is copied to other servers in the cluster, each instance is registered in the same shared cache session. The states of these respective instances are used to determine when coordinated callbacks must be executed.

Important:

Always copy polling notification nodes instead of creating new nodes with the same name and configuration. Each polling notification node is created with a GUID that forms part of the message ID of all documents published by the notification. If the instances do not have the same GUID, it can interfere with duplicate-message detection facilities.

When the first instance of a clustered polling notification is started, that instance is marked as "primary", its schedule and coordination settings are recorded, and the time calculated for the next poll are all recorded in the shared cache session. Being "primary" means that first instance executes the first poll (barring failures). When the "primary" schedules the next poll, it releases the "primary" status if coordination is distributed, or retains it if coordination is configured for standby mode.

When another instance of a clustered polling notification starts, that instance first detects that a previous instance is already started. Since this instance is not the first, it overwrites its own cluster and schedule settings with those recorded in the shared cache. This instance then schedules itself to "wake up" at the next scheduled poll time. When it wakes up, if another instance is marked as "primary", the notification instance will verify that the indicated instance is still active, then reschedule itself to wake up periodically until it detects that the poll was completed within the configured time limit. It then reschedules itself to wake up at the next scheduled poll and repeats the process. If the polling time arrives and no instance is marked as "primary", or it detects that the "primary" instance is no longer functioning, then the local instance assumes the "primary" role and executes the poll as described above. Since all coordination is based on timestamps recorded in the shared cache, it is very important for server clocks to be synchronized.

Configuration Settings

Cluster coordination is controlled by a number of configuration settings to control behavior and tune failure detection using timeouts.

Global Settings

The following parameters are set in the `server.cnf` file and apply globally to all clustered notifications for all adapters:

Parameter Name	Description
<code>watt.art.clusteredPollingNotification.keepAliveInterval</code>	Interval in milliseconds, with which a secondary instance will check to see if an executing instance is still alive. If not set, the secondary instance will change to the default <code>maxLockDuration</code> value of 180000 for the shared cache.
<code>watt.art.clusteredPollingNotification.keepAliveExpireTimeout</code>	Time in milliseconds, that an executing node can be late before it is assumed to have failed. In general, this setting must be equal to the amount of drift anticipated on the server clocks. If not set, the secondary instance will change to the default <code>maxLockDuration</code> value of 180000 for the shared cache.

Note:

The parameters can also be set using Integration Server Administrator. For more information, see *webMethods Integration Server Administrator's Guide*.

Adapter-Specific Settings

In the configuration directory of the adapter's package, the `clusterProperties.cnf` 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. Template-specific settings use the template class name to set the scope of the setting. The `clusterProperties.cnf` file is located in the `Integration Server_directory \instances\instance_name\packages\AdapterName\config` folder.

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.LatchedPollingNotification">
      <callbackScheme>1</callbackScheme>
      <runtimeModeLimit>standby</runtimeModeLimit>
    </template>
  </pollingNotifications>
</clusterProps>
```

```

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

```

The outer `<clusterProps>` wrapper contains the following three elements:

- `<pollingNotifications>` wraps settings for clustered polling notifications. The `<pollingNotifications>` wrapper must contain the following global settings that provide the adapter's default values:
 - A `<callbackScheme>`
 - A `<runtimeModeLimit>`

These can be followed by any number of template wrappers, which contain the `<callbackScheme>` and `<runtimeModeLimit>` settings specific to each template. Template specific settings are applied to notification nodes created from the associated template name.

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.

Values for `callbackScheme` Setting

Coordination Modes			
<code>callbackScheme</code>	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

Values for `runtimeModeLimit` Setting

<code>runtimeModeLimit</code>	Result
disable	Nodes created using this template cannot be coordinated across a cluster. Nodes are forced into the <code>disabled</code> coordination mode.

runtimeModelLimit Result

standby . Default	Nodes can be coordinated in <code>standby</code> mode or coordination may be disabled.
distribute	Nodes can be coordinated in <code>distributed</code> or <code>standby</code> mode, or coordination may be disabled.

When a polling notification is created or registered on a cluster-aware Integration Server, the Integration Server looks for a `clusterProperties.cnf` file in the adapter's config directory. If the file contains no entry for the notification's template, a new `<template>` entry is created using the settings specified globally for all polling notifications. If the file is completely absent or unreadable, a new file is created using the default settings identified earlier.

- `<listenerNotifications>` is for future use. Copy wrapper and contents from the example.
- `<listeners>` is for future use. Copy wrapper and contents from the example.

Node-Specific Settings

The polling notification schedule page in Integration Server Administrator includes cluster settings that are only editable in a clustered environment. On this page, the coordination mode may be set to one of the values supported by its template. For more information, see `runtimeModelLimit` earlier. In addition, timeouts may be separately configured for polling and setup callback operations.

7 Listener Notifications

■ Overview	150
■ Listener Classes	151
■ Asynchronous Listener Notification Classes	153
■ Synchronous Listener Notification Classes	154
■ Listener and Listener Notification Interactions	156
■ Listener Implementation	159
■ Listener Notification Implementation	167
■ Configuring and Testing Listener Nodes and Listener Notification Nodes	178

Overview

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

When the user of the adapter enables the polling notification node, the following events occur:

- Integration Server instantiates and initializes a polling notification object with settings from the node.
- Integration Server then invokes the polling notification object by calling its `runNotification` method on a periodic basis.
- The polling notification object must retrieve a connection and use it to determine if publishable events have occurred in the adapter resource.
- If an event has occurred, the polling notification object must generate a `WmRecord` object conforming to the output signature of the polling notification node, and publish it by calling the `doNotify` method.
- The polling notification object then goes to sleep until the next time it is invoked.
- If the `overlap` option is enabled in the notification schedule, a second object may be instantiated while the previous instance is still processing events detected in a previous invocation. The second instance retrieves another connection, and interrogates the resource again. This model can make state management significantly more difficult.

When the user of the adapter uses a listener notification, the responsibility of 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 the user of the adapter enables 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 Integration Server.
- Integration Server interrogates a configured list of listener notifications associated with the listener node until it finds a listener notification node that can process the event. Integration Server accomplishes this by calling the listener notification's `supports` method. The first notification to return `true` from this call is invoked using its `runNotification` method. This behavior is similar to the polling notification where any data about the event that was provided by the listener is passed as an argument to the `runNotification` method.

Synchronous and Asynchronous Listener Notifications

The ADK includes both, synchronous and asynchronous processing model.

An asynchronous listener notification publishes a document to a configured webMethods Broker, using the `doNotify` method.

- An asynchronous listener notification publishes a document.
- The notification object's `runNotification` method calls `doNotify` method.
- The users of the adapter may process the document's data as needed. For example, the users of the adapter can create an Integration Server trigger that receives the document and executes an Integration Server flow service or a Java service.
- Asynchronous listener notifications do not support session handling. When a synchronous listener notification calls an Integration Server service that needs information contained in the session data, it may appear to work 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.

A synchronous listener notification invokes a specified Integration Server service, and potentially receives a reply from the service and delivers the results back to the adapter resource.

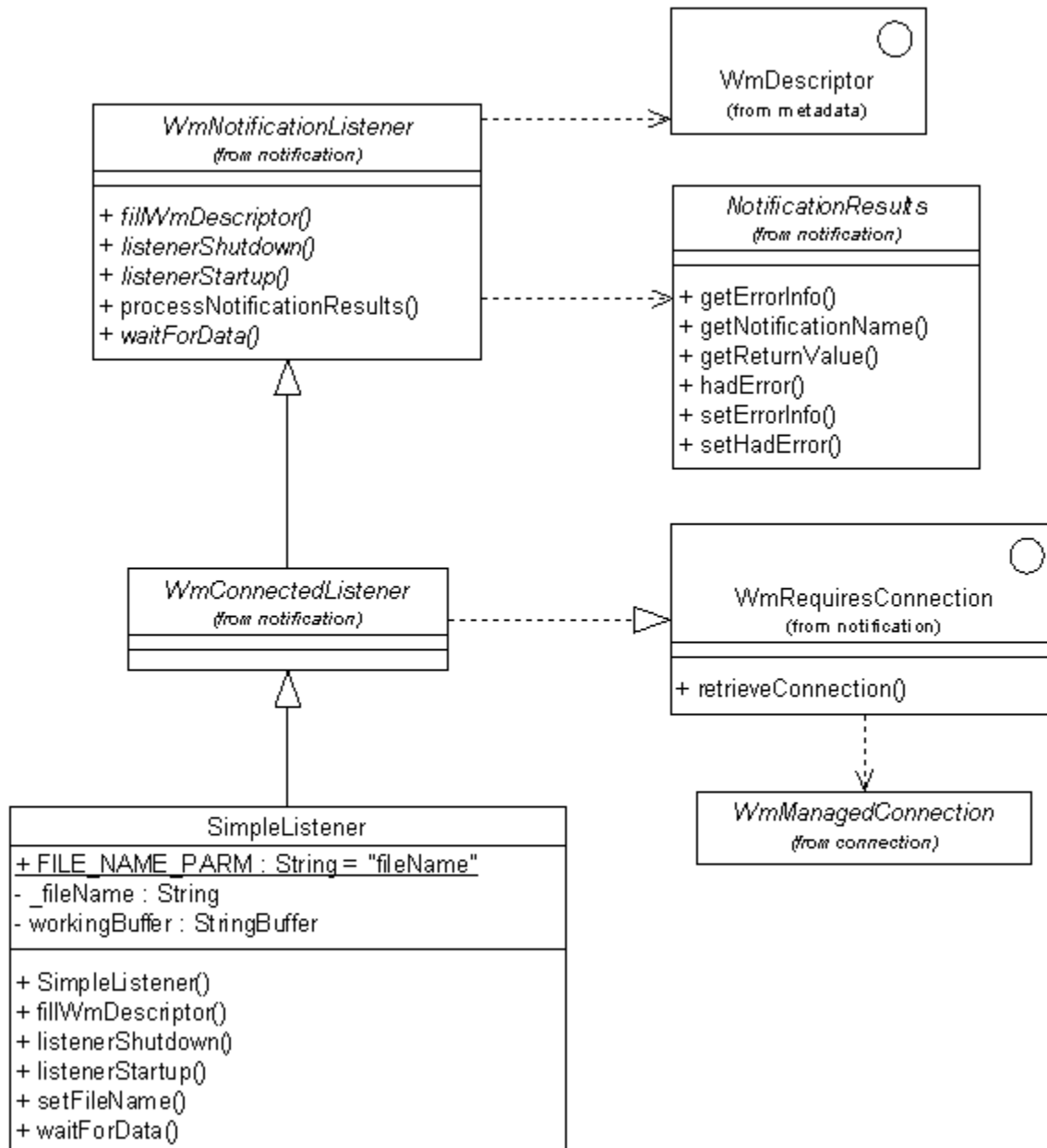
- A synchronous listener notification does not publish a document.
- The notification object's `runNotification` method calls `invokeService`.
- The users of the adapter cannot process the document's data as the processing is done in the `invokeService` method.
- Synchronous listener notifications do not support session handling. When a synchronous listener notification calls an Integration Server service that needs information contained in the session data, that service can fail.

Implementing Listeners and Listener Notifications

The implementation of a listener and listener notification is similar to the implementation of an adapter connection. Each implementation includes a Java class extending an ADK base class, and a namespace node in which design-time configuration data is stored. In the Java class, the metadata model for listeners and listener notification is nearly identical to that of adapter connections. The configuration pages are built from the listeners metadata. For more information, see [“Metadata Model for Connection” on page 63](#)

Listener Classes

The following figure shows the classes provided by the ADK to support listener notifications. It also shows the `com.wm.adk.notification.WmConnectedListener` implementation class *SimpleListener*.



Create a listener class by extending `com.wm.adk.notification.WmConnectedListener` base class. You must override the following base class methods in your `WmConnectedListener` implementation class:

Method	Description
<code>fillWmDescriptor</code>	Controls how metadata parameters are displayed, and defines rules for data entry for the listener's metadata parameters. From the standpoint of the adapter implementation, the model is identical to the connection model.
<code>listenerStartup</code>	Initializes the listener. This method is called during the listener startup sequence as well as during the recovery procedure after an <code>AdapterConnectionException</code> is encountered.

Method	Description
waitForData	Monitors the adapter resource. This method returns data that is analyzed by the support method of associated listener notifications. For more details, see “Listener and Listener Notification Interactions” on page 156.
listenerShutdown	Cleans up listener resources. This method is called during the listener shutdown sequence.

Note:
You may optionally override the processNotificationResults method to allow the listener implementation class to post-process listener notification results. For an example of using both methods, see [“Implementing Listener Methods”](#) on page 164.

In addition, the implementation class may override the following optional methods:

Method	Description
restrictNotificationTypes	Allows the listener implementation class to restrict the notification classes it supports by explicitly identifying them. For more information, see “Restricting Listeners to Register Specified Notification Templates” on page 163.
shutdownCallBack	Invoked on a thread separate from the listener's thread, this method allows the listener's waitForData loop to be gracefully interrupted prior to a normal shutdown.

Asynchronous Listener Notification Classes

The following figure shows the classes provided by the ADK to support asynchronous listener notifications.

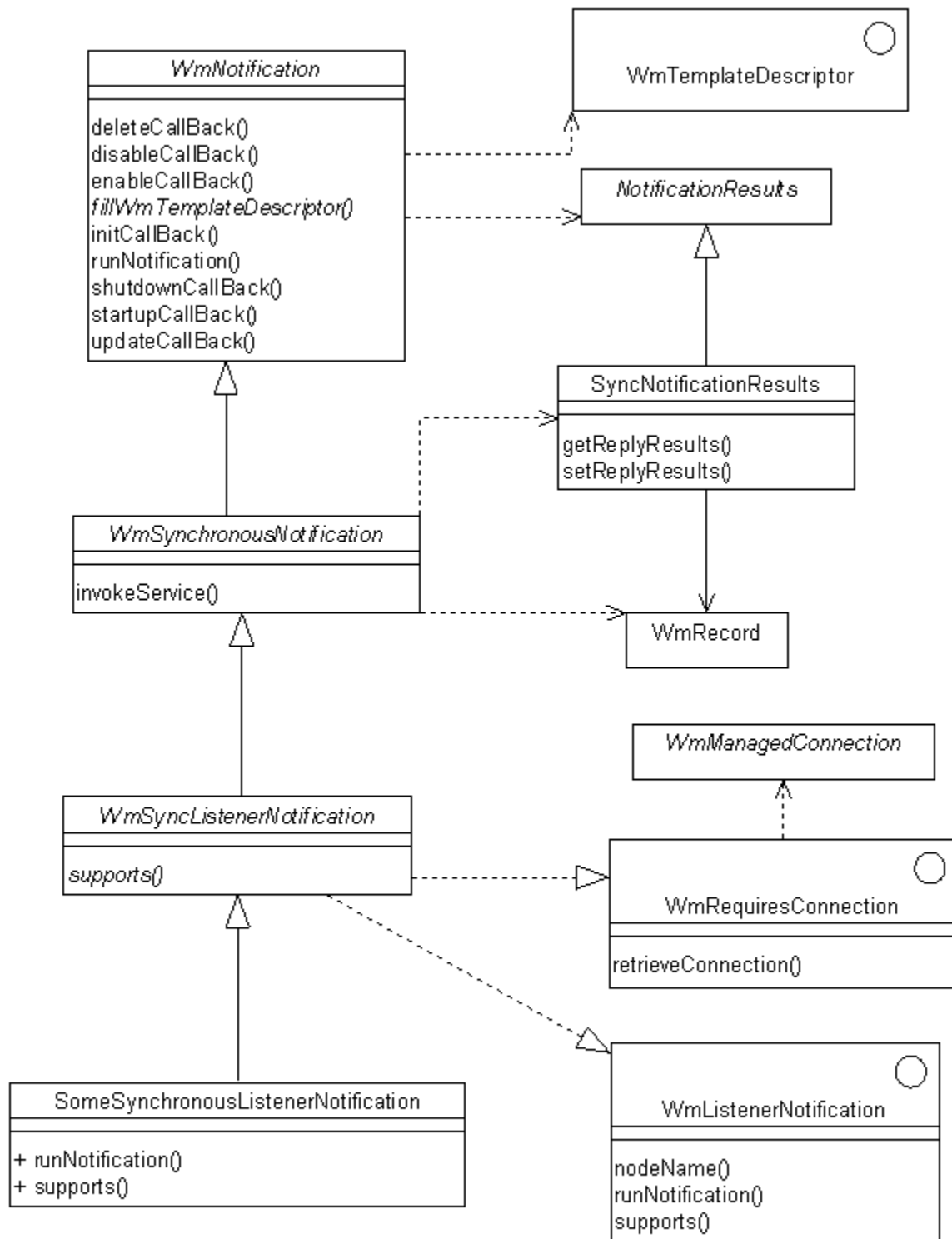
Create an asynchronous listener notification class by extending the base class `com.wm.adk.notification.WmAsyncListenerNotification`, as shown by the `SessionLogListenerNotification` class in the figure. As previously mentioned, the implementation of an asynchronous listener notification is similar to a polling notification. The key differences occur in the following methods:

Method	Description
<code>supports</code>	Determines whether the notification can process the object returned by the listener's <code>waitForData</code> method. <pre>boolean supports(Object)</pre>
<code>runNotification</code>	If the <code>supports</code> method returns <code>true</code> , the Integration Server calls the following <code>runNotification</code> method to process the data. <pre>NotificationResults runNotification(NotificationEvent)</pre> <p>The implementation of this method must</p> <ul style="list-style-type: none">■ Call one of the <code>WmAsynchronousNotification.doNotify</code> methods for each notification it wants to generate, based on the <code>NotificationEvent</code> data content.■ Never return <code>null</code>. <p>Note: The <code>NotificationEvent</code> simply wraps the data object returned from the listener's <code>waitForData</code> method.</p>

For more information about alternative method of managing resource domains, see .

Synchronous Listener Notification Classes

The following figure shows the classes provided by the ADK to support synchronous listener notifications.



Create a synchronous listener notification class by extending `com.wm.adk.notification.WmSyncListenerNotification` base class. The examples in this section do not implement a synchronous notification, so the figure in this section only shows a placeholder class.

A synchronous notification calls the `WmSynchronousNotification.invokeService` method, passing it a `WmRecord` object containing data consistent with the output signature of the notification node.

Synchronous listener notifications are expected to define both an input and output signature. The terms input and output are relative to the notification. The output signature specifies the format of the data that the notification places on the pipeline prior to invoking the service associated with the synchronous notification. The input signature describes the data that the notification expects to find on the pipeline after the invoked service has completed processing.

The service invoked by `WmSynchronousNotification.invokeService` method is specified at design time in the notification node data. When this service is invoked, it executes on a separate thread (and therefore in a different transactional context) from the listener. When the invoked service completes, Integration Server extracts the data on the pipeline that is identified in the notification node's input signature and delivers that data as a `WmRecord` wrapped in the `SyncNotificationResults` object returned by `invokeService` method.

Listener and Listener Notification Interactions

The design time interactions for listeners and listener notifications are essentially the same as the interactions for connections and polling notifications, as described in [“Connection Class Interactions” on page 67](#) and [“Polling Notification Interactions” on page 126](#), respectively. This section describes the runtime behavior of listeners and their associated notifications.

Important:

A listener and the associated listener notifications must reside in the same package in the `Integration Server_directory /instances/<instance_name>/packages/<your_package>` folder. Otherwise, data is lost when a package is reloaded.

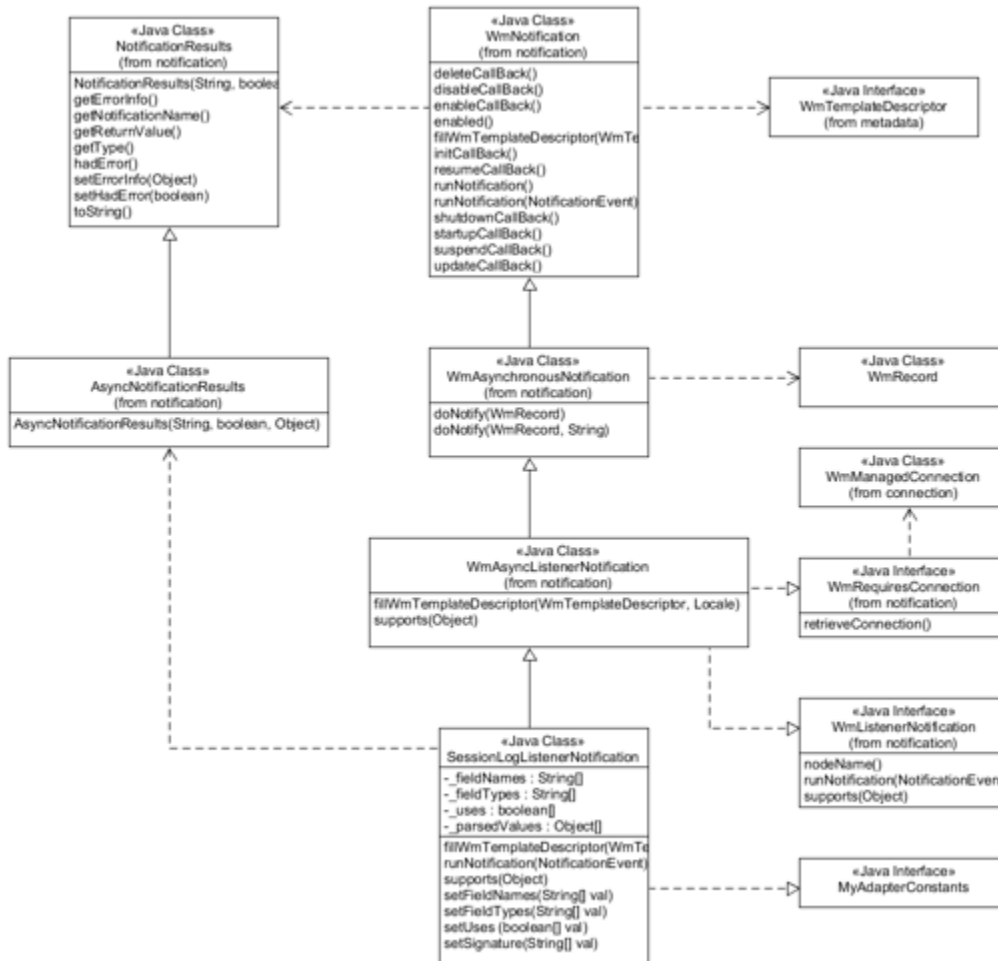
The figures that illustrate listener runtime interactions with synchronous and asynchronous notifications in this section show the runtime lifecycle of a listener from the time it is started until the time it is shut down. The difference between the two diagrams begins at step 3.3, when the server calls the `runNotification` method of the notification. Do not interpret the separation of these diagrams to imply that a given listener can use only synchronous or asynchronous notifications. On the contrary, step 3 represents a loop that repeats continuously while the listener is running, and any iteration of the loop may follow either course depending on the class type of the notification that indicates support for the notification event.

When a listener starts (or restarts), Integration Server performs the following:

- Retrieves a connection from the associated connection node.
- Calls the listener's `listenerStartup` method. This `listenerStartup` method performs the following:
 - Access the connection using the `retrieveConnection` method from the base class.
 - Initialization required prior to the first `waitForData` call.
- Disables the listener if an exception is thrown by the `listenerStartup` method, or if retrieving connection by Integration Server fails.

Note:

A listener instance holds the connection it retrieves during listener initialization for the lifetime of the listener instance. Disabling the connection node has no impact on this connection already held by the listener, but prevents the listener from starting or restarting in the event of an `AdapterConnectionException`.



After initialization of the listener is complete, Integration Server initiates the notification event-processing loop represented by step 3 of the interaction diagrams. This loop continues until one of the following events occurs:

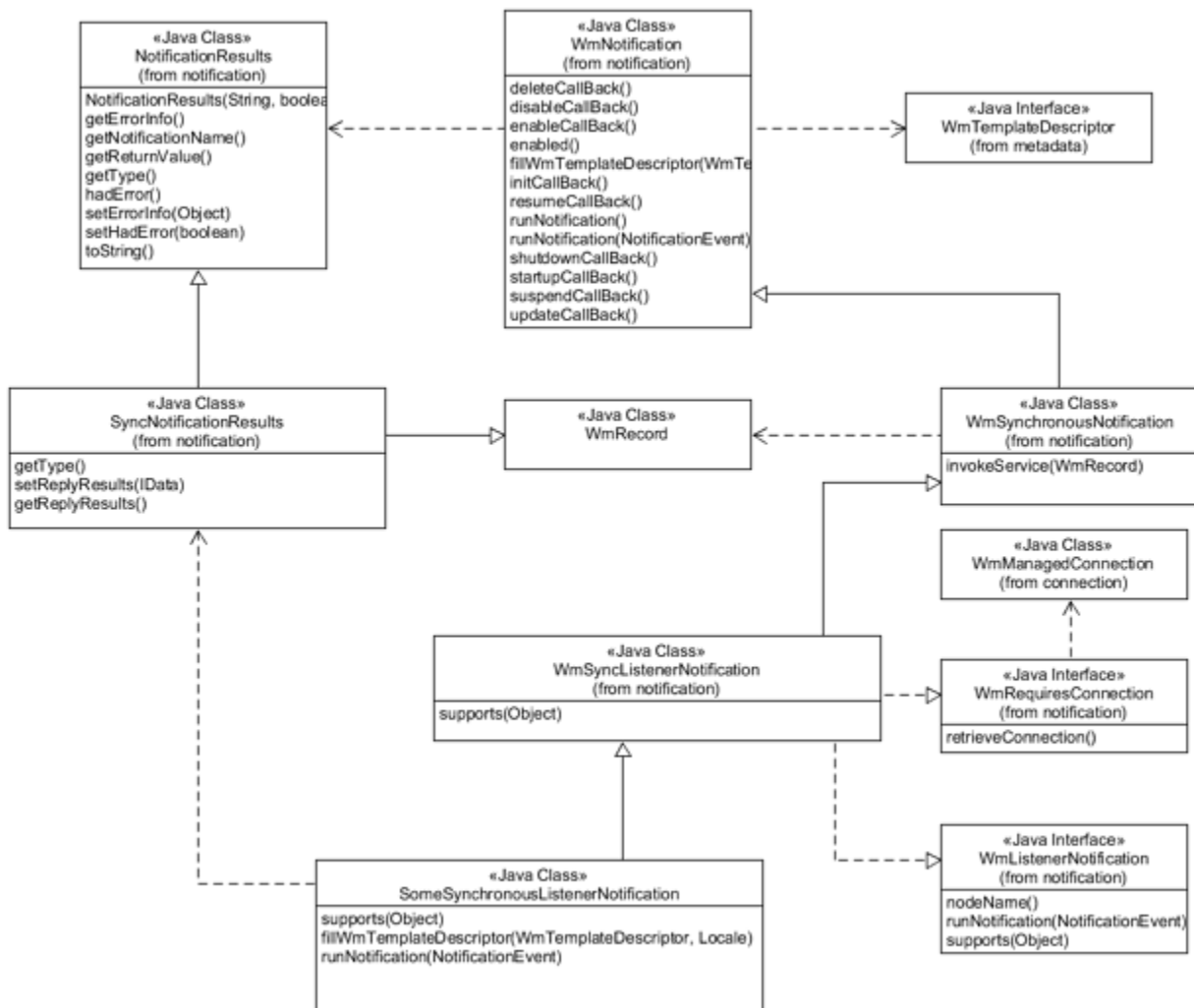
- The user of the adapter disables the listener in the adapter's administrative interface.
- Package containing the listener is disabled (or reloaded).

Note:

Disable the listener package before you disable the adapter package.

- Integration Server is shut down.
- Listener or a listener notification throws an `AdapterConnectionException`. This causes the listener to shut down and to attempt to restart with a new connection.
- Listener throws an `AdapterException` or a `RunTimeException`.

The following figure shows the listener runtime interactions with synchronous notification



The event-processing loop begins with a call to the listener's `waitForData` method. This method performs the following:

- Interrogates the adapter resource to determine whether an event has occurred that the listener should report.
- If the event has not occurred, the listener should return a `null` object.

The model assumes that the listener implements some form of blocking read operation with a time component that allows it to return periodically, even if no notification event has occurred. The adapter developer must provide appropriate means of configuring the timing characteristics of this blocked read, such as through a metadata parameter on either the listener or the connection.

- If the event has occurred, the listener's `waitForData` method returns a non-null object, Integration Server iterates through the listener notifications that are currently registered with the listener, which must be enabled and registered.

- The data object received from `waitForData` method is passed to the `supports` method (step 3.2) for each notification.
- For the first notification that returns `true` from its `supports` method:
 - Integration Server calls the `runNotification` method passing the data object wrapped as a `NotificationEvent` (step 3.3).
 - Integration Server calls the listener's `processNotificationResults` method with a `NotificationResults` as input object returned from `runNotification` (step 3.4).
- If no notification returned `true` from the `supports` method, or if `runNotification` threw an exception other than an `AdapterConnectionException`, then `processNotificationResults` is called with a `null` argument.

The listener notification's `runNotification` method (step 3.3) is responsible for the following:

- Interpreting the `NotificationEvent` object.
- Building a `WmRecord` object consistent with its output signature.

This may or may not require additional interaction with the resource. If it does, a connection is made available for that purpose through the `retrieveConnection` method, which is the same connection used by the listener.
- An asynchronous notification passes this `WmRecord` object to the `doNotify` method (step 3.3.1).
- The `doNotify` method publishes the document in the same way it does for polling notifications.
- An asynchronous notification must instantiate a new `AsynchNotificationResults` object (step 3.3.3), including the notification name, which can be retrieved from the base class `nodeName` method (step 3.3.2).
- A synchronous notification's implementation of `runNotification` method passes its output, `WmRecord` object to `invokeService` method instead of `doNotify` method (step 3.3.1).
- The service called by `invokeService` method is specified in the configuration of the synchronous listener notification node. The service is invoked on a separate thread and transaction context.
- The results of the service invocation are returned to `invokeService` method as a `SynchronousNotificationResults` object.
- The results are then available to be interrogated by the notification and/or the listener, using the methods provided by that class.

Listener Implementation

The example listener implementation provided in this section shows the basic mechanics of a simple listener that can be used to monitor activity on an Integration Server log file. The example listener notification parses a session log entry and produces asynchronous notifications.

The tasks for implementing a listener are as follows:

- Defining a WmConnectedListener Implementation Class
- Specifying Configuration Metadata for Listeners
- Implementing Listener Methods
- Creating getReader Method in the WmManagedConnection Implementation Class
- Updating the Resource Bundle
- Registering Listeners in the WmAdapter Implementation Class
- Compiling the Adapter
- Reloading Adapter
- Refreshing the Designer cache
- Configuring a Listener

Example Listener Class

```
package com.wm.MyAdapter.listeners;
import com.wm.adk.error.AdapterException;
import com.wm.adk.metadata.WmDescriptor;
import com.wm.adk.notification.WmConnectedListener;
import com.wm.adk.notification.NotificationResults;
import java.io.FileReader;
import java.util.Locale;
import javax.resource.ResourceException;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.connections.SimpleConnection;
```

```
public class SimpleListener extends WmConnectedListener
{
    public static final String FILE_NAME_PARM = "fileName";
    private String _fileName = null;
    public void setFileName(String val){_fileName = val;}
    private FileReader _reader = null;
    private StringBuffer workingBuffer = new StringBuffer();
    private String _lastDataObject = null;
    public void fillWmDescriptor(WmDescriptor descriptor, Locale locale)
        throws ResourceException
    {
        descriptor.setRequired(FILE_NAME_PARM);
        descriptor.setDescriptions(
            MyAdapter.getInstance().getAdapterResourceBundleManager(), locale);
    }
}
```

```
public void listenerStartup() throws ResourceException
{
    try
    {
        //_reader = ((SimpleConnection)retrieveConnection()).getReader();
        _reader = ((SimpleConnection)retrieveConnection()).getReader(_fileName);
        while(_reader.ready())
        {
            _reader.read(); // move to the end of the stream
        }
    }
}
```



```

    }
  }
  catch (Throwable t)
  {
    throw MyAdapter.getInstance().createAdapterException(100,t);
  }
}
public Object waitForData() throws ResourceException
{
  try
  {
    if(_reader.ready())
    {
      do
      {
        int i = _reader.read();
        if (i != -1)
        {
          char c = (char)i;
          workingBuffer.append(c);
          if(c == '\n')
          {
            _lastDataObject = new String(workingBuffer);
            workingBuffer = new StringBuffer();
            break;
          }
        }
      }
      else
      {
        break;
      }
    } while (_reader.ready());
  }
  catch (Throwable t)
  {
    throw MyAdapter.getInstance().createAdapterException(100,t);
  }
  return _lastDataObject;
}

```

```

public void listenerShutdown()
{
  try
  {
    _reader.close();
  }
  catch(Throwable t){}
}
public void processNotificationResults(NotificationResults results)
throws ResourceException
{
  if(results != null)
  {
    if(results.hasError())
    {
      MyAdapter.getLogger().logError(9999,
        "Error processing: " + this._lastDataObject +
        " errorInfo = " + (results.getErrorInfo() == null ? "" :
results.getErrorInfo().toString()));
    }
  }
}

```

```
    }  
  }  
  else  
  {  
    MyAdapter.getLogger().logError( 9999,  
      "No notification available to process:" +  
      this._lastDataObject);  
  }  
}  
}
```

Defining a WmConnectedListener Implementation Class

1. Create a class by extending `com.wm.adk.notification.WmConnectedListener` base class. In this example, the class created is `SimpleListener`.
2. Add skeletal implementation of the abstract methods.
3. Implement the `shutdownCallBack` method.

The `shutdownCallBack` method is invoked on a thread separate from the listener's thread. This feature allows the listener's `waitForData` loop to be gracefully interrupted prior to a normal shutdown. The thread that initiates `listenerShutdown` method invokes `shutdownCallBack` method in the following situations:

- When the listener node is disabled.
- When the package containing the listener node is reloaded.
- When Integration Server shuts down.

The system passes to this method a reference to the underlying resource connection. The method is invoked prior to calling the listener's `listenerShutdown` method. It will not be called if the shutdown is due to an exception.

The signature of this method is:

```
public void shutdownCallBack(WmManagedConnection wmConn) throws  
                             ResourceException
```

The `listenerShutdown` method is subsequently invoked by WmART on the listener's thread. This is where you must perform any listener-specific cleanup tasks.

Specifying Configuration Metadata for Listener Notifications

The next step for implementing a listener is to create the metadata constructs that the users of the adapter use for entering data when they create listener notification nodes. To do this, perform the following:

- Create metadata parameters appropriate for the function of the listener notification nodes. Each parameter has:

- A variable to hold the configured values.
- A String constant containing the name of the parameter.
- An accessor method.
- A set of resource bundle entries with a localizable parameter name and description as shown in [“Updating the Resource Bundle” on page 136](#).

For more information on metadata parameters, see [“Metadata Model for Connection” on page 63](#).

- Describe presentation for those metadata parameters.
- Set the data entry rules for those metadata parameters.

The example implementation includes one metadata parameter, that the users of the adapter use to create listener notification nodes. The following table describes the purpose of each of these parameters for data entry:

Parameter Description	
<i>fileName</i>	File to monitor.

Parameter	Accessor Method	Variable	Resource Bundle Entries
<i>fileName</i>	setFileName	FILE_NAME_PARM	None

Specifying the Display and Data Entry Attributes of the Data Entry Parameters

- After creating the parameters, specify the display and data entry attributes by calling various methods of the `WmDescriptor` interface from the service's `fillWmDescriptor` method.
- The example code places each data entry parameter into a single group (in display order) referenced by the constant `NOTIFICATION_SETUP_GROUP`. A constant instead of a string is used to name the group, because the same value is used in the resource bundle to specify a localizable group name.

Restricting Listeners to Register Specified Notification Templates

By default a listener supports all listener notification templates of the adapter. When the users of the adapter configure a new listener notification (either synchronous or asynchronous) using Designer, they may select from the complete list of all enabled listeners in the adapter. You may want to override this default behavior so that the listener implementation specifies exactly which notification templates it supports. To do this, implement the `restrictNotificationTypes` method in your listener implementation class. This method returns a `String` array containing the fully qualified path names of one or more notification template classes that the listener supports. When the users of the adapter attempt to configure a new listener notification in Designer, only those template classes in the returned array will appear in the Adapter Notification Editor. The following example shows how to use this method.

```
public class SimpleListener extends WmConnectedListener
{
    ...
    public String[] restrictNotificationTypes()
    {
        return new String[]{"com.wm.myadapter.notifications.FooNotification",
                            "com.wm.myadapter.notifications.BarNotification"};
    }
    ...
}
```

Note:

All names returned by this method should refer to classes that extend WmNotification.

Implementing Listener Methods

Create the private class attributes and implement the following methods:

- listenerStartup
- waitForData
- listenerShutdown
- processNotificationResults

In the example, the class created is *SimpleListener*. Create the private class attributes and implement the abstract methods to manage a *FileReader* object and variables to hold data read from the log.

The example also includes an implementation of *processNotificationResults*.

Creating getReader Method in WmManagedConnection Implementation Class

Create the *getReader* method in the *WmManagedConnection* implementation class. This method is called from the *listenerStartup* method in *WmConnectedListener* implementation class.

```
package com.wm.MyAdapter.connections;
..
..
import java.io.FileReader;
import java.io.FileNotFoundException;
..
..
public class SimpleConnection extends WmManagedConnection {
..
..
    public FileReader getReader(String fileName) throws AdapterException
    {
        FileReader _reader = null;

        try {
            _reader = new FileReader(fileName);
        }
    }
}
```

```

catch(Exception e) {
    throw MyAdapter.getInstance().createAdapterException(100,e);
}
return _reader;
}
}

```

Updating the Resource Bundle

Update the resource bundle with a display name, and description to make the listener more usable. For example:

```

package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.listeners.SimpleListener;
..
..
public class MyAdapterResource extends ListResourceBundle implements MyAdapterConstants{
    ..
    ..
    static final Object[][] _contents = {
        ..
        ..
        //Listener
        ,{SimpleListener.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Simple Listener"}
        ,{SimpleListener.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Use to monitor log files"}
        ,{SimpleListener.FILE_NAME_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Log File Name"}
    }
    protected Object[][] getContents() {
        // TODO Auto-generated method stub
        return _contents;
    }
}
}

```

Registering Listeners in the WmAdapter Implementation Class

You must register each listener class in the `WmAdapter` implementation class. You do this by passing the class name to the `AdapterTypeInfo.addListenerType` method in the `WmAdapter.fillAdapterTypeInfo` method in the `WmAdapter` implementation class. In the example, the listener class `SimpleListener` is registered in the adapter implementation class `MyAdapter`:

```

package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.listeners.SimpleListener;
..
..
public class MyAdapter extends WmAdapter implements MyAdapterConstants {
    ..
    ..
    public void fillAdapterTypeInfo(AdapterTypeInfo info, Locale locale)
    {

```

```
..  
..  
    info.addListenerType(SimpleListener.class.getName());  
}  
}
```

Compiling Adapter

Compile your adapter as described in [“Compiling the Adapter”](#) on page 43.

Reloading Adapter

Reload your adapter as described in [“Loading, Reloading, and Unloading Packages”](#) on page 52.

Refreshing the Designer cache

Refresh the Designer cache.

Configuring a Listener

Perform the following procedure to configure a listener node. You can use Designer to configure adapter service nodes.

> To configure a listener node

1. Start Integration Server Administrator.
2. In **Adapters** screen, select the name of your adapter.
3. Select **Listeners**.
4. In the **Listeners** screen, click **Configure New Listener**.
5. In the **Listener Types** screen, select the appropriate listener type defined for the adapter.
6. Complete the **Configure Listener Type > Adapter_Name** section as follows:

Field	Description\Action
Package	Namespace node package in which you want to create the listener.
Folder Name	Name of the folder in which to create the listener.
Listener Name	Name of the listener node.
Connection Name	Connection node.

Field	Description\Action
Retry Limit	Number of times that the system must attempt to start the listener if the initial attempt fails. Specifically, it specifies how many times to retry the listenerStartup method before issuing an AdapterConnectionException. When the value is set to 0, the system makes a single attempt. The default value is 5.
Retry Backoff Timeout	Number of seconds the system must wait between each attempt to start the listener. This field is irrelevant if the value of Retry Limit is 0. The default value is 10.

7. Click **Save Listener**.

Note:

Enabling the listener before you configure and enable its corresponding listener notification node produces a warning.

Note:

A listener instance holds the connection it retrieves during listener initialization for the lifetime of the listener instance. Disabling the connection node will not impact the connection already held by the listener, but it will prevent the listener from starting or restarting in the event of an AdapterConnectionException.

Listener Notification Implementation

The following example implements an asynchronous listener notification that recognizes session log entries and generates notifications from them. To distinguish a session log entry from another type of log entry, the listener parses the entry data in the notification's supports method. The supports method model is flexible enough to permit this approach because the same listener notification object instance that returns true from the supports call is guaranteed to receive the runNotification call.

The example listener notification uses an alternative approach to implementing resource domains that redirects resource domain activities back to the notification (or adapter service) that uses it. This model does a better job of encapsulating the notification functionality into a single class. The model is described in detail in . If you do not want to use this model, you may implement the resource domain code following the model described in [“Specifying Adapter Service Signature Resource Domains” on page 116](#).

Another new concept used in this example is the "uses" metadata mechanism. This is a shortcut mechanism commonly used to manipulate a metadata signature. You use it to add a column of check boxes to the adapter's interface so that the users of the adapter can select the fields to use for the notification. When they select the check boxes, values will appear in the Signature column. This metadata mechanism is described in [“The useParam Argument of setResourceDomain” on page 171](#), but you do not need to fully understand the constraint of that feature to implement this example.

The tasks for implementing an asynchronous listener notification are as follows:

- Defining a WmAsyncListenerNotification Implementation Class
- Specifying Configuration Metadata for Listener Notifications
- Implementing Configuration Resource Domains for Listener Notifications
- Implementing the supports and runNotification Method
- Updating the Resource Bundle
- Registering Listener Notifications in the Adapter
- Compiling the Adapter
- Reloading Adapter
- Refreshing the Designer cache
- Configuring and Testing Listener Notification Nodes

Example Listener Notification Class

```
package com.wm.MyAdapter.listeners;
import com.wm.adk.error.AdapterException;
import com.wm.adk.metadata.WmDescriptor;
import com.wm.adk.notification.WmAsyncListenerNotification;
import com.wm.adk.notification.NotificationResults;
import com.wm.adk.notification.AsyncNotificationResults;
import com.wm.adk.notification.NotificationEvent;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.*;
import com.wm.adk.cci.record.WmRecord;
import com.wm.adk.cci.record.WmRecordFactory;
import javax.resource.ResourceException;
import java.util.Locale;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.StringTokenizer;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.MyAdapterConstants;
import com.wm.MyAdapter.connections.SimpleConnection;
```

```
public class SessionLogListenerNotification extends WmAsyncListenerNotification
    implements MyAdapterConstants
{
    private String[] _fieldNames = null;
    private String[] _fieldTypes = null;
    private boolean[] _uses = null;
    public void setFieldNames(String[] val){_fieldNames = val;}
    public void setFieldTypes(String[] val){_fieldTypes = val;}
    public void setUses (boolean[] val){_uses = val;}
    public void setSignature(String[] val){}
    public static final String NOTIFICATION_SETUP_GROUP =
        "SessionLogListenerNotification.setup";
    public static final String FIELD_NAMES_PARM = "fieldNames";
    public static final String FIELD_TYPES_PARM = "fieldTypes";
    public static final String USES_PARM = "uses";
    public static final String SIG_PARM = "signature";
```



```

public static final String FIELD_NAMES_RD =
    "SessionLogListenerNotification.fieldNames.rd";
public static final String FIELD_TYPES_RD =
    "SessionLogListenerNotification.fieldTypes.rd";
public static final String[] _sigFieldNames = {
    "timeStamp",
    "component",
    "rootContext",
    "parentContext",
    "currentContext",
    "server",
    "eventCode",
    "user",
    "sessionName",
    "RPCs",
    "age"};
private Object[] _parsedValues = new Object[_sigFieldNames.length];

public void fillWmTemplateDescriptor(WmTemplateDescriptor descriptor, Locale l)
    throws ResourceException
{
    String[] parms = new String[] {FIELD_NAMES_PARM,
        FIELD_TYPES_PARM,
        USES_PARM,
        SIG_PARM};
    descriptor.createGroup(NOTIFICATION_SETUP_GROUP, parms);
    descriptor.createFieldMap(parms, false);
    descriptor.createTuple(new String[]{FIELD_NAMES_PARM, FIELD_TYPES_PARM});
    descriptor.setResourceDomain(FIELD_NAMES_PARM, FIELD_NAMES_RD, null);
    descriptor.setResourceDomain(FIELD_TYPES_PARM, FIELD_TYPES_RD, null);
    descriptor.setResourceDomain(SIG_PARM,
        WmTemplateDescriptor.OUTPUT_FIELD_NAMES, new String[]{
            FIELD_NAMES_PARM, FIELD_TYPES_PARM}, USES_PARM);
    descriptor.setDescriptions(MyAdapter.getInstance().
        getAdapterResourceBundleManager(), l);
}

public Boolean adapterCheckValue(WmManagedConnection connection,
    String resourceDomainName, String[][] values,
    String testValue) throws AdapterException
{
    return true;
}

public ResourceDomainValues[] adapterResourceDomainLookup(
    WmManagedConnection connection, String resourceDomainName,
    String[][] values) throws AdapterException
{
    ResourceDomainValues[] results = null;
    if (resourceDomainName.equals(FIELD_NAMES_RD)
        || resourceDomainName.equals(FIELD_TYPES_RD))
    {
        ResourceDomainValues names = new ResourceDomainValues(
            FIELD_NAMES_RD, _sigFieldNames);
        ResourceDomainValues types = new ResourceDomainValues(
            FIELD_TYPES_RD, new String[] {
                Date.class.getName(), //timestamp
                String.class.getName(), // component
                String.class.getName(), // rootContext
                String.class.getName(), // parentContext
                String.class.getName(), // currentContext
                String.class.getName(), // server
            }
        );
    }
}

```

```

        Integer.class.getName(), // eventCode
        String.class.getName(), // user
        String.class.getName(), // sessionName
        Integer.class.getName(), // RPCs
        Long.class.getName() // age
    });
    results = new ResourceDomainValues[] {names,types};
}
return results;
}
public void registerResourceDomain(WmManagedConnection connection,
WmAdapterAccess access) throws AdapterException
{
    access.addResourceDomainLookup(this.getClass().getName(),
        FIELD_NAMES_RD, connection);
    access.addResourceDomainLookup(this.getClass().getName(),
        FIELD_TYPES_RD, connection);
}
public boolean supports(Object data) throws ResourceException
{
    boolean result = false;
    try
    {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd H:mm:ss zzz");
        String sData = (String)data;
        this._parsedValues[0] = sdf.parse(sData.substring(0,48));
        StringTokenizer st = new StringTokenizer(sData.substring(49)," ",false);
        this._parsedValues[1] = st.nextToken();
        this._parsedValues[2] = st.nextToken();
        this._parsedValues[3] = st.nextToken();
        this._parsedValues[4] = st.nextToken();
        /*
        st.nextToken(); // skip the session ID
        this._parsedValues[5] = st.nextToken();
        this._parsedValues[6] = new Integer(st.nextToken());
        this._parsedValues[7] = st.nextToken();
        this._parsedValues[8] = st.nextToken();
        this._parsedValues[9] = new Integer(st.nextToken());
        this._parsedValues[10] = new Long(st.nextToken());
        */
        result = true;
    }
    catch(Throwable t){}
    return result;
}

public NotificationResults runNotification(NotificationEvent event)
throws ResourceException
{
    NotificationResults result = null;
    WmRecord notice = WmRecordFactory.getFactory().createWmRecord("notUsed");
    for(int i = 0; i < _sigFieldNames.length;i++)
    {
        if (_uses[i])
        {
            notice.put(_sigFieldNames[i],_parsedValues[i]);
        }
    }
    this.doNotify(notice);
    result = new AsyncNotificationResults(this.nodeName(),true,null);
}

```

```

return result;
}
}

```

Defining a WmAsyncListenerNotification Implementation Class

Create a class by extending `com.wm.adk.notification.WmAsyncListenerNotification` base class. This class must provide default implementations for the following abstract methods

- `fillWmTemplateDescriptor`
- `supports`
- `runNotification`

If you are not using the alternative resource domain management model, skip the following methods:

- `adapterCheckValue`
- `adapterResourceDomainLookup`
- `registerResourceDomain`

Specifying Configuration Metadata for Listener Notifications

The example listener notification in this section includes only those configuration parameters that are directly related to the signature. The only new concept in this section is the use of the *useParam* argument of the `WmTemplateDescriptor.setResourceDomain` method.

The useParam Argument of setResourceDomain

The `WmTemplateDescriptor` interface provides two signatures for the `setResourceDomain` method, one of which includes the argument *useParam*:

```

public void setResourceDomain(java.lang.String name,
                             java.lang.String resourceDomainName,
                             java.lang.String[] dependencies,
                             java.lang.String useParam)

```

- Use this argument as a filter to determine which of the available fields will be included in the signature. The example implements the *useParam* argument as `USES_PARM` constant, in the `setResourceDomain` method near the end of the example.
- The `USES_PARM` parameter is only meaningful when the parameter identified in the name argument and the parameter names in *useParam* parameter are in the same fieldMap, and the data type of the *useParam* parameter is `boolean[]`. When these conditions are satisfied, the adapter service editor treats the *useParam* setting as a filter on the resource domain association. When this occurs, the resource domain values are only applied to the name parameter row in the fieldMap if the corresponding value in the *useParam* is `true` which occurs when the user of the adapter selects the check box.

- When the user of the adapter selects the fields to use for the notification by selecting the check boxes in the **Uses** column, values appear in the **Signature** column. When you save changes to the notification node, the signature changes are reflected in the associated document type.

Implementing Configuration Resource Domains for Listener Notifications

Note:

In this example, notice the use of `class.getName` method when specifying data types. Using this technique enables you to catch spelling errors at compile time.

Implementing the supports and runNotification Method

This example implementation of the `supports` method parses the contents of the data object that is originally returned from the listener's `waitForData` method, into the `_parsedValues` object variable. Any error in the parsing process indicates that the data object was not a session log entry, and the `supports` method returns a `false` value.

The `runNotification` method implementation assumes that the `supports` method was successful in parsing the data object, so it does not need the `NotificationEvent` argument. It merely populates a `WmRecord` object by inserting the parsed names, using the same key names array that populate the resource domain.

Note:

The `runNotification` method calls `doNotify` method to publish the document. The `doNotify` method has two forms. When calling the `WmAsynchronousNotification.doNotify(WmRecord rec, String msgId)` form of this method, the length (the number of characters) of the value in `msgId` should not exceed a particular limit. To determine this limit, call the `WmAsynchronousNotification.adapterMaxMessageIdLen()` method. There is a fixed number of characters available in Integration Server to hold a notification ID. Of these, the `WmART` package reserves a certain number to hold a unique ID that it inserts prior to dispatching a notification. The remaining characters are available to you when calling `WmAsynchronousNotification.doNotify(WmRecord rec, String msgId)`. For more information, see the Javadoc for `WmAsynchronousNotification.doNotify`.

Updating the Resource Bundle

Update the resource bundle with a display name, and description to make the listener notification more usable.

```
package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.listeners.SessionLogListenerNotification;
..
..
public class MyAdapterResource extends ListResourceBundle implements MyAdapterConstants{
    ..
    ..
    static final Object[][] _contents = {
```

```

    ..
    ..
    //SessionLog Listener Notification
    ,{SessionLogListenerNotification.class.getName() +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "SessionLog Listener Notification"}
    ,{SessionLogListenerNotification.class.getName() +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Use SessionLog Listener Notification to monitor log files"}
    ,{SessionLogListenerNotification.FIELD_NAMES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Field Name"}
    ,{SessionLogListenerNotification.FIELD_NAMES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Field Name To Check"}
    ,{SessionLogListenerNotification.FIELD_TYPES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Field Types"}
    ,{SessionLogListenerNotification.FIELD_TYPES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Field Types To Check"}
    }
    protected Object[][] getContents() {
    // TODO Auto-generated method stub
    return _contents;
    }
}

```

Registering Listener Notifications in the Adapter

You must register each listener notification class in the `WmAdapter` implementation class. You do this by passing the class name to the `AdapterTypeInfo.addNotificationType` method in the `WmAdapter.fillAdapterTypeInfo` method in the `WmAdapter` implementation class. In the example, the listener notification class `SessionLogListenerNotification` is registered in the adapter implementation class `MyAdapter`:

For example:

```

package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.listeners.SessionLogListenerNotification;
..
..
public class MyAdapter extends WmAdapter implements MyAdapterConstants {
..
..
    public void fillAdapterTypeInfo(AdapterTypeInfo info, Locale locale)
    {
        ..
        ..
        info.addNotificationType(SessionLogListenerNotification.class.getName());
    }
}

```

Compiling Adapter

Compile your adapter as described in [“Compiling the Adapter”](#) on page 43.

Reloading Adapter

Reload your adapter as described in [“Loading, Reloading, and Unloading Packages”](#) on page 52.

Refreshing the Designer cache

Refresh the Designer cache.

Configuring and Testing Listener Nodes and Listener Notification Nodes

Now you are ready to configure a listener notification node as follows:

- [Configuring Listener Notification Nodes](#)
- [Testing Listener Nodes and Listener Notification Nodes](#)

Configuring Listener Notification Nodes

A listener notification node can be either synchronous or asynchronous. When an event occurs, an asynchronous listener notification publishes a document. You must create a trigger that receives the document and executes a service to process the document's data. With a synchronous listener notification, you can designate a flow service to process the data produced by the notification. A synchronous notification does not use a trigger.

> [To configure a listener notification node](#)

1. Start Designer.
2. Select a namespace node package where you want to create the listener notification.
3. Create a folder in the selected package, and navigate to that folder in **Package Navigator**.
4. Select **File > New**.
5. Select **Adapter Notification** from the list of elements.
6. In the **Create a New Adapter Notification** screen, type a name for your listener notification in the **Element name** field, and click **Next**.

7. In the **Select Adapter Type** screen, select the name of your adapter, and click **Next**.
8. In the **Select a Template** screen, select a listener notification template, and click **Next**.
9. In the **Select an Adapter Notification Listener** screen, select the appropriate adapter listener name, and click **Next**.
10. Perform one of the following steps:
 - If you selected an asynchronous listener notification template, click **Finish**.
 - If you selected a synchronous listener notification template, select a flow service to process the data produced by the notification, click **Next**, and then click **Finish**.

The Adapter Notification Editor opens.

11. In the Adapter Notification Editor, select an event.
12. Select **File > Save**.

If you selected an asynchronous listener notification template, the adapter creates a listener notification and a document named *notificationNamePublishDocument*. The publishable document contains the fields which are configured in the Listener Notification template.

Note:

Documents are not created for synchronous listener notifications.

When choosing the service for a synchronous notification, ensure that it does not require session data. For more information, see [“Synchronous and Asynchronous Listener Notifications” on page 150](#).


Testing Listener and Listener Notification Nodes

Creating the Flow Service for the Listener Notification Node

Perform the following procedure to create the flow service for the listener notification node.

➤ **To create the flow service for the listener notification node**

1. Start Designer.
2. In the Package Navigator, select the folder where you want to create the flow service.
3. Select **File > New**.
4. Select **Flow Service** from the list of elements.

5. In the **Create a New Flow Service** screen, type *TestMyAdapterFlowService* in the **Element name** field, and click **Next**.
6. On the **Select the Source Type** screen, select **Empty Flow**, and click **Finish**.
7. Click  to insert a flow step.
8. Navigate to the **pub.flow:savePipelineToFile** service in the **WmPublic** package, and click **OK**.

Note:

The `savePipelineToFile` service saves the contents of the pipeline (from the listener notification event) to the file that you specify in the **fileName** parameter.


9. Click the **Pipeline** tab.
10. Open the **fileName** parameter in the pipeline and set its value to *MonitorListenerNotificationPipeline.log*.


Click **OK**.

Creating the Trigger for the Listener Notification Node

Perform the following procedure to create the trigger for the Polling notification node.

> To create the trigger for the Polling notification node

1. Start Designer.
2. In the Package Navigator, select the folder where you want to create the trigger.
3. Select **File > New**.
4. Select **webMethods Messaging Trigger** from the list of elements.
5. On the **webMethods Messaging Trigger** screen, type *TestMyAdapterMsgTrigger* in the **Element name** field and click **Finish**.
6. In the trigger editor, in the Conditions section, accept the default **Condition1**.
7. In the Condition detail section, in the **Service** field, select or type the flow service name *TestMyAdapterFlowService*.
8. Click  to insert document types. Select **TestMyListenerNotificationPublishDocument** and click **OK**.

- Click  to save your trigger.

Enabling Listener Notification Nodes and Listener Nodes

Enabling the listener before you configure and enable its corresponding listener notification node produces a warning.

➤ To enable a listener node and a listener notification node

- Start Integration Server Administrator.
- In **Adapters** screen, select the name of your adapter.
- Select **Listener Notifications**.
- In the **Listener Notification** screen, enable the listener notification node by clicking **No** in the **Enabled** column. The **Enabled** column now shows **Yes** (enabled).
- Select **Listeners**.
- In the **Listeners** screen, enable the listener by clicking **No** in the **Enabled** column. The **Enabled** column now shows **Yes** (enabled).

Testing Listener

The file *MonitorListenerNotificationPipeline.log* is created in *Integration Server_directory / instances/<instance_name>/pipeline/* folder. This file contains one entry each time the file added in the listener is updated:

```
<?xml version="1.0" encoding="UTF-8"?>
<IDataXMLCoder version="1.0">
  <record javaclass="com.wm.data.ISMemDataImpl">
    <value name="fileName">MonitorListenerNotificationPipeline.log</value>
    <record
name="TestMyAdapterListener:TestMyAdapterListenerNotificationPublishDocument"
javaclass="com.wm.data.ISMemDataImpl">
      <Date name="timeStamp" type="java.util.Date">Thu Oct 21 12:34:45 IST 2021</Date>
      <value name="component"></value>
      <value name="rootContext">Unable</value>
      <record name="_env" javaclass="com.wm.data.ISMemDataImpl">
        <value name="locale"></value>
        <value name="activation">wm624455fb0-b66d-4344-b92f-cee92639b14c</value>
        <value
name="businessContext">wm6:2281c61b-a0dd-4dc2-bb31-6eaea2052a1c\snull\snull:
wm624455fb0-b66d-4344-b92f-cee92639b14c:null:IS_61:null</value>
        <value name="uuid">wm:14c48f70-323e-11ec-b7fd-000000000152</value>
        <value name="trackId">wm:14c48f70-323e-11ec-b7fd-000000000152</value>
        <value name="pubId">islocalpubid</value>
        <Date name="enqueueTime" type="java.util.Date">Thu Oct 21 12:41:15 IST
2021</Date>
        <Date name="recvTime" type="java.util.Date">Thu Oct 21 12:41:15 IST 2021</Date>
      </record>
    </record>
  </record>
</IDataXMLCoder>
```

```
<number name="age" type="java.lang.Integer">0</number>
</record>
</record>
</IDataXMLCoder>
```

Configuring and Testing Listener Nodes and Listener Notification Nodes

Now you are ready to configure a listener notification node as follows:

- Configuring Listener Notification Nodes
- Testing Listener Nodes and Listener Notification Nodes

Configuring Listener Notification Nodes

A listener notification node can be either synchronous or asynchronous. When an event occurs, an asynchronous listener notification publishes a document. You must create a trigger that receives the document and executes a service to process the document's data. With a synchronous listener notification, you can designate a flow service to process the data produced by the notification. A synchronous notification does not use a trigger.

> To configure a listener notification node

1. Start Designer.
2. Select a namespace node package where you want to create the listener notification.
3. Create a folder in the selected package, and navigate to that folder in **Package Navigator**.
4. Select **File > New**.
5. Select **Adapter Notification** from the list of elements.
6. In the **Create a New Adapter Notification** screen, type a name for your listener notification in the **Element name** field, and click **Next**.
7. In the **Select Adapter Type** screen, select the name of your adapter, and click **Next**.
8. In the **Select a Template** screen, select a listener notification template, and click **Next**.
9. In the **Select an Adapter Notification Listener** screen, select the appropriate adapter listener name, and click **Next**.
10. Perform one of the following steps:

- If you selected an asynchronous listener notification template, click **Finish**.
- If you selected a synchronous listener notification template, select a flow service to process the data produced by the notification, click **Next**, and then click **Finish**.

The Adapter Notification Editor opens.

11. In the Adapter Notification Editor, select an event.
12. Select **File > Save**.

If you selected an asynchronous listener notification template, the adapter creates a listener notification and a document named *notificationNamePublishDocument*. The publishable document contains the fields which are configured in the Listener Notification template.

Note:

Documents are not created for synchronous listener notifications.

When choosing the service for a synchronous notification, ensure that it does not require session data. For more information, see [“Synchronous and Asynchronous Listener Notifications” on page 150](#).

Testing Listener and Listener Notification Nodes

Creating the Flow Service for the Listener Notification Node

Perform the following procedure to create the flow service for the listener notification node.

➤ **To create the flow service for the listener notification node**

1. Start Designer.
2. In the Package Navigator, select the folder where you want to create the flow service.
3. Select **File > New**.
4. Select **Flow Service** from the list of elements.
5. In the **Create a New Flow Service** screen, type *TestMyAdapterFlowService* in the **Element name** field, and click **Next**.
6. On the **Select the Source Type** screen, select **Empty Flow**, and click **Finish**.
7. Click ➤ to insert a flow step.
8. Navigate to the **pub.flow:savePipelineToFile** service in the **WmPublic** package, and click **OK**.

Note:

The `savePipelineToFile` service saves the contents of the pipeline (from the listener notification event) to the file that you specify in the **fileName** parameter.



9. Click the **Pipeline** tab.
10. Open the **fileName** parameter in the pipeline and set its value to `MonitorListenerNotificationPipeline.log`.

Click **OK**.

Creating the Trigger for the Listener Notification Node

Perform the following procedure to create the trigger for the Polling notification node.

➤ To create the trigger for the Polling notification node

1. Start Designer.
2. In the Package Navigator, select the folder where you want to create the trigger.
3. Select **File > New**.
4. Select **webMethods Messaging Trigger** from the list of elements.
5. On the **webMethods Messaging Trigger** screen, type `TestMyAdapterMsgTrigger` in the **Element name** field and click **Finish**.
6. In the trigger editor, in the Conditions section, accept the default **Condition1**.
7. In the Condition detail section, in the **Service** field, select or type the flow service name `TestMyAdapterFlowService`.
8. Click  to insert document types. Select **TestMyListenerNotificationPublishDocument** and click **OK**.
9. Click  to save your trigger.

Enabling Listener Notification Nodes and Listener Nodes

Enabling the listener before you configure and enable its corresponding listener notification node produces a warning.

➤ To enable a listener node and a listener notification node

1. Start Integration Server Administrator.
2. In **Adapters** screen, select the name of your adapter.
3. Select **Listener Notifications**.
4. In the **Listener Notification** screen, enable the listener notification node by clicking **No** in the **Enabled** column. The **Enabled** column now shows **Yes** (enabled).
5. Select **Listeners**.
6. In the **Listeners** screen, enable the listener by clicking **No** in the **Enabled** column. The **Enabled** column now shows **Yes** (enabled).

Testing Listener

The file *MonitorListenerNotificationPipeline.log* is created in *Integration Server_directory / instances/<instance_name>/pipeline/* folder. This file contains one entry each time the file added in the listener is updated:

```
<?xml version="1.0" encoding="UTF-8"?>
<IDataXMLCoder version="1.0">
  <record javaclass="com.wm.data.ISMemDataImpl">
    <value name="fileName">MonitorListenerNotificationPipeline.log</value>
    <record
name="TestMyAdapterListener:TestMyAdapterListenerNotificationPublishDocument"
javaclass="com.wm.data.ISMemDataImpl">
      <Date name="timeStamp" type="java.util.Date">Thu Oct 21 12:34:45 IST 2021</Date>
      <value name="component"></value>
      <value name="rootContext">Unable</value>
      <record name="_env" javaclass="com.wm.data.ISMemDataImpl">
        <value name="locale"></value>
        <value name="activation">wm624455fb0-b66d-4344-b92f-cee92639b14c</value>
        <value
name="businessContext">wm6:2281c61b-a0dd-4dc2-bb31-6eaea2052a1c\snull\snull:
wm624455fb0-b66d-4344-b92f-cee92639b14c:null:IS_61:null</value>
        <value name="uuid">wm:14c48f70-323e-11ec-b7fd-000000000152</value>
        <value name="trackId">wm:14c48f70-323e-11ec-b7fd-000000000152</value>
        <value name="pubId">islocalpubid</value>
        <Date name="enqueueTime" type="java.util.Date">Thu Oct 21 12:41:15 IST
2021</Date>
        <Date name="recvTime" type="java.util.Date">Thu Oct 21 12:41:15 IST 2021</Date>
        <number name="age" type="java.lang.Integer">0</number>
      </record>
    </record>
  </record>
</IDataXMLCoder>
```


8 Runtime Activities

■ Overview	184
■ Retry and Recovery Architecture	184
■ Runtime Connection Allocation for Adapter Services	187

Overview

A well designed adapter must include the following runtime capabilities:

- Ability to identify, and recover from temporary errors.
- Ability to retrieve and manage connections.
- Allow the user to dynamically control the type of connection used for each service invocation.

For more information, see [“Runtime Connection Allocation for Adapter Services”](#) on page 187.

Retry and Recovery Architecture

A highly available and reliable system has the ability to recover from temporary errors. If a transient error is encountered during execution, then the system must perform the following tasks:

1. Detect the error
2. Remove or regenerate the component causing the error
3. Retry the operation

The first phase in recovery is defining a transient error. For a webMethods adapter, a transient error goes away in time, without requiring human intervention on Integration Server. For example:

- **Non-transient error.** If an adapter service performing an insert is called, and the backend resource rejects the insert because the data format is incorrect, then the error is not transient. The data must be manually reformatted for the insert to complete correctly.
- **Transient Error.** If an adapter service performing an insert is called, and the backend resource is offline for scheduled maintenance, then the insert fails. However, when the system comes back on line, the insert works. Retrying the operation in this scenario is useful.

Note:

- The adapter must detect the transient error.
- Integration Server provides the remove, regenerate, and retry functions.

Detection

The ability to recognize these errors is critical in every component of an adapter. It is up to the adapter developer to determine which backend errors are transient. When an adapter recognizes this situation, it throws an `AdapterConnectionException` to alert Integration Server about the error. The `AdapterConnectionException` is a subclass of the `ResourceException`. `AdapterConnectionException` is a valid exception from any ADK method that declares a `throws ResourceException`.

An adapter developer must isolate the errors that are transient. When Integration Server catches the `AdapterConnectionException`, then Integration Server initiates the retry operations. If Integration Server initiates the retry operations for non-transient errors, then the processing time is wasted.

Removal

Transient exceptions within adapters are related to connection errors. Clear and regenerate the connection used in the operation before retrying.

Based on the `AdapterConnectionException` received from the adapter, Integration Server cleans the entire pool or just the current connection. By default, the exception is an indication to clean the entire pool. For most applications this is the correct behavior.

- Destroying, and discarding the object, cleans the current connection.
- Notifying the connection pool manager, and destroying and discarding all the free connections cleans the pool.
- Releasing the connections destroys and discards the busy connections.
- The connection node remains enabled.

Regeneration

The regeneration occurs when a new connection is requested. If the connection is not pooled, an attempt is made to create the connection. If successful, a connection is created and used. If the connection is pooled and only the single connection is destroyed, another connection is reserved from the pool and used. If the pool is cleared, the pool attempts to fill to the minimum number of connections specified for the pool. If successful, a connection is reserved and used.

If the transient error occurs at any stage of the pool regeneration or connection creation, the adapter must throw an `AdapterConnectionException`, that ends the current retry operation. Then Integration Server starts another retry operation or ends the retry loop if the maximum attempts are made.

Retry Mechanisms

The retry facility is provided by Integration Server. The retry facility depends on the adapter construct that detected the failure. A retry is configured with a retry count and a backoff time. Retry count sets the number of times the action is tried before being considered a fatal error. The backoff time is the time interval between retries.

Note:

Throwing a non-retryable exception on any retry iteration is treated as a fatal exception and the retries stop.

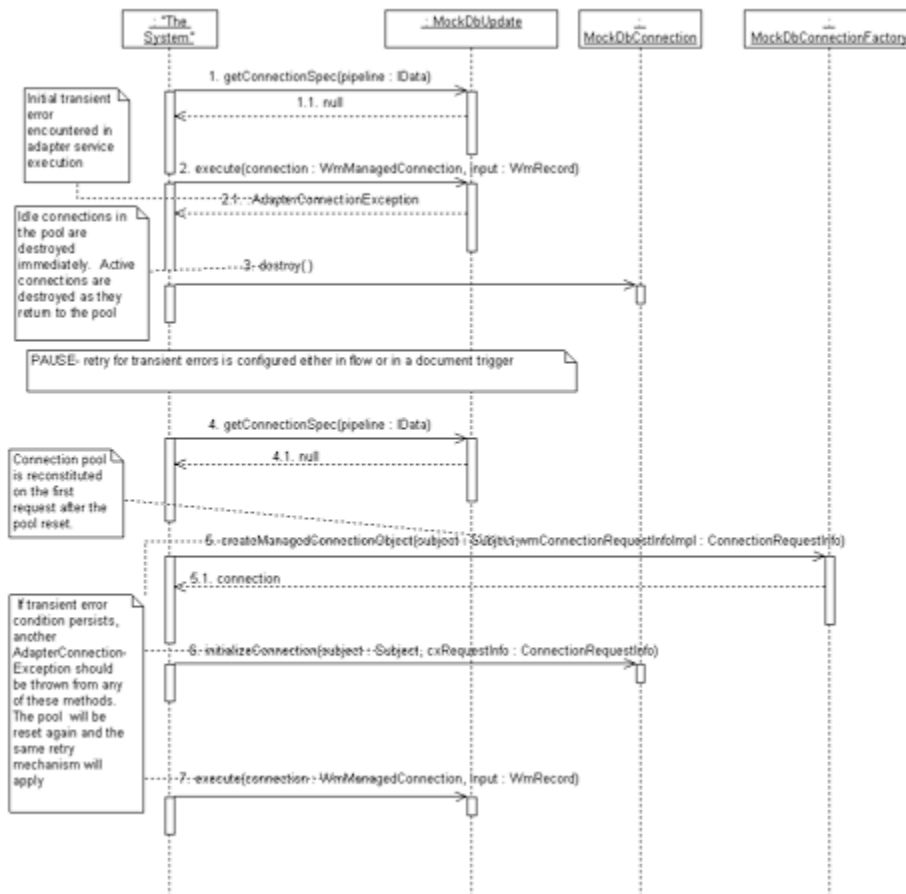
Below is a list of adapter objects and how retry behavior applies to each:

- **Connection Pools.** The retry process occurs in the following scenarios:
 - Beginning of the pool startup. If the backend system is down, and the user tries to enable the pool from Integration Server Administrator, the pool startup retries until the number of retries is exhausted or the retryable exception goes away. If the retries are exhausted, the pool is disabled.
 - Creating the pool during Integration Server startup.

- **Listeners.** The retry process occurs during the listener startup or during listener execution. If the number of retries is exhausted, the listener is disabled. During execution, if the listener notification produces an `AdapterConnectionException`, the corresponding listener goes into the retry mode.
- **Adapter Services.** Integration Server provides retry mechanism on services invoked from the triggers or flows invoked. An `AdapterConnectionException` caught from the adapter services execute method, is rethrown as an `ISRuntimeException`. Integration Server recognizes `ISRuntimeException` as a transient error and the retry cycle begins.
- **Polling Notifications.** The polling notification retry works differently from the other components. If a transient error is detected during the scheduled execution, the connection is cleaned. The next scheduled interval acts as the retry back off time. However, there is no maximum number of attempts. The notification remains enabled and executes on its next scheduled interval.

Retry Process in Adapter Services

The diagram shows the interactions with an adapter during an adapter service retry scenario.



- The transient error condition is initially detected in the adapter service's `execute` method (step 2).
- The adapter throws an `AdapterConnectionException` (step 2.1).
- The adapter throwing the `AdapterConnectionException` resets the connection pool by destroying all the idle connections immediately and any active connections as they return to the connection pool (step 3).
- When the connection pool is reset, the error is sent to the Integration Server service invocation logic as a retryable exception.
- If so configured, and after an appropriate timeout period, the adapter service is again invoked (beginning at step 4).
- Since the connection pool was emptied after the `AdapterConnectionException`, a new connection is created for the service invocation (steps 5 and 6).
- If a transient error condition occurs, then another `AdapterConnectionException` must be thrown from either the `createManagedConnectionObject` method or `initializeConnection` method call. The pool is reset again (step 3) and the retry process begins.

Runtime Connection Allocation for Adapter Services

When an adapter service is invoked, either directly or from a flow service, the Integration Servers' webMethods Adapter Runtime provides a connection object to the adapter services' implementation of `WmAdapterService.execute` method. This section describes how connections are retrieved and managed, and how to dynamically control the type of connection used for each service invocation.

At runtime, all connection activities for adapter services are performed inside a transaction context that holds references to connections used when the context is open. This is true regardless of whether the referenced connections are transacted or not. 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 may be created using the `pub.art.transaction:startTransaction` and closed using `pub.art.transaction:commitTransaction` or `pub.art.transaction:rollbackTransaction`. For more information about using these services, see *webMethods Integration Server Built-In Services Reference* for your release.

When Integration Servers' webMethods 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 first determines whether a connection from the required connection pool and the partition are in the context; if so, Integration Server uses the connection from the transaction context for use by the adapter service instead of requesting another from the connection pool.

When a connection is requested from a particular connection pool, the request may identify a partition in the form of an adapter-generated `ConnectionRequestInfo` object. A connection pool partition is a logical division within a given connection pool where connection objects in different partitions are used at different times in an adapter-defined way.

- If no `ConnectionRequestInfo` object is provided in the request, then the default (or null) partition is used.
- If the pool has an available connection in the specified partition, the pool marks that connection as busy and returns the connection to the caller.
- If the pool has no available connection in the specified partition, and the pool is not full, then the pool requests a new connection from the associated connection factory, including any provided `ConnectionRequestInfo` object.

Dynamically Selecting a Connection Node

Each connection node must 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.

The connection node used for a particular invocation is determined as follows:

1. The adapter may specify a connection name by overriding the default implementation of the `WmAdapterService.getConnectionSpec` method to return a `WmConnectionSpec` object containing the connection name. For more information on using `WmConnectionSpec` objects, see [“Implementing WmConnectionSpec” on page 192](#).
2. If the connection is not specified using `WmAdapterService.getConnectionSpec` method, the connection name may be specified on the pipeline in the **\$connectionName** field. Integration Server checks for a value in **\$connectionName**, if the field is part of the service's input signature. For more information about exposing **\$connectionName** field in the service signature, see the Javadoc for `WmDescriptor.showConnectionName` method.
3. If a connection node name has still not been specified, the service's default connection is used. The default connection is the connection that is used when the adapter service is created, unless it is changed using either the `pub.art.service:setAdapterServiceNodeConnection` or the `wm.art.dev.service:updateAdapterServiceNode` service.

To update a service node, in Designer you must either lock the node for editing or check out the node. When the Integration Servers' `watt.server.ns.lockingMode` property is set to `system`, you must obtain **Write ACL** access to the service node that you want to edit before updating the node. For information about obtaining Write ACL access, see the *webMethods Service Development Help* for your release.

Partitioned Connection Pools

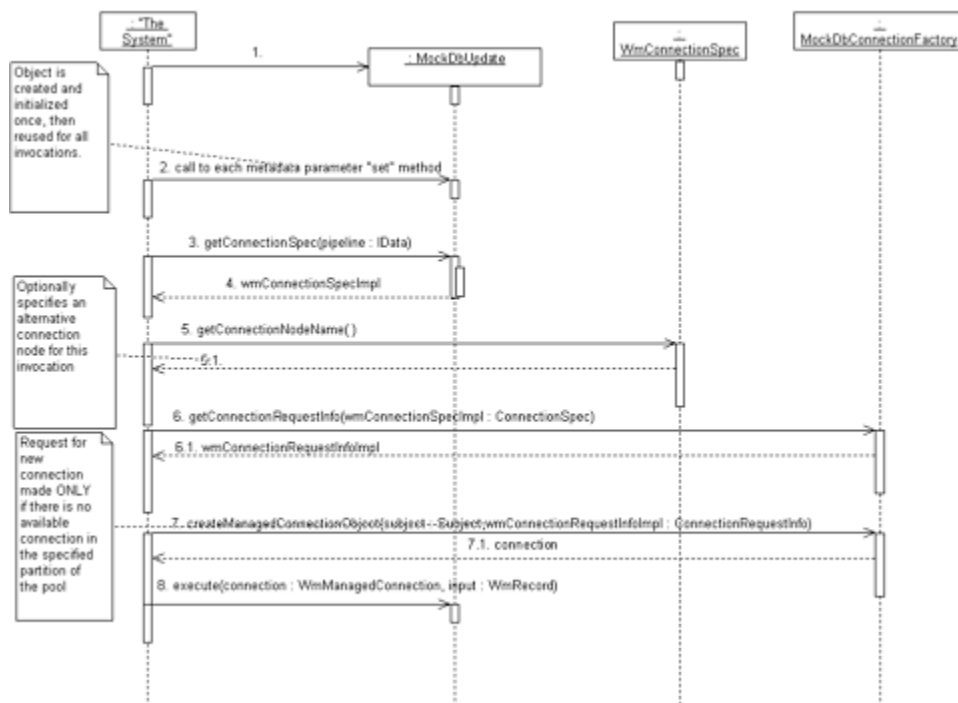
While a connection node defines a general set of connections to a backend resource, it is sometime necessary to connect to the backend using the attributes that are specific to a particular operation or data set. For example, the backend may require that a connection be established with a specific set of user credentials in order to update a given record. If the number of unique attributes is small, it may be possible to define a connection pool with each set of attributes and select the appropriate pool at run time based on the service being called or the data being manipulated. When this is not practical, an adapter must implement connection pool partitioning.

A connection pool partition is a logical division within a given connection pool where connection objects in different partitions are used at different times in an adapter-defined way. Connections in different partitions have different permissions, and are often associated with different users. However, the use of partitions and how they are segregated is entirely determined by the adapter implementation. Connection objects are assigned to a particular partition at the time they are created and remain in the same partition for the life of the object.

When a connection pool is started, it is initialized with connections in the default (null) partition. Additional partitions are created and populated as the connections from those partitions are requested.

Obtaining a Connection in a Partition of the Connection Pool

The diagram shows the interactions when a connection is obtained in a specified partition of the pool.



The process of requesting a connection from a particular partition begins in the `WmAdapterService.getConnectionSpec` method. Adapter developers must extend the `WmConnectionSpec` class to capture any context information necessary to determine the required partition. This may include information from the invocation pipeline, as well as information about the service being invoked, including the metadata parameter settings.

The `WmConnectionSpec` object returned by the service implementation is then passed to the `getConnectionRequestInfo` method of the connection factory associated with the selected connection node. The connection factory is then required to identify the connection pool partition based on the information in the `WmConnectionSpec` object.

Note:

The connection node need not be identified in the `WmConnectionSpec` object.

For more information about selecting a connection node for an invocation, see [“Enabling Connection Nodes” on page 69](#).

Implementing Partitioned Connection Pools

In order to support partitioned connection pools in an adapter, the adapter developer must perform the following:

- Create a class that defines the partition by extending the `com.wm.adk.connection.WmConnectionRequestInfo` base class.
- Create a class to hold any required information gathered from an adapter service invocation by extending the `com.wm.adk.connection.WmConnectionSpec` base class.
- Implement the `getConnectionRequestInfo` method in the adapter's `WmManagedConnectionFactory` implementation.
- Implement the `getConnectionSpec` method in the adapter's `WmAdapterService` implementation.

For example, MegaBank is a company, regularly acquiring and absorbing the customer base of smaller banks. A customer acquired in this way is assigned a new MegaBank user name with which the customer can access the accounts as well as other services provided by the larger bank. Since MegaBank has adapters for most major banking systems, MegaBank is able to seamlessly integrate the new bank's system very quickly by mapping the customer's new MegaBank user name with login information already present in the acquired bank's system. Using partitioned pools, connections are established using authentication information already known to the backend.

Example `WmConnectionRequestInfo`

```
public class MbConnectionRequestInfo extends WmConnectionRequestInfo
{
    private final String userId;
    private final String mbUserName;
    private final Credentials credentials;
    private final String logName;

    /**
     * Sole constructor. verifies all fields populated.
     *
     * @param mbUserName - common user name used across the integration
     * @param userId - name of user as known to the backend system
     * @param credentials - credentials required for this user to access
     *                      the backend system
     * @throws IllegalArgumentException - if any argument is null
     */
    public MbConnectionRequestInfo(String mbUserName, String userId,
                                   Credentials credentials) throws IllegalArgumentException
    {
        if (mbUserName == null || userId == null || credentials == null)
        {
```

```

        throw new IllegalArgumentException();
    }
    this.userId = userId;
    this.mbUserName = mbUserName;
    this.credentials = credentials;
    this.logName = mbUserName + "(" + userId + ")";
}

public Credentials getCredentials()
{
    return credentials;
}
public String getMbUserName()
{
    return mbUserName;
}
public String getUserId()
{
    return userId;
}

/**
 * Name used by the connection pool when creating log entries relevant
 * to the partition identified by this object.
 */
public String getLoggableName()
{
    return logName;
}
/**
 * In a ConnectionRequestInfo object the hashCode of objects that
 * identify the same partition must be the same, so we use only the
 * hashCode of the userId String.
 */
public int hashCode()
{
    return this.userId.hashCode();
}
/**
 * If two ConnectionRequestInfo objects identify the same partition,
 * then the equals method must return true. We compare the userIds of
 * the two object rather than allow the default equals implementation
 * which only checks if they are the same object instance.
 * @param obj - object being compared to this.
 */
public boolean equals(Object obj)
{
    boolean isEqual = false;
    if(obj instanceof MbConnectionRequestInfo)
    {
        isEqual = this.userId.equals(((MbConnectionRequestInfo)obj).userId);
    }
    return isEqual;
}
}

```

Example WmConnectionSpec

```

public class MbConnectionSpec extends WmConnectionSpec
{

```

```
private String mbUserName;
public MbConnectionSpec()
{
    super();
}
public String getMbUserName()
{
    return mbUserName;
}
public void setMbUserName(String mbUserName)
{
    this.mbUserName = mbUserName;
}
}
```

Implementing WmConnectionRequestInfo

A ConnectionRequestInfo object defines a partition both within the connection pool and in WmManagedConnectionFactory.createManagedConnectionObject when a new connection is created. The connection pool organized its connections based on the ConnectionRequestInfo object used when the connection was created. In this case, ConnectionRequestInfo objects X and Y are considered the same if `X.hashCode() == Y.hashCode() && X.equals(Y)`. The remainder of the ConnectionRequestInfo implementation is defined by the adapter.

1. Create a class by extending the `com.wm.adk.connection.WmConnectionRequestInfo` base class.

In the example, a *MbConnectionRequestInfo* class is created.

2. Create a partition with which the backend connection is established.

In the example, a partition is defined by the *userId* with which the backend connection is established.

3. Add the methods to ensure that the connection pool properly recognizes objects that refer to the same partition.

In the example, the `hashCode` and `equals` methods refer to the *userId*.

4. Implement the `getLogableName` abstract method to provide a partition name that can be recorded in Integration Server logs.

In the example, `getLogableName` method is also used to correlate the MegaBank user name with the ID used to access the backend.

Note:

Care must be taken in implementing this method to avoid exposing any sensitive data such as passwords.

Implementing WmConnectionSpec

A WmConnectionSpec object is used during an adapter service invocation to identify a connection node and/or to hold any contextual information from the service configuration and invocation

pipeline that is needed to identify the partition of the connection. Accessors for the connection node name are provided in the base class. Contextual information need for partition definition is specific to the adapter and must be implemented in an adapter-defined subclass.

1. Create a class by extending the `com.wm.adk.connection.WmConnectionSpec` base class.

In the example, a `MbConnectionSpec` class is created.

2. Create a get and a set method for the username.

In the example, each customer is assigned a username that allows access to all of the bank's services. The username value must be cross-referenced to obtain the name by which that user is known on the specific backend system. This cross-referencing is delegated to the `WmConnectionFactory.getConnectionRequestInfo` implementation.

Updating the Connection Factory

The adapter's `WmManagedConnectionFactory` implementation must be able to produce valid `WmConnectionRequestInfo` objects and it must be able to use those objects to create connections with the required characteristics to support the partitioned connection pool feature.

1. Implement the `getConnectionRequestInfo` method.

In the example, `getConnectionRequestInfo` uses information from the `ConnectionSpec` object to produce a valid `ConnectionRequestInfo` object. The MegaBank user name is used as a key to lookup a set of user ID and credentials appropriate for the backend being accessed by this connection.

Note:

Details of how this lookup is implemented are adapter specific and outside the scope of this document.

```
/**
 * Produce a ConnectionRequestInfo object based on the provided ConnectionSpec
 * object. If an mbUserName is provided, lookup the name to use on the backend
 * for this connection pool.
 */
public ConnectionRequestInfo getConnectionRequestInfo(ConnectionSpec spec)
{
    MbConnectionRequestInfo partitionDef = null;
    if(spec != null && spec instanceof MbConnectionSpec)
    {
        synchronized(this)
        {
            try
            {
                String mbUserName = ((MbConnectionSpec)spec).getMbUserName();
                lookupUser(mbUserName);
                partitionDef = new MbConnectionRequestInfo(mbUserName,
                    this.backendUserID, this.credentials);
            }
            catch(IllegalArgumentException ex)
            {
                // no partition info available for this user. Swallow the
                // exception and return null
            }
        }
    }
}
```

```

}
}
}
return partitionDef
}

```

2. Update the createManagedConnectionObject method.

The ConnectionRequestInfo object returned from getConnectionRequestInfo method is passed to the createManagedConnectionObject method when the connection pool needs a new connection belonging to a particular partition. In the example, this information is used to override default logon information that is otherwise established based on the metadata parameter settings.

Note:

The createManagedConnectionObject method implementations must be able to establish default-partition connections when null is passed in the cxRequestInfo argument.

```

/**
 * Create a connection using userId and credentials from the provided
 * connectionRequestInfo object, if provided. If connectionRequestInfo is
 * not provided use the default userId and credentials information established
 * at node startup from metadata parameters.
 */
public WmManagedConnection createManagedConnectionObject(Subject subject,
    ConnectionRequestInfo cxRequestInfo) throws ResourceException
{
    String connUser;
    Credentials connCredentials;
    if (cxRequestInfo != null && cxRequestInfo instanceof
        MbConnectionRequestInfo)
    {
        connUser = ((MbConnectionRequestInfo)cxRequestInfo).getUserId();
        connCredentials =
            ((MbConnectionRequestInfo)cxRequestInfo).getCredentials();
    }
    else
    {
        connUser = this.defaultUserId;
        connCredentials = this.defaultCredentials;
    }
    return new MbConnection(connUser, connCredentials);
}

```

9 Usage Scenarios

■ How to register an adapter with the Integration Server?	196
■ How to create an adapter connection implementation?	209
■ How to create an adapter service implementation?	216
■ How to create a polling notification implementation?	253
■ How to create an adapter listener implementation?	278

How to register an adapter with the Integration Server?

An adapter definition is the framework of an adapter. Although an adapter definition is recognized as an adapter by Integration Server, it lacks functionality. This chapter describes how to create an adapter definition.

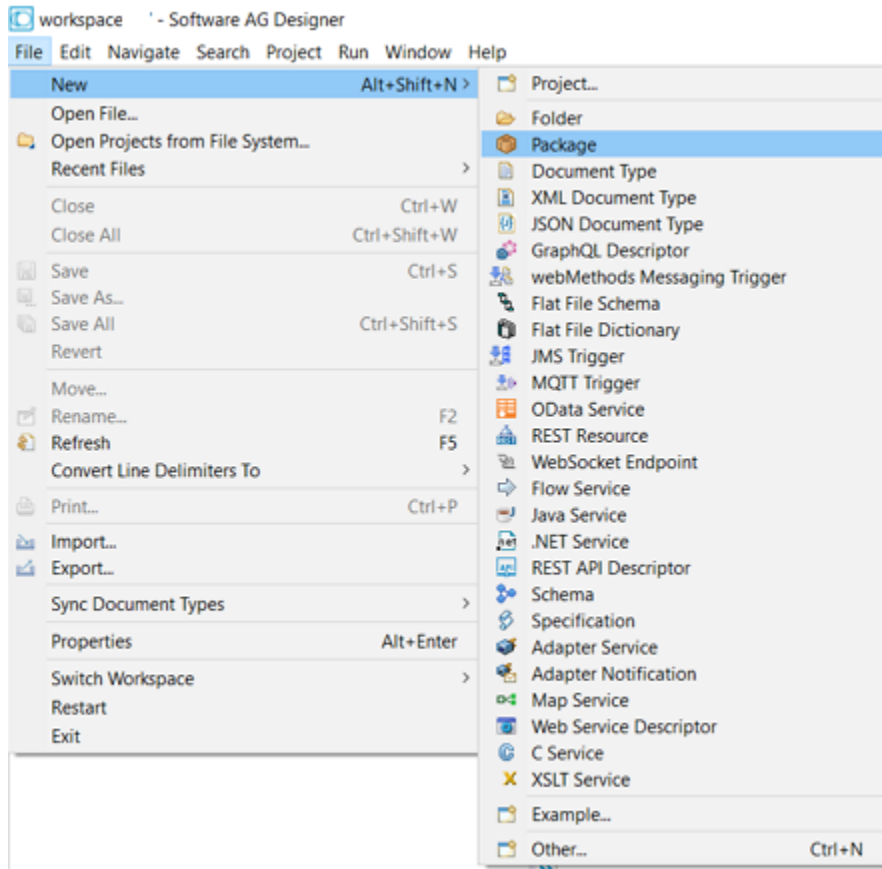
Pre-requisites:

- Integration Server 9.12 or later installed.
- Designer 9.12 or later installed.
- Integration Server Administrator access.
- Java 1.8 or later installed.
- Basic understanding of Integration Server, Designer, Integration Server Administrator, Java.

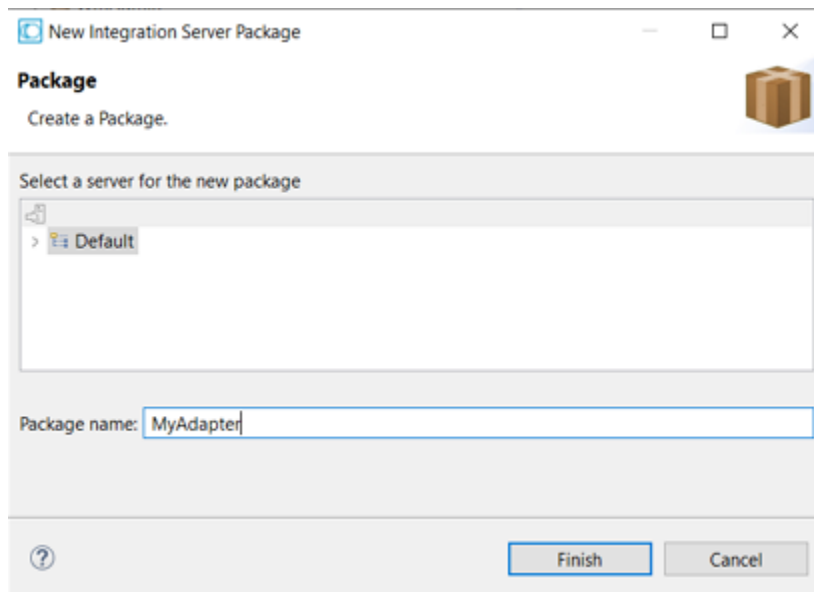
In Integration Server Administrator, in the navigation, when you select **Adapters**, the adapter created by you appears in the dropdown.

Creating an Adapter Package

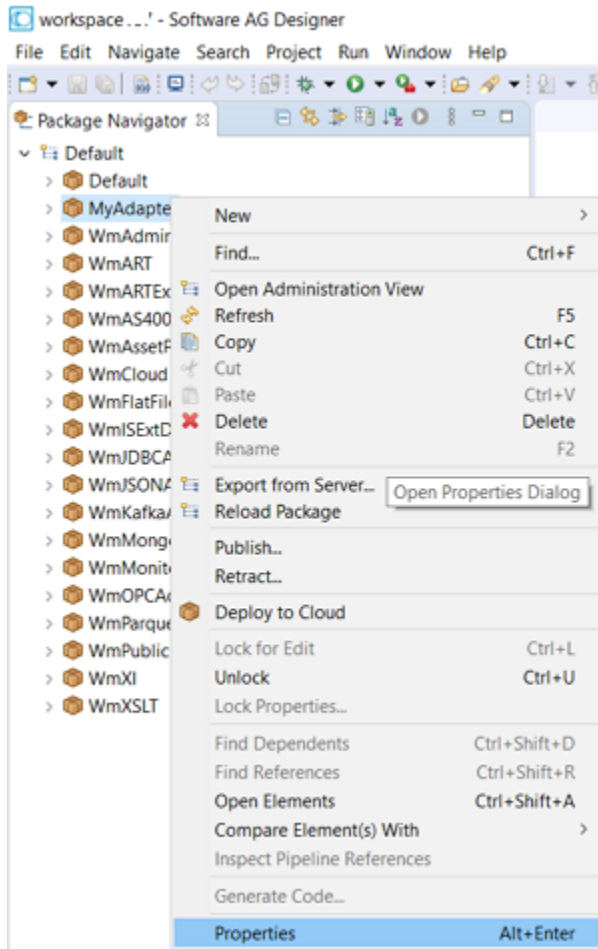
1. Start Integration Server.
2. Open Designer.
3. In Designer, in the **Package Navigator**, select the **Default** package.
4. Select **File > New > Package**.




5. Enter the **Package name**. For example: *MyAdapter*.



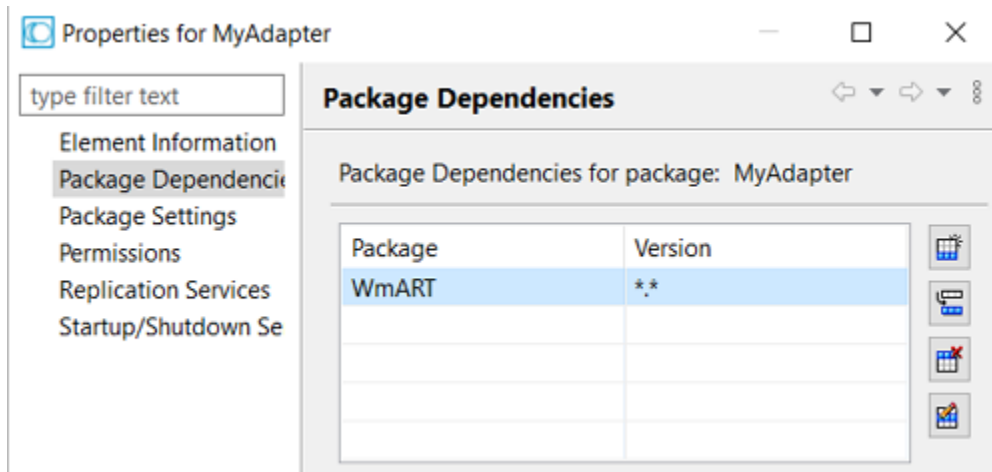
6. Select the package created and right-click to select **Properties**.



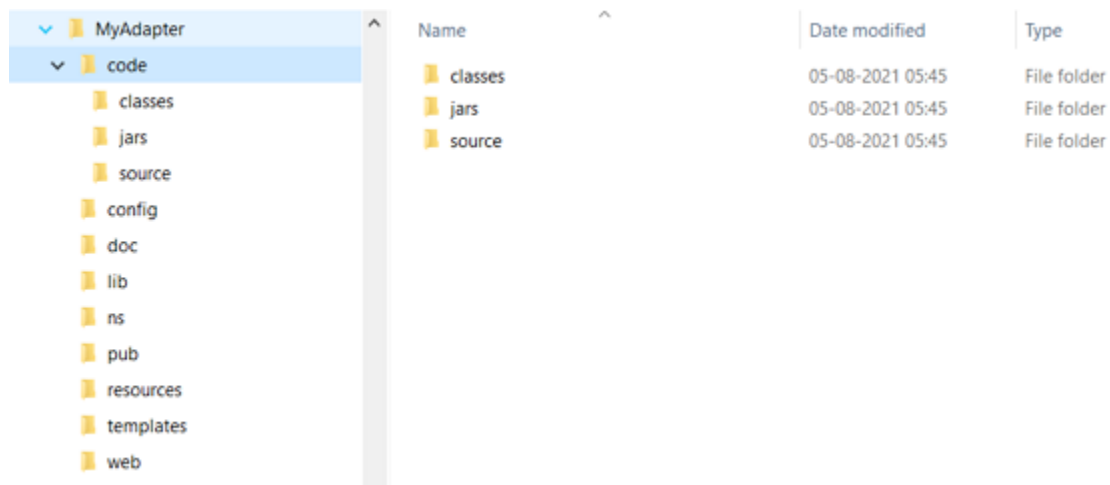
7. In **Properties**, select **Package Dependencies**.

Click  to add a row and specify values for the following fields:

Field	Value
Package	WmART
Version	*.*.*



The adapter package that contains the adapter implementation is created. The following folder structure is created in *Integration Server_directory* \instances*<instance_name>* \packages\MyAdapter.



Creating an Adapter Implementation Class

1. Start the editor to create Java files.
2. Create directories corresponding to your Java package structure in the webMethods package you created using Designer. For example: *Integration Server_directory* \instances*<instance_name>* \packages\MyAdapter\code\source folder.
3. Create the following classes and interfaces:
 - a. An interface that contains the constants for the adapter implementation.

In the example, the interface created is *MyAdapterConstants*:

```
package com.wm.MyAdapter;
public interface MyAdapterConstants {
```

```

static final int ADAPTER_MAJOR_CODE = 9001;
static final String ADAPTER_JCA_VERSION = "1.0";
static final String ADAPTER_NAME = "MyAdapter";
static final String ADAPTER_VERSION = "9.12";

//Using next statement will create cyclic class loading dependency issue
//therefore, the resource bundle class name is fully spelled out
//static final String ADAPTER_SOURCE_BUNDLE_NAME =
MyAdapterResource.class.getName();
static final String ADAPTER_SOURCE_BUNDLE_NAME =
    "com.wm.MyAdapter.MyAdapterResource";
}

```

- b. A class by extending the base class `com.wm.adk.WmAdapter`.

In the example, the class created is *MyAdapter*:

```

package com.wm.MyAdapter;
import java.util.Locale;
import com.wm.adk.WmAdapter;
import com.wm.adk.error.AdapterException;
import com.wm.adk.info.AdapterTypeInfo;
import com.wm.adk.log.ARTLogger;

```

```

public class MyAdapter extends WmAdapter implements MyAdapterConstants{
    public static MyAdapter _instance = null;
    public static ARTLogger _logger = null;

    public MyAdapter() throws AdapterException { super(); }
    public void fillAdapterTypeInfo(AdapterTypeInfo arg0, Locale arg1) {}
    public String getAdapterJCAVersion() { return ADAPTER_JCA_VERSION; }
    public int getAdapterMajorCode() { return ADAPTER_MAJOR_CODE; }
    public String getAdapterName() { return ADAPTER_NAME; }
    public String getAdapterResourceBundleName() { return
ADAPTER_SOURCE_BUNDLE_NAME; }
    public String getAdapterVersion() { return ADAPTER_VERSION; }
    public static ARTLogger getLogger() { return _logger; }
}

```

```

public void initialize() throws AdapterException {
    // TODO Auto-generated method stub
    _logger = new ARTLogger(getAdapterMajorCode(),
        getAdapterName(),
        getAdapterResourceBundleName());
    _logger.logDebug(9999,"My Adapter Initialized");
}
public void cleanup() {
    if (_logger != null)
        _logger.close();
}
}

```

```

public static MyAdapter getInstance() {
    // TODO Auto-generated method stub
    if (_instance != null)
        return _instance;
    else {
        synchronized (MyAdapter.class) {
            if (_instance != null) {

```



```

        return _instance;
    }
    try {
        _instance = new MyAdapter();
        return _instance;
    } catch (Throwable t) {
        t.printStackTrace();
        return null;
    }
}
}
}
}
}
}

```

- c. Create a resource bundle class by extending the base class `java.util.ListResourceBundle`.

In the example, the class created is *MyAdapterResource*:

```

package com.wm.MyAdapter;
import java.util.ListResourceBundle;
import com.wm.adk.ADKGLOBAL;

public class MyAdapterResource extends ListResourceBundle implements
MyAdapterConstants{
    static final String IS_PKG_NAME = "/MyAdapter/";
    static final Object[][] _contents = {
        // adapter type display name.
        {ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME, "My Adapter"}
        // adapter type descriptions.
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Adapter for MyAdapter Server (a Sample System)"}
        // adapter type vendor.
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_VENDORNAME, "Software AG"}
        //Copyright URL Page
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_THIRDPARTYCOPYRIGHTURL,
        IS_PKG_NAME + "copyright.html"}
        //Copyright Encoding
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_COPYRIGHTENCODING, "UTF-8"}
        //About URL Page
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_ABOUT, IS_PKG_NAME +
        "About.html"}
        //Release Notes URL Page
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_RELEASENOTEURL, IS_PKG_NAME +
        "ReleaseNotes.html"}
    };
    protected Object[][] getContents() {
        // TODO Auto-generated method stub
        return _contents;
    }
}
}

```

- d. Specify the adapter's default resource bundle in your `WmAdapter` implementation class. You can return the name of your default resource bundle using the `getAdapterResourceBundleName` method in your `WmAdapter` implementation class.

Example of `getAdapterResourceBundleName` in *MyAdapter* class:

```
@Override
public String getAdapterResourceBundleName() {
    // TODO Auto-generated method stub
    return ADAPTER_SOURCE_BUNDLE_NAME;
}
```

Example of `ADAPTER_SOURCE_BUNDLE_NAME` constant in `MyAdapterConstants` interface:

```
//static final String ADAPTER_SOURCE_BUNDLE_NAME =
MyAdapterResource.class.getName();
static final String ADAPTER_SOURCE_BUNDLE_NAME =
    "com.wm.MyAdapter.MyAdapterResource";
```

- e. Create the reference pages for copyright, and index page for the adapter in `adapterPackageName/pub` folder.

Note:

You must create your reference pages in the same `adapterPackageName/pub` folder in the `webMethods` package you created using Designer.

All the Java classes and html pages for adapter implementation are created.

Creating Adapter Startup and Shutdown Java Services

1. Create adapter startup and shutdown Java Service.
 - Create Java class for adapter startup and shutdown, which is used to generate corresponding Java Services using jcode utility.
 - Create a folder structure for the Java package for adapter admin class. For example: `adapterPackageName\wm\mycompany\adapteradmin`. In the example, the Java package created is `adapterPackageName\wm\MyAdapter`.
 - Create adapter admin Java class.

```
package wm.MyAdapter;
//--- <<IS-START-IMPORTS>> ---
import com.wm.MyAdapter.*;
import com.wm.adk.admin.AdapterAdmin;
import com.wm.app.b2b.server.ServiceException;
import com.wm.data.IData;
//--- <<IS-END-IMPORTS>> ---
public class MyAdapterAdmin {

    public static final void startUp (IData pipeline)
        throws ServiceException
    {
        // --- <<IS-START(startUp)>> ---
        AdapterAdmin.registerAdapter(MyAdapter.getInstance());
        // --- <<IS-END>> ---
    }
    public static final void shutDown (IData pipeline)
        throws ServiceException
    {
        // --- <<IS-START(shutDown)>> ---
```

```

        MyAdapter instance = MyAdapter.getInstance();
        instance.cleanup();
        AdapterAdmin.unregisterAdapter(instance);
        // --- <<IS-END>> ---
    }
}

```

Note:

Tags are used to mark the beginning and end of imports and methods. For more information, see *webMethods Service Development Help*.

- Create adapter startup and shutdown Java Service using Designer.
 - Open Designer.
 - Select the webMethods package you created using Designer.
 - Create a folder structure for the Java package for adapter admin Java Services. For example: `wm\mycompanyname\adapteradmin`. In the example, the folder structure created is `wm\MyAdapter`.
 - Create a new Java Service for adapter startup. In the example, the Java Service created is *startUp*.
 - Add the following Java code.

```

public static final void startUp(IData pipeline) throws
ServiceException {
    // --- <<IS-START(startUp)>> ---
    AdapterAdmin.registerAdapter(MyAdapter.getInstance());
    // --- <<IS-END>> ---
}

```

Note:

Tags are used to mark the beginning and end of imports and methods. For more information, see *webMethods Service Development Help*.

- Create a new Java Service for adapter shutdown. In the example, the Java Service created is *shutDown*.
- Add the following Java code.

```

public static final void shutDown(IData pipeline) throws
ServiceException {
    // --- <<IS-START(shutDown)>> ---
    MyAdapter instance = MyAdapter.getInstance();
    instance.cleanup();
    AdapterAdmin.unregisterAdapter(instance);
    // --- <<IS-END>> ---
}

```

Note:

Tags are used to mark the beginning and end of imports and methods. For more information, see *webMethods Service Development Help*.

The Java classes for adapter admin are created.

Compiling the Adapter

1. Create an ANT script to compile the adapter classes, and deploy the admin classes as Java Services. For example: `build.xml` and `build.properties`.

File `build.properties`:

```
# The Site Name
debug=on
optimize=off
deprecation=off
webM.home=C:/softwareag/912
server.home=${webM.home}/IntegrationServer
package=MyAdapter
instance_name=default
srcdir=${server.home}/instances/${instance_name}/packages/${package}/code/source
destdir=${server.home}/instances/${instance_name}/packages/${package}/code/classes
```

File `build.xml`:

```
<?xml version="1.0"?>
<project name="Adapter using ADK" default="deploy" basedir=".">
  <property file="build.properties" />
  <!-- classes belonging to this package -->
  <path id="this.package.classpath">
    <fileset dir="${server.home}/instances/${instance_name}/packages/${package}/">
      <include name="code/classes"/>
    </fileset>
  </path>
  <!-- All classes that need to be found by this script -->
  <path id="total.classpath">
    <pathelement
location="${server.home}/instances/${instance_name}/packages/WmART/code/jars/wmart.jar"/>
    <pathelement location="${server.home}/lib/wm-isserv.jar"/>
    <pathelement location="${webM.home}/common/lib/wm-isclient.jar"/>
    <pathelement location="${webM.home}/common/lib/glassfish/gf.jakarta.resource.jar"/>
    <pathelement
location="${server.home}/instances/${instance_name}/packages/WmART/code/classes"/>

    <path refid="this.package.classpath"/>
  </path>

  <!-- Compile the java files of this package -->
  <target name="createclasses" depends="init">
    <echo>Creating classes</echo>
    <mkdir dir="${destdir}"/>
    <javac debug="${debug}" optimize="${optimize}"
deprecation="${deprecation}" srcdir="${srcdir}"
destdir="${destdir}">
      <classpath>
        <path refid="total.classpath"/>
      </classpath>
    </javac>
  </target>
  <!-- Execute jcode -->
  <target name="execjcode" depends="createclasses">
```

```

<echo>Deploying classes</echo>
<exec executable="${server.home}/instances/${instance_name}/bin/jcode"
  vmlauncher="false" failonerror="true">
  <arg value="fragall" />
  <arg value="${package}" />
</exec>
</target>
<!-- delete .class files built in this package -->
<target name="cleanclasses">
  <echo>Cleaning classes</echo>
  <mkdir dir="${destdir}"/>
  <delete quiet="false">
    <fileset dir="${destdir}" includes="**/*.class"/>
  </delete>
</target>

<!-- if this package depends on classes found in other packages,
  add targets to build those classes here. -->
<target name="init">
  <tstamp/>
</target>

<target name="packageDependencies" depends="" />
<target name="clean" depends="cleanclasses" />
<target name="classes" depends="cleanclasses, createclasses" />
<target name="deploy" depends="execjcode" />
<target name="all" depends="packageDependencies, cleanclasses, execjcode" />
<target name="remake" depends="packageDependencies, cleanclasses, createclasses"
/>
</project>

```

- a. Set the classpath in *total.classpath* in the ANT script.

The JAR files required to compile your source code are as follows:

- *Software AG_directory* \common\lib\wm-isclient.jar
- *Software AG_directory* \common\lib\glassfish\gf.jakarta.resource.jar
- *Integration Server_directory* \lib\wm-isservice.jar
- *Integration Server_directory* \instances\<instance_name>\packages\WmART\code\jars\wmart.jar

The folder containing the class files required to compile your source code is as follows:

- *Integration Server_directory* \instances\<instance_name>\packages\WmART\code\classes

Software AG_directory is the folder in which webMethods components are installed and *Integration Server_directory* is the folder in which Integration Server is installed.

2. Run the ANT script to compile the Java classes.

```
ant classes
```

3. Compile the Java classes and deploy in Integration Server as Java Services.

- a. Run the ANT script to compile the Java classes and deploy in Integration Server as Java Services.

```
ant deploy
```

Note:

If you have created adapter admin Java class for startup and shutdown, then the corresponding Java Services are deployed using jcode utility. The jcode utility is provided with Integration Server. For more information, see *webMethods Service Development Help*.

- b. If you have created startup and shutdown Java Services using Designer, you must compile it using Integration Server Administrator.

- Start Integration Server Administrator.
- Select **Settings > Extended > Edit Extended Settings**.
- Set the property `watt.server.compile` to include the path to Java compiler and the classpath to include the `wmart.jar` in `Integration Server_directory \ instances \ <instance_name> \ packages \ WmART \ code \ jars \ wmart.jar`. For example:

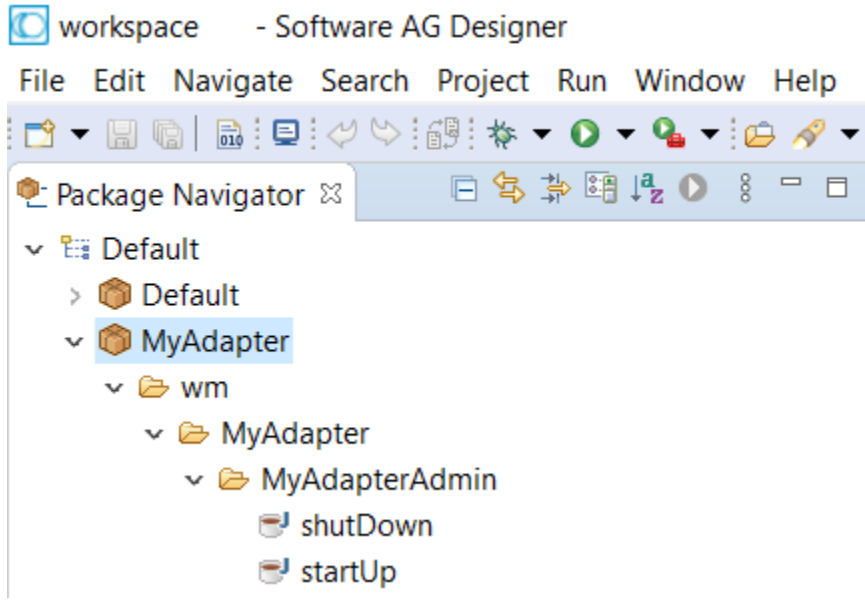
```
watt.server.compile=C:\softwareag\912\jvm\jvm\bin\javac
-classpath
{0};C:\softwareag\912\IntegrationServer\instances\default\packages\
WmART\code\jars\wmart.jar; -d {1} {2}
```



4. Restart Integration Server.

5. Refresh Designer.

The Java Services are deployed. The startup and shutdown Java Services appear.

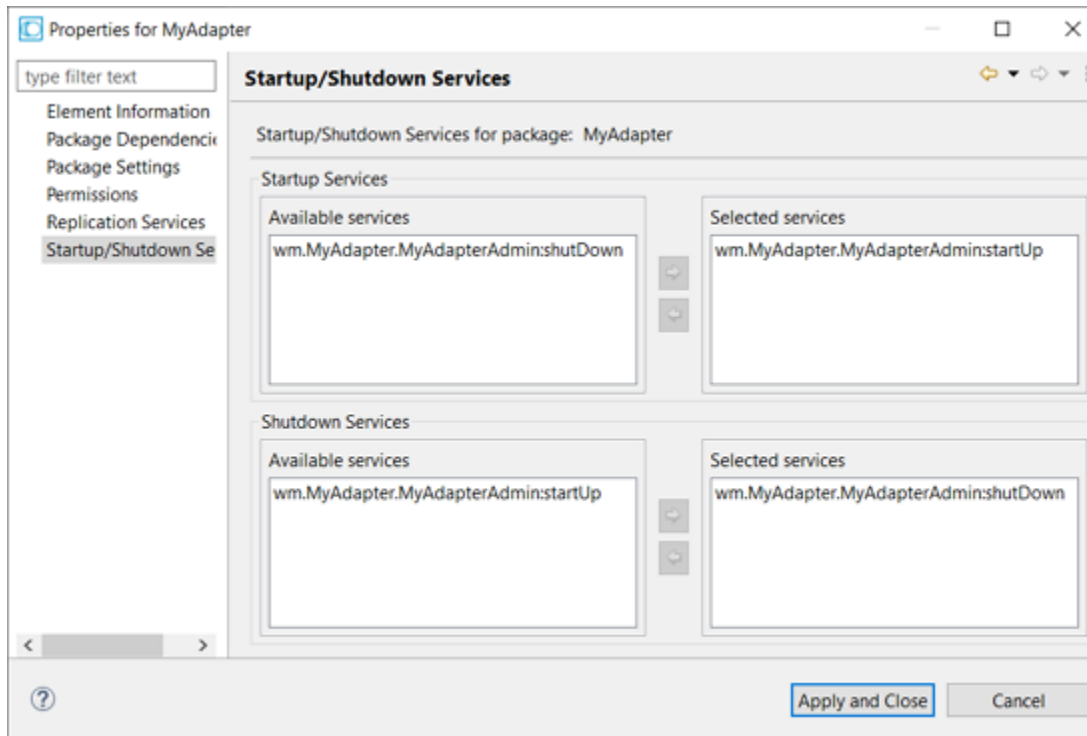


Registering the Adapter Startup and Shutdown Java Services in Integration Server

1. Start Designer.
2. In the **Package Navigator**, select the webMethods package you created.
3. Set the **Startup Services** and **Shutdown Services**.

Perform the following operations:

- In the **Properties > StartUp/Shutdown Services > Startup Services**, add the startup Java Service created.
- In the **Properties > StartUp/Shutdown Services > Shutdown Services**, add the shutdown Java Service created.

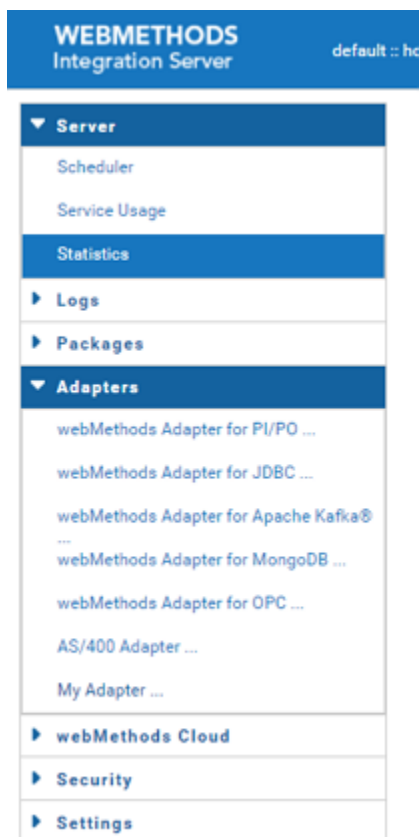


4. Restart Integration Server.

Testing the Adapter Registration

1. Start Integration Server Administrator.
2. In Integration Server Administrator, in the navigation, select **Adapters**.

The adapter you have created appears in the menu.



3. In the Integration Server Administrator, select **MyAdapter > About**.

The adapters' about page appears with the display name, description and copyright.

4. In the Integration Server Administrator, select **Packages > MyAdapter > Home**

The adapters' index page appears.

How to create an adapter connection implementation?

An adapter connection connects to an adapter resource. This chapter describes how to create an adapter connection implementation.

Pre-requisites:

- webMethods Integration Server 9.12 or later installed.
- Designer 9.12 or later installed.
- Integration Server Administrator access.
- Java 1.8 or later installed.
- Basic understanding of webMethods Integration Server, Designer, Integration Server Administrator, Java.

1. Start the editor to create Java files for adapter connection implementation.
2. Create directories corresponding to your Java package structure in the webMethods package you created using Designer. For example: `com\mycompany\adapter\myAdapter\connections`. In the example, the folder created is `com\wm\MyAdapter\connection`.

Note:

You must create your Java package and classes in the `adapterPackageName\code\source` folder in the webMethods package you created using Designer.

3. Create the following classes and interfaces:
 - a. Create a class by extending the `com.wm.adk.connection.WmManagedConnection` base class.

In the example, created *SimpleConnection* class:

```
package com.wm.MyAdapter.connections;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.*;
import com.wm.MyAdapter.MyAdapter;
public class SimpleConnection extends WmManagedConnection {
    String hostName;
    int port;
    public SimpleConnection(String hostNameValue, int portValue)
    {
        super();
        hostName = hostNameValue;
        port = portValue;
        MyAdapter.getInstance().getLogger().logDebug(9999,
            "Simple Connection created with hostName = "
            + hostName + "and port = " + Integer.toString(port));
    }
    public void destroyConnection()
    {
        MyAdapter.getInstance().getLogger().logDebug(9999,"Simple Connection
        Destroyed");
    }

    // The remaining methods support metadata for related services, etc.
    // Implement content as needed.
    public void registerResourceDomain(WmAdapterAccess access)
    {}
    public Boolean adapterCheckValue( String serviceName, String resourceDomainName,
        String[][] values, String testValue)
    { return null; }
    public ResourceDomainValues[] adapterResourceDomainLookup(String serviceName,

        String resourceDomainName, String[][] values)
    { return null; }
}
```

- b. Create a class by extending the `com.wm.adk.connection.WmManagedConnectionFactory` base class.

In the example, created *SimpleConnectionFactory* class:

```
package com.wm.MyAdapter.connections;
import com.wm.adk.connection.WmManagedConnectionFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.info.ResourceAdapterMetadataInfo;
import com.wm.adk.metadata.WmDescriptor;
import com.wm.adk.error.AdapterException;
import java.util.Locale;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.MyAdapterConstants;

public class SimpleConnectionFactory extends WmManagedConnectionFactory
implements MyAdapterConstants {
    private String hostName;
    private int port;
    public void setHostName(String hostNameValue){hostName = hostNameValue;}
    public void setPort(int portValue){port = portValue;}
    public SimpleConnectionFactory(){super();}
    public WmManagedConnection createManagedConnectionObject(
        javax.security.auth.Subject subject,
        javax.resource.spi.ConnectionRequestInfo cxRequestInfo)
    {
        return new SimpleConnection(hostName, port);
    }
    public void fillWmDescriptor(WmDescriptor d,Locale l) throws
        AdapterException
    {
        d.createGroup(GROUP_SIMPLE_CONNECTION,
            new String[]{SIMPLE_SERVER_HOST_NAME, SIMPLE_SERVER_PORT_NUMBER});
        d.setValidValues(SIMPLE_SERVER_PORT_NUMBER, new String[]
{"5555","1555","4000"});
        d.setDescriptions(
            MyAdapter.getInstance().getAdapterResourceBundleManager(),l);
    }
    public void fillResourceAdapterMetadataInfo(
        ResourceAdapterMetadataInfo info, Locale locale)
    {}
}
}
```

- c. Update the interface that contains the constants (you created for the adapter implementation) and add the connection constants:

In the example, update *MyAdapterConstants* interface:

```
package com.wm.MyAdapter;
import com.wm.MyAdapter.connections.SimpleConnectionFactory;
public interface MyAdapterConstants {

    static final int ADAPTER_MAJOR_CODE = 9009;
    static final String ADAPTER_JCA_VERSION = "1.0";
    static final String ADAPTER_NAME = "MyAdapter";
    static final String ADAPTER_VERSION = "9.12";

    // using next statement will create cyclic class loading dependency issue
    // therefore, the resource bundle class name is fully spelled out
    //static final String ADAPTER_SOURCE_BUNDLE_NAME =
MyAdapterResource.class.getName();
    static final String ADAPTER_SOURCE_BUNDLE_NAME =
```

```

"com.wm.MyAdapter.MyAdapterResource";

// added at Phase 2 to support connector
static final String CONNECTION_TYPE = SimpleConnectionFactory.class.getName();
// for all the properties, make sure the value matches the get/set method
// naming convention. you have to understand how a Java introspection
// build a property name using the names of the get and set methods
// added at Phase 2 to support connector
// connector properties
static final String GROUP_SIMPLE_CONNECTION = "SimpleServerConnection";
static final String SIMPLE_SERVER_HOST_NAME = "hostName";
static final String SIMPLE_SERVER_PORT_NUMBER = "port";
}

```

- d. Update the resource bundle implementation class to add the display name and description of the connection factory class and the fields in the connection factory class.

In the example, update *MyAdapterResource* class's `Object[][] _contents` as follows:

```

package com.wm.MyAdapter;
import java.util.ListResourceBundle;
import com.wm.adk.ADKGLOBAL;
import com.wm.MyAdapter.connections.SimpleConnectionFactory;

public class MyAdapterResource extends ListResourceBundle implements
MyAdapterConstants{
    static final String IS_PKG_NAME = "/MyAdapter/";
    static final Object[][] _contents = {
        // adapter type display name.
        {ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME, "MyAdapter"}
        // adapter type descriptions.
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Adapter for MyAdapter Server (a Sample System)"}
        // adapter type vendor.
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_VENDORNAME, "Software AG"}
        //Copyright URL Page
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_THIRDPARTYCOPYRIGHTURL,
IS_PKG_NAME + "copyright.html"}
        //Copyright Encoding
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_COPYRIGHTENCODING, "UTF-8"}
        //About URL Page
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_ABOUT, IS_PKG_NAME + "About.html"}
        //Release Notes URL Page
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_RELEASENOTEURL, IS_PKG_NAME +
"ReleaseNotes.html"}
        //SimpleConnection
        ,{SimpleConnectionFactory.class.getName() +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Simple Connection"}
        ,{SimpleConnectionFactory.class.getName() +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Simple framework for demonstration purposes"}
        ,{SimpleConnectionFactory.SIMPLE_SERVER_HOST_NAME +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Host Name"}
        ,{SimpleConnectionFactory.SIMPLE_SERVER_PORT_NUMBER +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Port"}
    }
}

```

```

};

protected Object[][] getContents() {
    // TODO Auto-generated method stub
    return _contents;
}
}

```

- e. Register the connection type to the adapter by updating your `fillAdapterTypeInfo` method in your `WmAdapter` implementation class.

In the example the *SimpleConnectionFactory* connection factory class is registered using the method `addConnectionFactory` in the adapter implementation class's *MyAdapter.fillAdapterTypeInfo* method:

```

package com.wm.MyAdapter;
..
..
..
import com.wm.MyAdapter.connections.SimpleConnectionFactory;
..
..
..
public class MyAdapter extends WmAdapter implements MyAdapterConstants {
..
..
..
    public void fillAdapterTypeInfo(AdapterTypeInfo info, Locale locale) {
        info.addConnectionFactory(SimpleConnectionFactory.class.getName());
    }
}

```

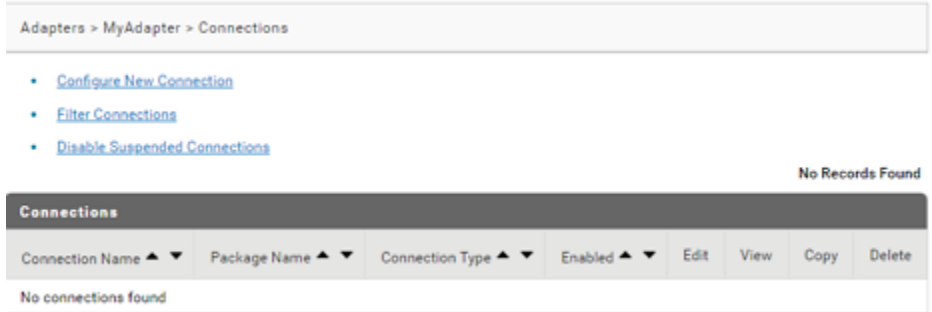
4. Execute the ANT script created in adapter definition to compile, and deploy the adapter in Integration Server.

Use the files `build.xml` and `build.properties`.

```
ant deploy
```

5. Restart Integration Server.
6. Start Integration Server Administrator.
7. In Integration Server Administrator, select **Adapters > MyAdapter**.

The adapters' connection configured are listed.



- In the **Adapters > MyAdapter > Connections** screen, select **Configure New Connections**.

The adapters' connection types are listed.



- In the **Adapters > My Adapter > Connection Types** screen, select the **Connection Type Simple Connection**.

The adapters' connection properties to configure are listed:

Adapters > MyAdapter > Configure Connection Type

- [Return to MyAdapter Connection Types](#)

Configure Connection Type > MyAdapter

Package	Default
Folder Name	<input type="text"/>
Connection Name	<input type="text"/>
Connection Properties	
Host Name	<input type="text"/>
Port	5555
Connection Management Properties	
Enable Connection Pooling	true
Minimum Pool Size	1
Maximum Pool Size	10
Pool Increment Size	1
Block Timeout (msec)	1000
Expire Timeout (msec)	1000
Startup Retry Count	0
Startup Backoff Timeout (sec)	10

10. In the **Adapters > My Adapter > Configure Connection Type** screen, add the details and **Save Connection**.

The **Adapters > My Adapter > Connections** screen lists the connection you added.

Adapters > MyAdapter > Connections

- [Configure New Connection](#)
- [Filter Connections](#)
- [Disable Suspended Connections](#)

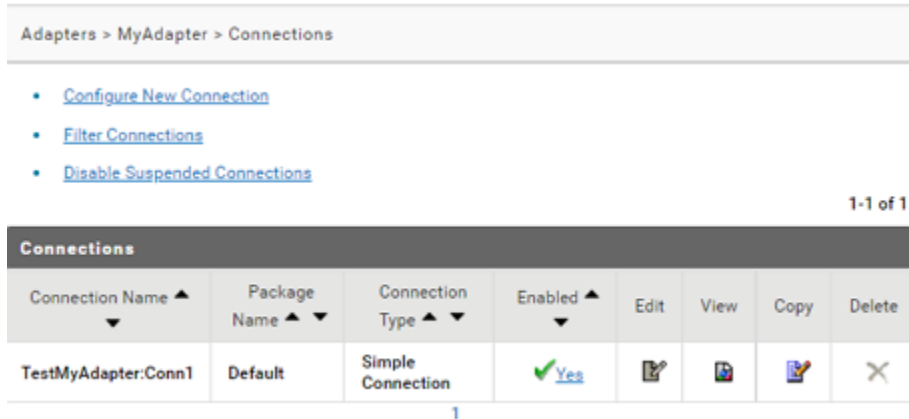
1-1 of 1

Connections							
Connection Name	Package Name	Connection Type	Enabled	Edit	View	Copy	Delete
TestMyAdapter:Conn1	Default	Simple Connection	No				

1

11. In the **Adapters > My Adapter > Connections** screen, click **No** in the **Enabled** column.

The **Enabled** column now shows **Yes**.



How to create an adapter service implementation?

An adapter service defines an operation that the adapter performs on an adapter resource. Adapter services operate like webMethods Integration Server flow services or Java services. Adapter services have input and output signatures, you call them within flow services, and you can audit them from the Integration Server's audit system.

Ensure that you have:

- webMethods Integration Server 9.12 or later installed.
- Designer 9.12 or later installed.
- Integration Server Administrator access.
- Java 1.8 or later installed.
- Basic understanding of webMethods Integration Server, Designer, Integration Server Administrator, Java.

How to create a basic adapter service implementation without any elements?

1. Start the editor to create Java files for the adapter service implementation.
2. Create directories corresponding to your Java package structure in the webMethods package you created using Designer. For example: `com\mycompany\adapter\myAdapter\services`. In the example, the folder created is `com\wm\MyAdapter\services`.

Note:

You should create your Java package and classes in the `adapterPackageName\code\source` folder in the webMethods package you created using Designer.

3. Create a class by extending the `com.wm.adk.cci.interaction.WmAdapterService` base class.

In the example, created *MockDbUpdate* class:

```
package com.wm.MyAdapter.services;
import com.wm.adk.cci.interaction.WmAdapterService;
import com.wm.adk.cci.record.WmRecord;
import com.wm.adk.cci.record.WmRecordFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.WmTemplateDescriptor;
import java.util.Hashtable;
import java.util.Locale;
import javax.resource.ResourceException;
import com.wm.MyAdapter.MyAdapter;

public class MockDbUpdate extends WmAdapterService {
    public void fillWmTemplateDescriptor(WmTemplateDescriptor d,Locale l)
        throws ResourceException {
    }
    public WmRecord execute(WmManagedConnection connection, WmRecord input)
        throws ResourceException {
        return null;
    }
}
```

4. Update the resource bundle implementation class to add the display name and description of the adapter service.

In the example, updated *MyAdapterResource* class:

```
package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.services.MockDbUpdate;
public class MyAdapterResource extends ListResourceBundle implements
MyAdapterConstants{
    ..
    ..
    static final Object[][] _contents = {
    ..
    ..
        //Adapter Services
        ,{MockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Mock Update Service"}
        ,{MockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Simulates a database update service"}
    }
    protected Object[][] getContents() {
        // TODO Auto-generated method stub
        return _contents;
    }
}
```

5. Register the adapter service by updating your *fillResourceAdapterMetadataInfo* method in your *WmManagedConnectionFactory* connection factory implementation class.

In the example, the *MockDbUpdate* class is registered using *fillResourceAdapterMetadataInfo* method in the *SimpleConnectionFactory* connection factory class:

```
package com.wm.MyAdapter.connections;
```

```

import com.wm.adk.connection.WmManagedConnectionFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.info.ResourceAdapterMetadataInfo;
import com.wm.adk.metadata.WmDescriptor;
import com.wm.adk.error.AdapterException;
import java.util.Locale;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.MyAdapterConstants;
import com.wm.MyAdapter.services.MockDbUpdate;
public class SimpleConnectionFactory extends WmManagedConnectionFactory implements
    MyAdapterConstants {
    ..
    ..
    ..
    public void fillResourceAdapterMetadataInfo(ResourceAdapterMetadataInfo info,
        Locale locale) {
        info.addServiceTemplate(MockDbUpdate.class.getName());
    }
}

```

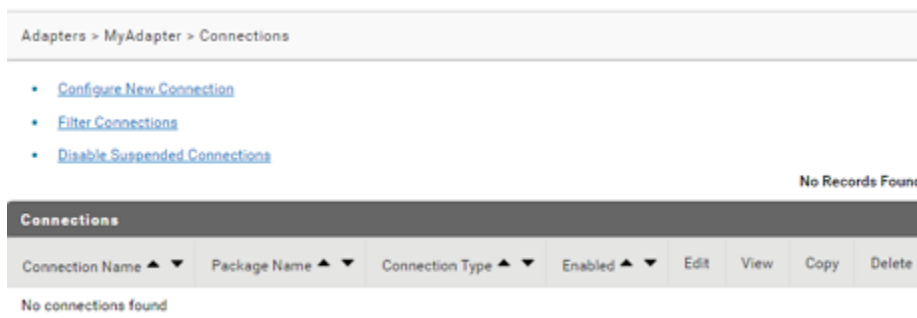
- Execute the ANT script created in the adapter definition to compile, and deploy the adapter in Integration Server.

Use the files `build.xml` and `build.properties`.

```
ant deploy
```

- Restart Integration Server.
- Start Integration Server Administrator.
- Select **Adapters > MyAdapter > Connections**.

The screen lists the adapters' connection configured:



- In the **Adapters > MyAdapter > Connections** screen, select **Configure New Connection**.

The screen lists the adapters' connection types:

Adapters > MyAdapter > Connection Types

- [Return to MyAdapter Connections](#)

Connection Types	
Connection Type	Description
Simple Connection	Simple framework for demonstration purposes

11. In the **Adapters > MyAdapter > Connection Types** screen, select the connection type **Simple Connection**.

The screen lists the adapters' connection properties to configure:

Adapters > MyAdapter > Configure Connection Type

- [Return to MyAdapter Connection Types](#)

Configure Connection Type > MyAdapter

Package	Default ▾
Folder Name	<input type="text"/>
Connection Name	<input type="text"/>
Connection Properties	
Host Name	<input type="text"/>
Port	5555 ▾
Connection Management Properties	
Enable Connection Pooling	true ▾
Minimum Pool Size	<input type="text" value="1"/>
Maximum Pool Size	<input type="text" value="10"/>
Pool Increment Size	<input type="text" value="1"/>
Block Timeout (msec)	<input type="text" value="1000"/>
Expire Timeout (msec)	<input type="text" value="1000"/>
Startup Retry Count	<input type="text" value="0"/>
Startup Backoff Timeout (sec)	<input type="text" value="10"/>
<input type="button" value="Save Connection"/> <input type="button" value="Test Connection"/>	





12. In the **Adapters > MyAdapter > Configure Connection Type** screen, add the details and click **Save Connection**.

The **Adapters > MyAdapter > Connections** screen lists the connection you added.

Adapters > MyAdapter > Connections

- [Configure New Connection](#)
- [Filter Connections](#)
- [Disable Suspended Connections](#)

1-1 of 1

Connections							
Connection Name ▲ ▼	Package Name ▲ ▼	Connection Type ▲ ▼	Enabled ▲ ▼	Edit	View	Copy	Delete
TestMyAdapter:Conn1	Default	Simple Connection	No				

1






13. In the **Adapters > MyAdapter > Connections** screen, click **No** in the **Enabled** column for the connection you added.

The **Enabled** column now shows **Yes**.

Adapters > MyAdapter > Connections

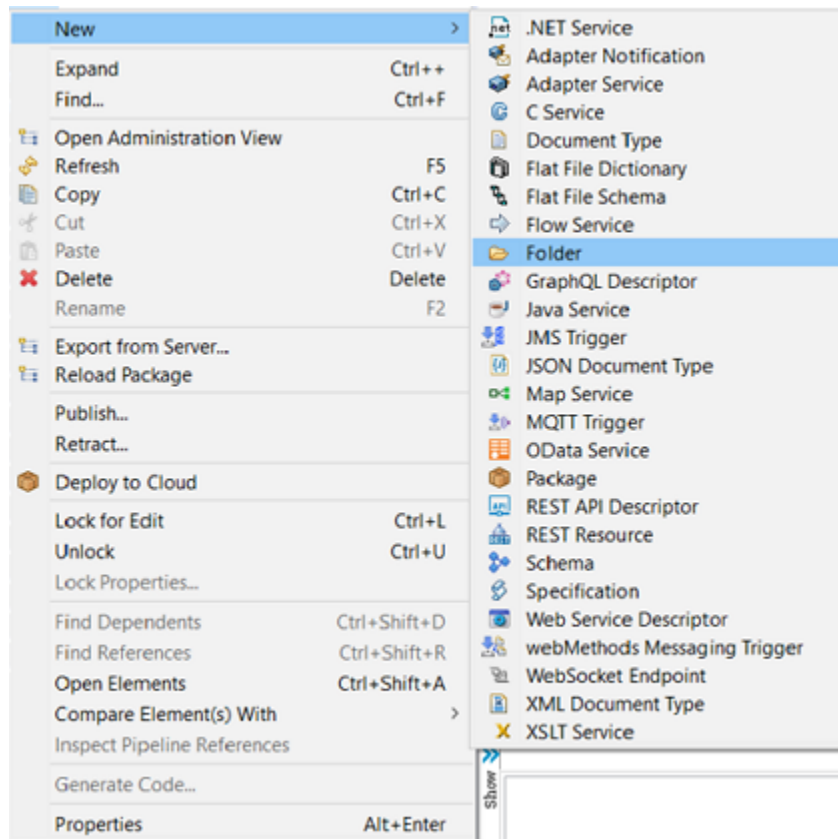
- [Configure New Connection](#)
- [Filter Connections](#)
- [Disable Suspended Connections](#)

1-1 of 1

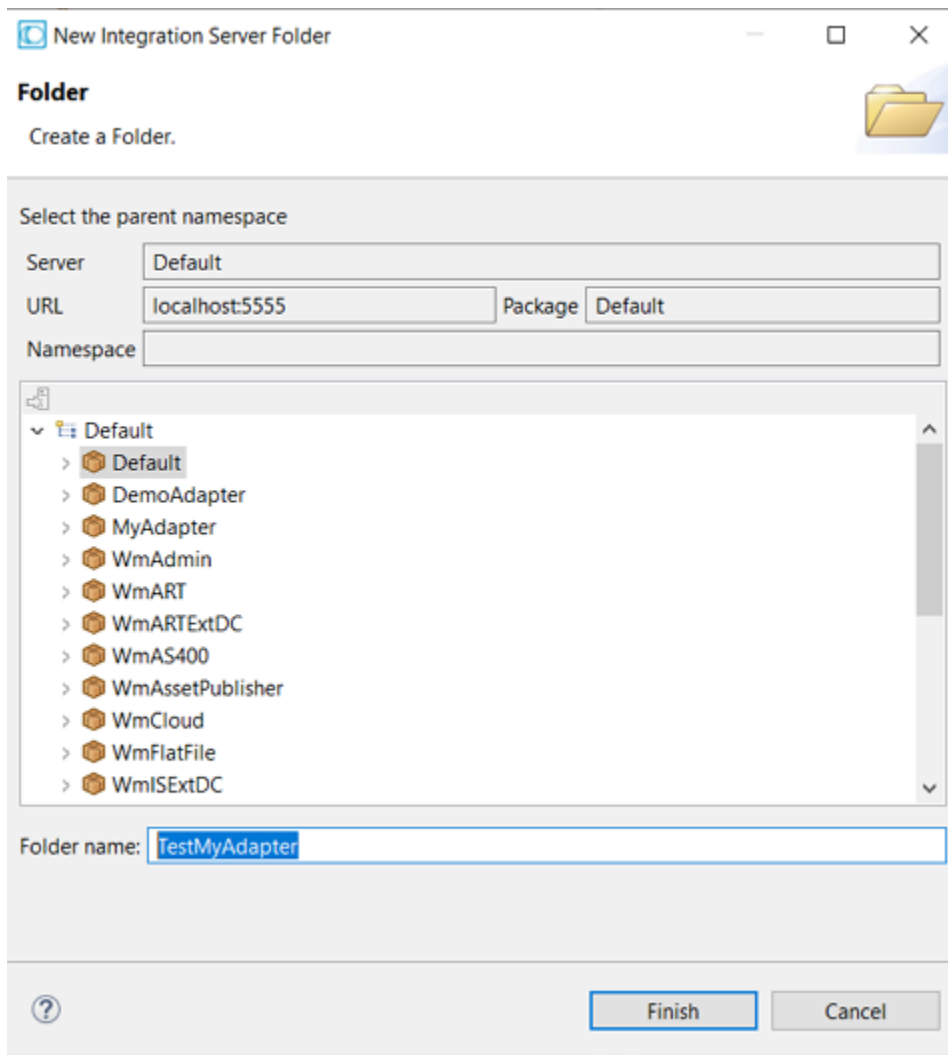
Connections							
Connection Name ▲ ▼	Package Name ▲ ▼	Connection Type ▲ ▼	Enabled ▲ ▼	Edit	View	Copy	Delete
TestMyAdapter:Conn1	Default	Simple Connection	 Yes				

1

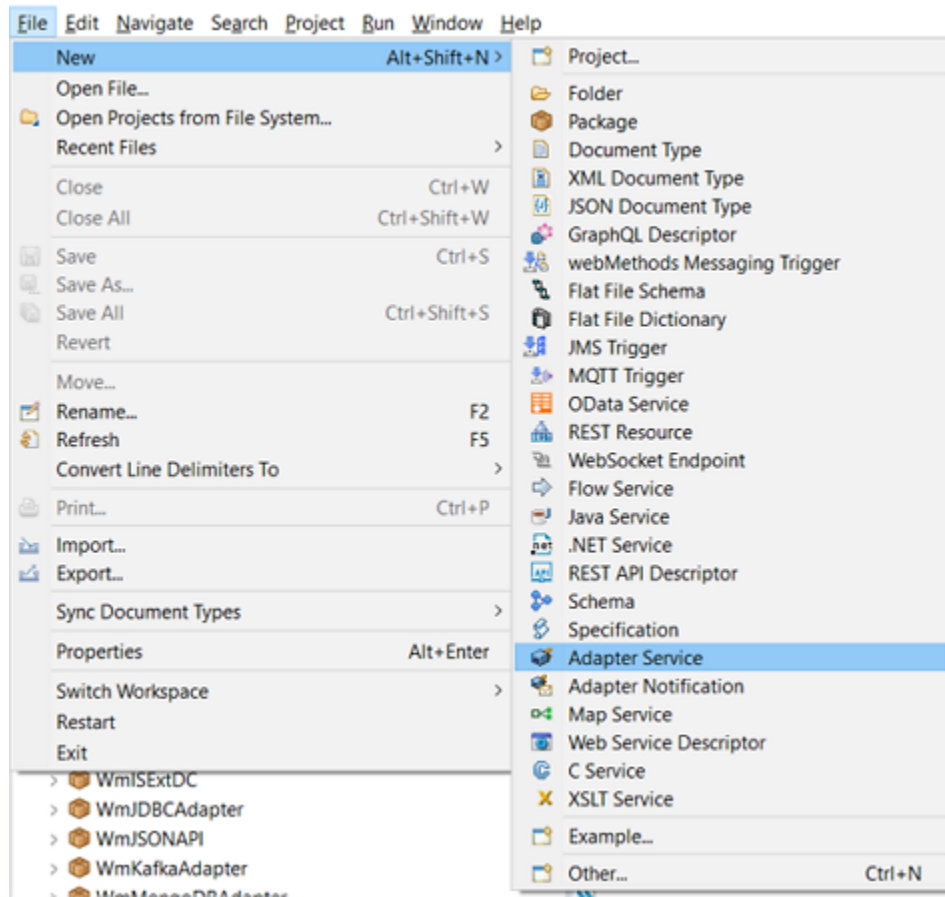
14. In Designer, create the adapter service.
- In **Package Navigator**, select the **Default** package.
 - Select **File > New > Folder**.



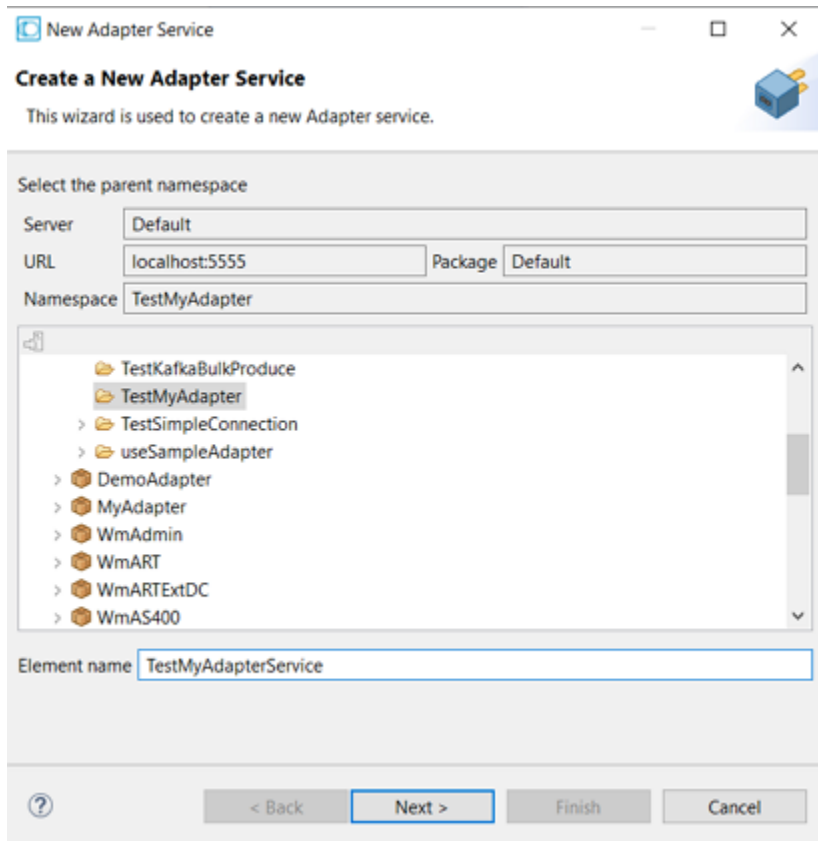
- c. Enter the **Folder name**. For example: *TestMyAdapter*.



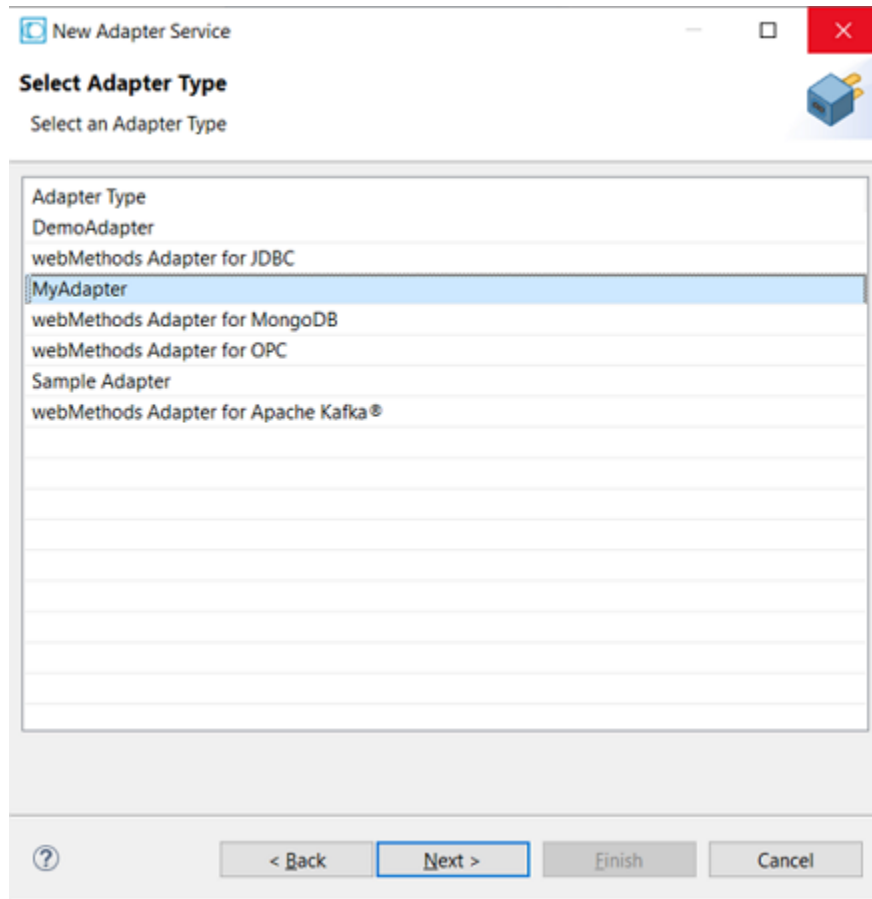
- d. In the **Package Navigator**, select the **Default > TestMyAdapter**.
- e. Select **File > New > Adapter Service**.



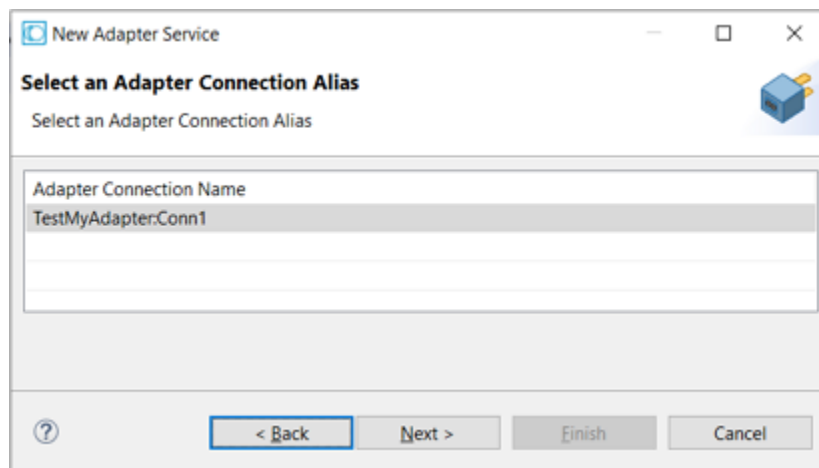
- f. Enter the **Element name** and click **Next**. For example: *TestMyAdapterService*.



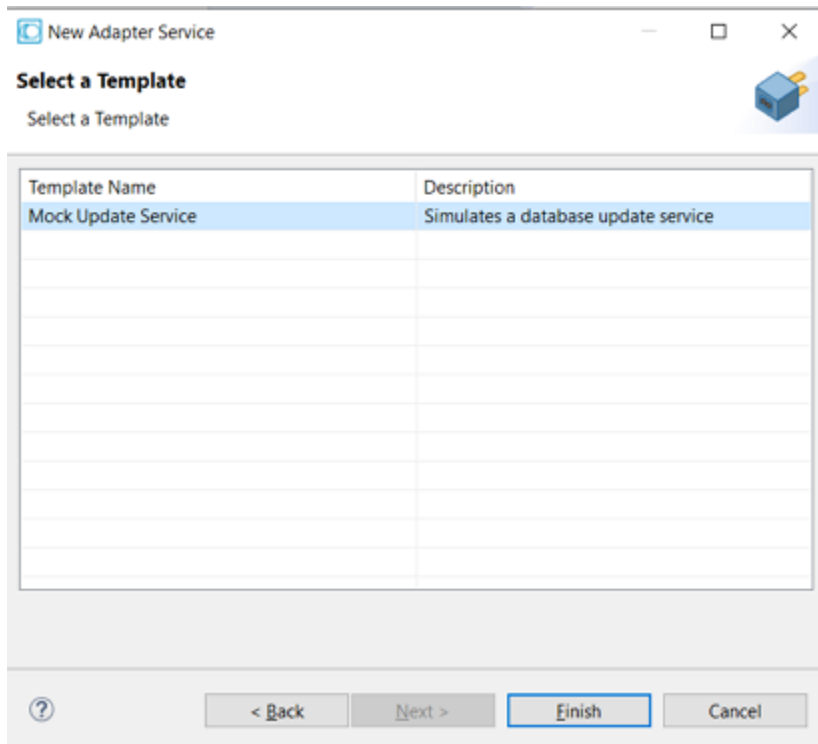
- g. In the **Select Adapter Type**, select an adapter type for which you want to create the service. For example: *MyAdapter*.



- h. In the **Select an Adapter Connection Alias**, select an adapter connection. For example: ***TestMyAdapter:Conn1***.

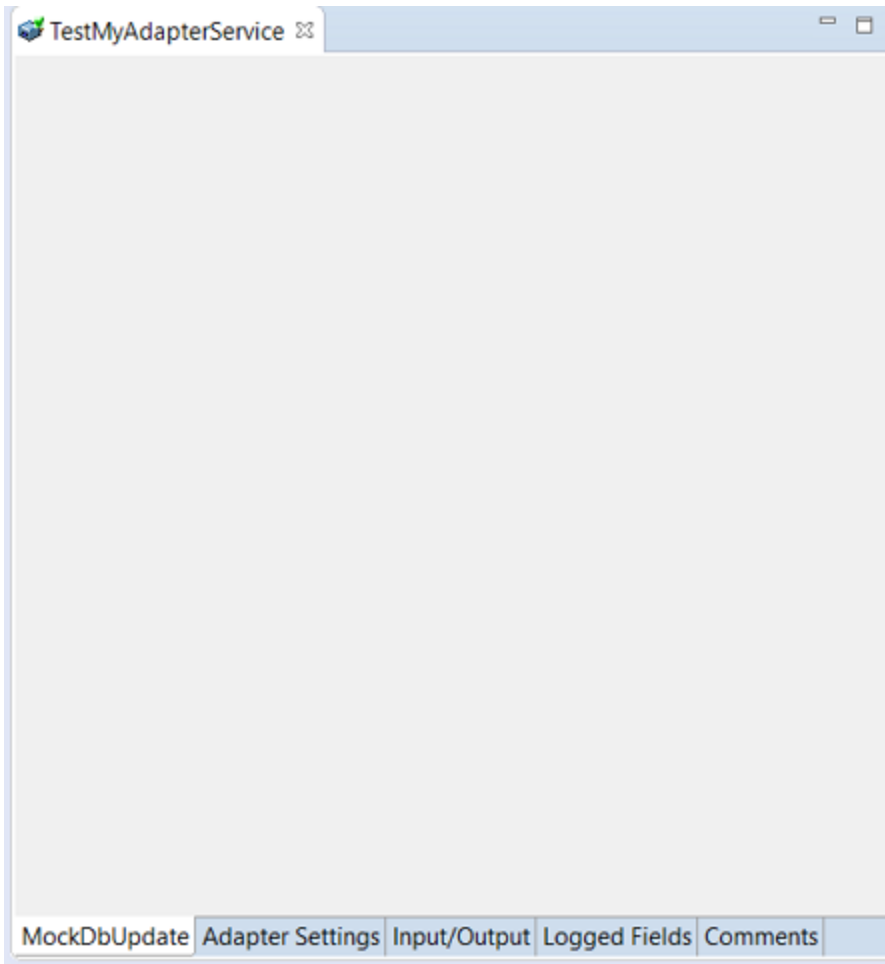


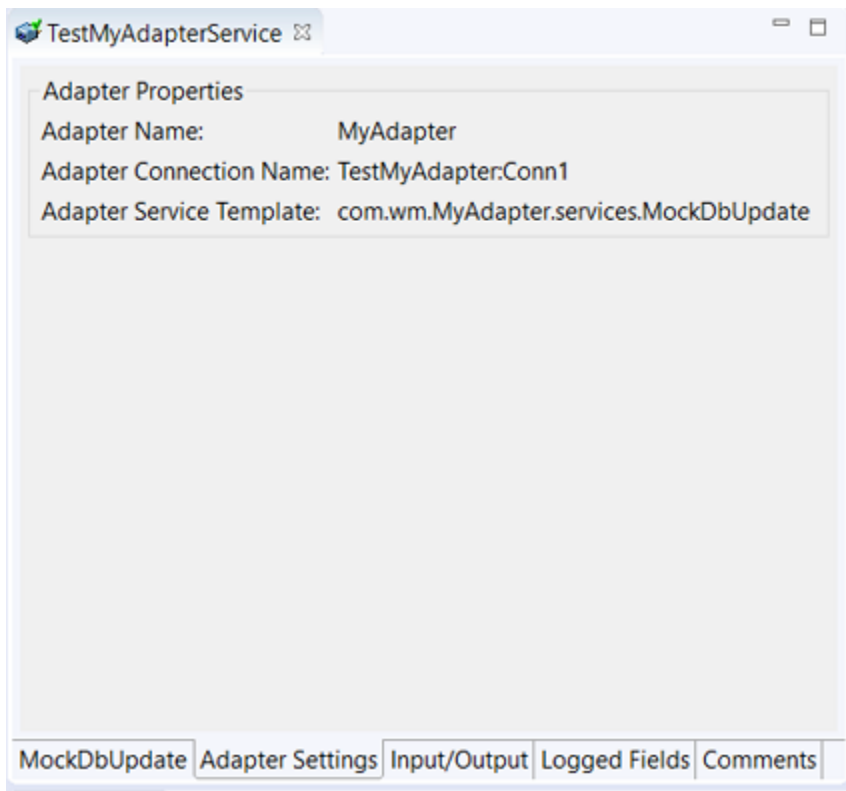
- i. In the **Select a Template**, select an adapter template and click **Finish**. For example: ***Mock Update Service***.

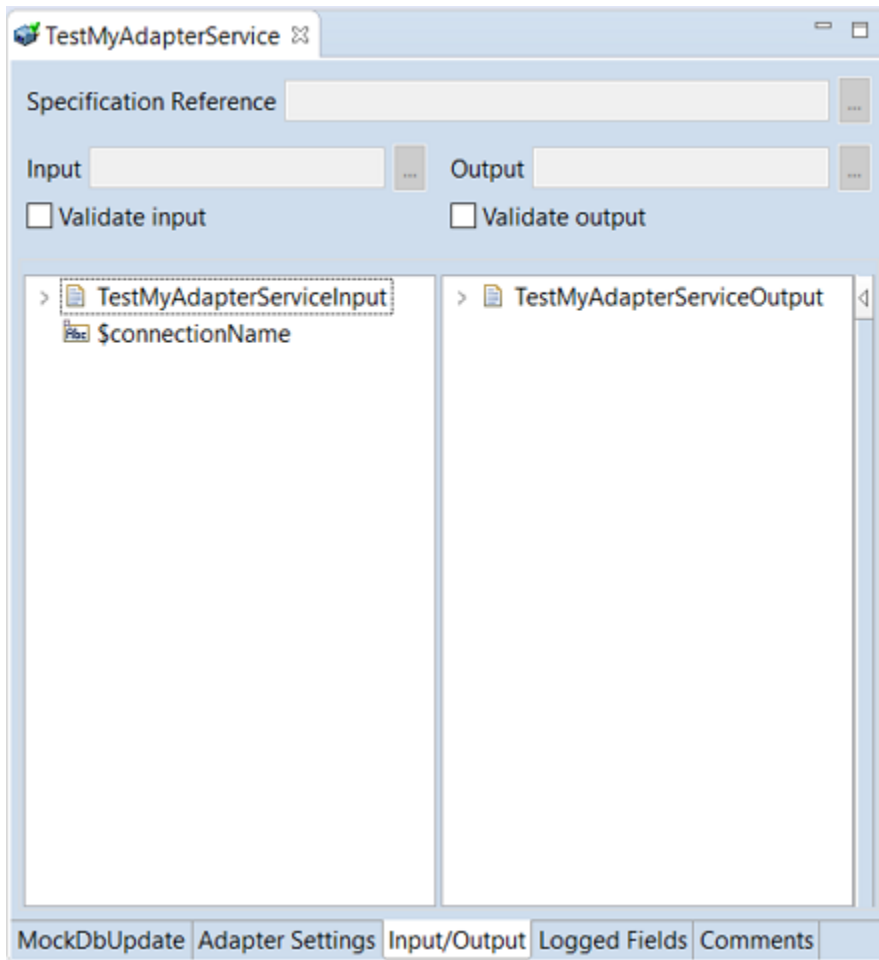


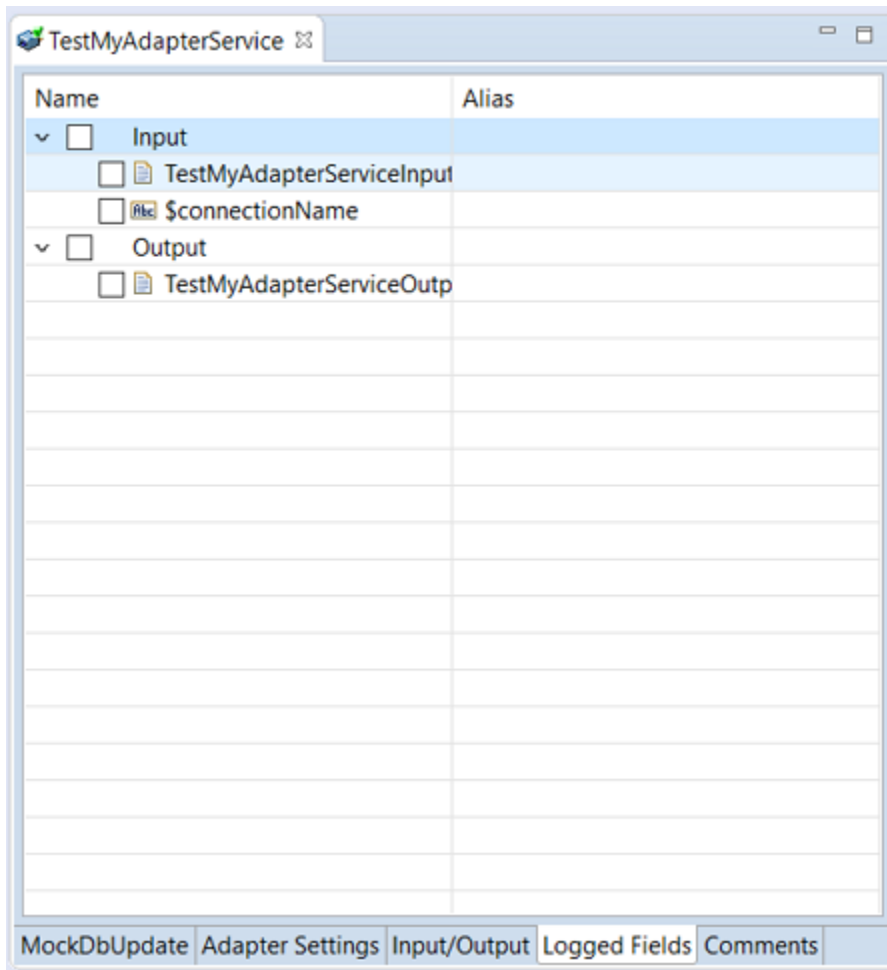
A new adapter service of type *MyAdapter* is created. You will see the following tabs:

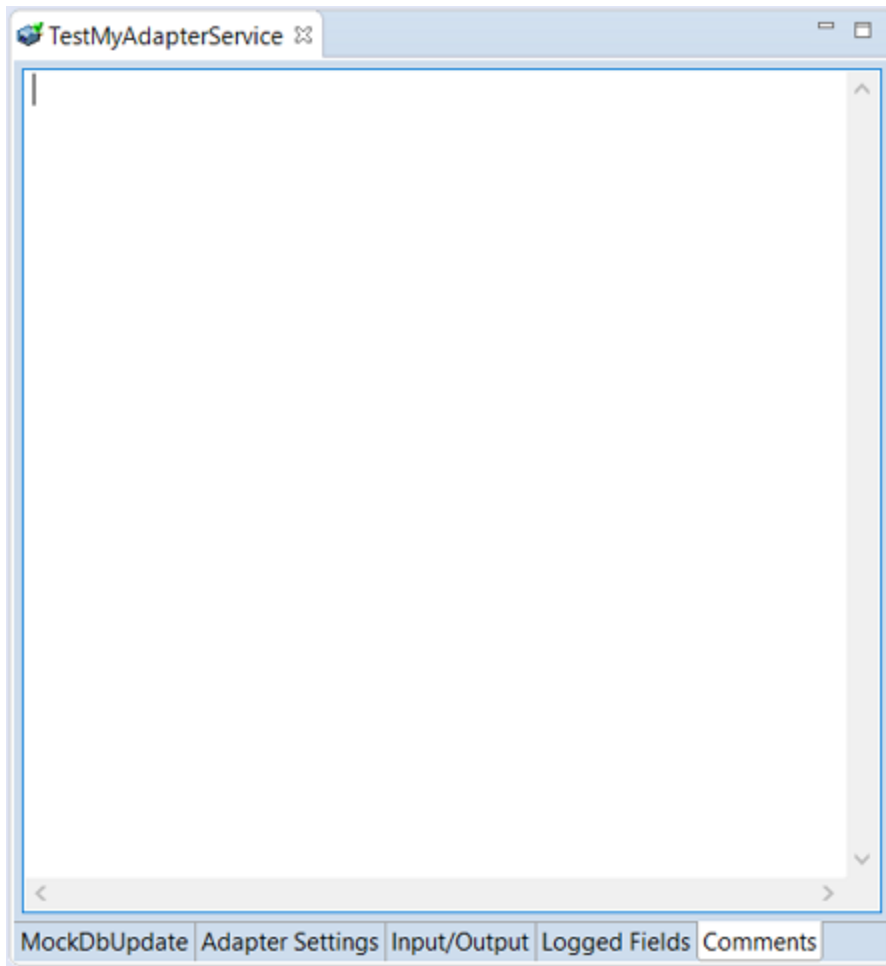
1. *MockDbUpdate*.
2. Adapter Settings
3. Input/Output
4. Logged Fields
5. Comments











How to display elements in an adapter service?

1. Update the `com.wm.adk.cci.interaction.WmAdapterService` implementation class.
 - a. Create a class attribute, and a corresponding set method for each metadata parameter. For example, a class attribute `_tableName`, and a corresponding set method `setTableName` for the parameter table name.

Class Attribute Name	Class Attribute Set Method Name
<code>_tableName</code>	<code>setTableName</code>
<code>_columnNames</code>	<code>setColumnNames</code>
<code>_columnTypes</code>	<code>setColumnTypes</code>
<code>_repeating</code>	<code>setRepeating</code>
<code>_overrideTypes</code>	<code>setOverrideTypes</code>

For example class *MockDbUpdate*:

```
package com.wm.MyAdapter.services;
import com.wm.adk.cci.interaction.WmAdapterService;
import com.wm.adk.cci.record.WmRecord;
import com.wm.adk.cci.record.WmRecordFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.WmTemplateDescriptor;
import java.util.Hashtable;
import java.util.Locale;
import javax.resource.ResourceException;
import com.wm.MyAdapter.MyAdapter;

public class MockDbUpdate extends WmAdapterService {
    private String _tableName;
    private String[] _columnNames;
    private String[] _columnTypes;
    private boolean _repeating;
    private String[] _overrideTypes;
    public static final String TABLE_NAME_PARM = "tableName";
    public static final String COLUMN_NAMES_PARM = "columnNames";
    public static final String COLUMN_TYPES_PARM = "columnTypes";
    public static final String REPEATING_PARM = "repeating";
    public static final String OVERRIDE_TYPES_PARM = "overrideTypes";
    public void setTableName(String val){ _tableName = val;}
    public void setColumnNames(String[] val){ _columnNames = val;}
    public void setColumnTypes(String[] val){ _columnTypes = val;}
    public void setRepeating(boolean val){ _repeating = val;}
    public void setOverrideTypes(String[] val){_overrideTypes = val;}

    public void fillWmTemplateDescriptor(WmTemplateDescriptor d,Locale l)
        throws ResourceException
    { }
    public WmRecord execute(WmManagedConnection connection, WmRecord input)
        throws ResourceException
    {
        return null;
    }
}
```

2. Update the resource bundle implementation class to add the display name and description of the adapter service.

In the example, the class is *MyAdapterResource*:

```
package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.services.MockDbUpdate;
public class MyAdapterResource extends ListResourceBundle implements
MyAdapterConstants{
    ..
    ..
    static final Object[][] _contents = {
    ..
    ..
    //Adapter Services
    ,{MockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Mock Update Service"}
```



```

    ,{MockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
      "Simulates a database update service"}
  }
  protected Object[][] getContents() {
    // TODO Auto-generated method stub
    return _contents;
  }
}

```

3. Register the adapter service by updating your `fillResourceAdapterMetadataInfo` method in your `WmManagedConnectionFactory` implementation class.

In the example, the adapter service `MockDbUpdate` is registered using the method `fillResourceAdapterMetadataInfo` in the connection factory class `SimpleConnectionFactory` class:

```

package com.wm.MyAdapter.connections;
import com.wm.adk.connection.WmManagedConnectionFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.info.ResourceAdapterMetadataInfo;
import com.wm.adk.metadata.WmDescriptor;
import com.wm.adk.error.AdapterException;
import java.util.Locale;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.MyAdapterConstants;
import com.wm.MyAdapter.services.MockDbUpdate;
public class SimpleConnectionFactory extends WmManagedConnectionFactory implements
  MyAdapterConstants {
  ..
  ..
  ..
  public void fillResourceAdapterMetadataInfo(ResourceAdapterMetadataInfo info,
  Locale locale) {
    info.addServiceTemplate(MockDbUpdate.class.getName());
  }
}

```

4. Execute the ANT script created in the adapter definition to compile, and deploy the adapter in Integration Server.


Use the files `build.xml` and `build.properties`.


```
ant deploy
```

5. Restart Integration Server.
6. Start Integration Server Administrator.
7. In Designer, create the adapter service.


A new adapter service of type `MyAdapter` is created with the metadata parameters as shown in the `MockDBUpdate` tab below:

tableName

 columnNames

 overrideTypes

repeating

 columnTypes

How to group elements, display data in tables and perform lookups in an adapter service?

1. Update the `com.wm.adk.cci.interaction.WmAdapterService` implementation class.
 - a. Create a class attribute, a set method, a constant, and a resource domain for each metadata parameter. For example, a class attribute `_tableName`, a set method `setTableName`, a constant `TABLE_NAME_PARM`, and a resource domain `TABLES_RD` (which needs to lookup) for the parameter table name.

Class Attribute Name	Class Attribute Set Method Name	Class Attribute Name Constant	Resource Domain Name Constant
<i>_tableName</i>	setTableName	TABLE_NAME_PARM	TABLES_RD
<i>_columnNames</i>	setColumnNames	COLUMN_NAMES_PARM	COLUMN_NAMES_RD
<i>_columnTypes</i>	setColumnTypes	COLUMN_TYPES_PARM	COLUMN_TYPES_RD
<i>_repeating</i>	setRepeating	REPEATING_PARM	None
<i>_overrideTypes</i>	setOverrideTypes	OVERRIDE_TYPES_PARM	OVERRIDE_TYPES_RD

- b. In the fillWmTemplateDescriptor method, call WmTemplateDescriptor.createGroup method, WmTemplateDescriptor.createFieldMap method, and WmTemplateDescriptor.createTuple method.
- c. In the fillWmTemplateDescriptor method, call WmTemplateDescriptor.setResourceDomain method, and WmTemplateDescriptor.setDescriptions method.
- d. Create the private Hashtable[] unpackRequest(WmRecord request) method to process and convert the webMethods's datatype WmRecord to Hashtable[].
- e. Create the private WmRecord packResonse(Hashtable[] response) method to process and convert the Hashtable[] to webMethods's datatype WmRecord.
- f. In the execute method, call unpackRequest method, and packResonse method.

For example, *MockDbUpdate* class :

```
package com.wm.MyAdapter.services;
import com.wm.adk.cci.interaction.WmAdapterService;
import com.wm.adk.cci.record.WmRecord;
import com.wm.adk.cci.record.WmRecordFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.WmTemplateDescriptor;
import com.wm.data.IData;
import com.wm.data.IDataCursor;
import com.wm.data.IDataFactory;
import com.wm.data.IDataUtil;
import java.util.Hashtable;
import java.util.Locale;
import javax.resource.ResourceException;
import com.wm.MyAdapter.MyAdapter;

public class MockDbUpdate extends WmAdapterService {
    //MockDB Group
    public static final String UPD_SETTINGS_GRP = "Mock Settings";
    public static final String TABLE_NAME_PARM = "tableName";
    public static final String COLUMN_NAMES_PARM = "columnNames";
    public static final String COLUMN_TYPES_PARM = "columnTypes";
    public static final String REPEATING_PARM = "repeating";
```

```

public static final String OVERRIDE_TYPES_PARM = "overrideTypes";
private String _tableName;
private String[] _columnNames;
private String[] _columnTypes;
private boolean _repeating;
private String[] _overrideTypes;
public void setTableName(String val){ _tableName = val;}
public void setColumnNames(String[] val){ _columnNames = val;}
public void setColumnTypes(String[] val){ _columnTypes = val;}
public void setRepeating(boolean val){ _repeating = val;}
public void setOverrideTypes(String[] val){_overrideTypes = val;}
public static final String TABLES_RD = "tablesRD";
public static final String COLUMN_NAMES_RD = "columnNamesRD";
public static final String COLUMN_TYPES_RD = "columnTypesRD";
public static final String OVERRIDE_TYPES_RD = "overrideTypesRD";

```

```

public void fillWmTemplateDescriptor(WmTemplateDescriptor d,Locale l)
throws ResourceException {
//MockDB Grouping and resource domain setup
d.createGroup(UPD_SETTINGS_GRP, new String [] {
TABLE_NAME_PARM,
REPEATING_PARM,
COLUMN_NAMES_PARM,
COLUMN_TYPES_PARM,
OVERRIDE_TYPES_PARM}
);
d.createFieldMap(new String[] {
COLUMN_NAMES_PARM,
COLUMN_TYPES_PARM,
OVERRIDE_TYPES_PARM},
true);
d.createTuple(new String[]{COLUMN_NAMES_PARM,COLUMN_TYPES_PARM});

d.setResourceDomain(TABLE_NAME_PARM, TABLES_RD, null);
d.setResourceDomain(COLUMN_NAMES_PARM, COLUMN_NAMES_RD,
new String[]{TABLE_NAME_PARM});
d.setResourceDomain(COLUMN_TYPES_PARM, COLUMN_TYPES_RD,
new String[]{TABLE_NAME_PARM});
d.setResourceDomain(OVERRIDE_TYPES_PARM, OVERRIDE_TYPES_RD, null);
//Call to setDescription
d.setDescriptions(MyAdapter.getInstance().
getAdapterResourceBundleManager(),l);
}
public WmRecord execute(WmManagedConnection connection, WmRecord input)
throws ResourceException {
Hashtable[] request = this.unpackRequest(input);
return this.packResonse(request);
}
private Hashtable[] unpackRequest(WmRecord request) throws ResourceException {
Hashtable data[] = null;
IData mainIData = request.getIData();
IDataCursor mainCursor = mainIData.getCursor();
try {
String tableName = this._tableName;
String[] columnNames = this._columnNames;
if(mainCursor.first(tableName)) {
IData[] recordIData;
if(this._repeating) {
recordIData = IDataUtil.getIDataArray (mainCursor,tableName);
data = new Hashtable[recordIData.length];
}
}
}
}

```

```

    }
    else {
        recordIData = new IData[] {IDataUtil.getIData(mainCursor)};
        data = new Hashtable[1];
    }
    for(int rec=0;rec<recordIData.length;rec++) {
        IDataCursor recordCursor = recordIData[rec].getCursor();
        data[rec] = new Hashtable();
        for(int c = 0; c < columnNames.length;c++) {
            if(recordCursor.first(columnNames[c])) {
                data[rec].put(tableName + "." + columnNames[c],
                    recordCursor.getValue());
            }
        }
        recordCursor.destroy();
    }
}
else {
    throw MyAdapter.getInstance().createAdapterException(9999,
        new String[] {"No Request Data"});
}
}
catch (Throwable t) {
    throw MyAdapter.getInstance().createAdapterException(9999,
        new String[] {"Error unpacking request data"},t);
}
finally {
    mainCursor.destroy();
}
return data;
}
private WmRecord packResonse(Hashtable[] response) throws ResourceException {
    WmRecord data = null;
    try {
        IData[] recordIData = new IData[response.length];
        String tableName = this._tableName;
        String[] columnNames = this._columnNames;
        for(int rec = 0; rec < response.length; rec++) {
            recordIData[rec] = IDataFactory.create();
            IDataCursor recordCursor = recordIData[rec].getCursor();
            for(int col = 0; col < columnNames.length;col++) {
                IDataUtil.put(recordCursor,columnNames[col],
                    response[rec].get(tableName + "." +
                    columnNames[col]));
            }
            recordCursor.destroy();
        }
        IData mainIData = IDataFactory.create();
        IDataCursor mainCursor = mainIData.getCursor();
        if(this._repeating) {
            IDataUtil.put(mainCursor,tableName,recordIData);
        }
        else {
            IDataUtil.put(mainCursor,tableName,recordIData[0]);
        }
        mainCursor.destroy();
        data = WmRecordFactory.getFactory().createWmRecord("nameNotUsed");
        data.setIData(mainIData);
    }
    catch (Throwable t) {

```

```

    throw MyAdapter.getInstance().createAdapterException(9999,
        new String[] {"Error packing response data"},t);
    }
    return data;
    }
}

```

2. Update the WmManagedConnection implementation class.

- Add the data for the mock tables in the adapter service.
- Add the methods `adapterCheckValue`, `adapterResourceDomainLookup`, and `registerResourceDomain`

For example, *SimpleConnection* class:

```

package com.wm.MyAdapter.connections;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.*;
import com.wm.adk.error.AdapterException;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.services.MockDbUpdate;
public class SimpleConnection extends WmManagedConnection {
    String hostName;
    int port;

    //Adapter Services variables
    private String[] mockTableNames = { "CUSTOMERS","ORDERS","LINE_ITEMS"};
    private String[][] mockColumnNames = {
        {"name","id", "ssn"},
        {"id","date","customer_id"},
        {"order_id","item_number","quantity","description"}
    };
    private String [][] mockDataTypes = {
        {"java.lang.String","java.lang.Integer", "java.lang.String"},
        {"java.lang.Integer", "java.util.Date", "java.lang.Integer"},
        {"java.lang.Integer", "java.lang.Integer", "java.lang.Integer",
        "java.lang.String"}
    };
    public SimpleConnection(String hostNameValue, int portValue)
    {
    ..
    ..
    }
    public void destroyConnection()
    {
    ..
    ..
    }

    // The remaining methods support metadata for related services, etc.
    // Implement content as needed.
    public Boolean adapterCheckValue(String serviceName,
        String resourceDomainName,
        String[][] values,
        String testValue) throws AdapterException
    {
        Boolean result = new Boolean(false);
        if(resourceDomainName.equals(MockDbUpdate.OVERRIDE_TYPES_RD))
        {

```

```

    try
    {
        Object o = Class.forName(testValue).getConstructor(
            new Class[] {String.class}).newInstance(new Object[]{"0"});
        result = new Boolean(true);
    }
    catch (Throwable t){}
}
return result;
}
public ResourceDomainValues[] adapterResourceDomainLookup(String serviceName,
String resourceDomainName, String[][] values) throws AdapterException
{
    ResourceDomainValues[] results = null;

    //MockDB Group Lookup
    if(resourceDomainName.equals(MockDbUpdate.COLUMN_NAMES_RD) ||
        resourceDomainName.equals(MockDbUpdate.COLUMN_TYPES_RD))
    {
        String tableName = values[0][0];
        for(int x = 0; x < this.mockTableNames.length;x++)
        {
            if(this.mockTableNames[x].equals(tableName))
            {
                ResourceDomainValues columnsRdvs = new ResourceDomainValues(
                    MockDbUpdate.COLUMN_NAMES_RD,this.mockColumnNames[x]);
                columnsRdvs.setComplete(true);
                ResourceDomainValues typesRdvs = new ResourceDomainValues(
                    MockDbUpdate.COLUMN_TYPES_RD, this.mockDataTypes[x]);
                typesRdvs.setComplete(true);
                results = new ResourceDomainValues[] {columnsRdvs,typesRdvs};
                break;
            }
        }
    }

    return results;
}
public void registerResourceDomain(WmAdapterAccess access)
throws AdapterException
{
    //MockDB Group Registering Resource Domain
    ResourceDomainValues tableRdvs = new ResourceDomainValues(
        MockDbUpdate.TABLES_RD, mockTableNames);
    tableRdvs.setComplete(true);
    access.addResourceDomain(tableRdvs);
    access.addResourceDomainLookup(MockDbUpdate.COLUMN_NAMES_RD,this);
    access.addResourceDomainLookup(MockDbUpdate.COLUMN_TYPES_RD,this);
    ResourceDomainValues rdvs = new ResourceDomainValues(
        MockDbUpdate.OVERRIDE_TYPES_RD, new String[] {""});
    rdvs.setComplete(false);
    rdvs.setCanValidate(true);
    access.addResourceDomain(rdvs);
    access.addCheckValue(MockDbUpdate.OVERRIDE_TYPES_RD,this);
}
}

```

3. Update the resource bundle implementation class to add the display name and description of the adapter service.

In the example, updated *MyAdapterResource* class:

```
package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.services.MockDbUpdate;
public class MyAdapterResource extends ListResourceBundle implements
MyAdapterConstants{
..
..
static final Object[][] _contents = {
..
..
//Adapter Services
,{MockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
"Mock Update Service"}
,{MockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
"Simulates a database update service"}
//MockDB Group Resource Domain Values
,{MockDbUpdate.UPD_SETTINGS_GRP + ADKGLOBAL.RESOURCEBUNDLEKEY_GROUP,
MockDbUpdate.UPD_SETTINGS_GRP}
,{MockDbUpdate.TABLE_NAME_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME, "Table Name"}
,{MockDbUpdate.TABLE_NAME_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION, "Select Table Name"}
,{MockDbUpdate.COLUMN_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
"Column Names"}
,{MockDbUpdate.COLUMN_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
"Name of column updated by this service"}
,{MockDbUpdate.COLUMN_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
"Column Types"}
,{MockDbUpdate.COLUMN_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
"Default data type for column"}
,{MockDbUpdate.OVERRIDE_TYPES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME, "Override Data Types"}
,{MockDbUpdate.OVERRIDE_TYPES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION, "Type to override column default"}
,{MockDbUpdate.REPEATING_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME, "Update Multiple Rows?"}
,{MockDbUpdate.REPEATING_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
"Select if input will include multiple rows to update"}
}
protected Object[][] getContents() {
// TODO Auto-generated method stub
return _contents;
}
}
```

4. Register the adapter service by updating your *fillResourceAdapterMetadataInfo* method in your *WmManagedConnectionFactory* implementation class.

In the example, *MockDbUpdate* class is registered using the *fillResourceAdapterMetadataInfo* method in the *SimpleConnectionFactory* connection factory class:

```
package com.wm.MyAdapter.connections;
import com.wm.adk.connection.WmManagedConnectionFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.info.ResourceAdapterMetadataInfo;
```



```

import com.wm.adk.metadata.WmDescriptor;
import com.wm.adk.error.AdapterException;
import java.util.Locale;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.MyAdapterConstants;
import com.wm.MyAdapter.services.MockDbUpdate;
public class SimpleConnectionFactory extends WmManagedConnectionFactory implements
    MyAdapterConstants {
    ..
    ..
    ..
    public void fillResourceAdapterMetadataInfo(ResourceAdapterMetadataInfo info,
        Locale locale) {
        info.addServiceTemplate(MockDbUpdate.class.getName());
    }
}

```

5. Execute the ANT script created in the adapter definition to compile, and deploy the adapter in Integration Server.

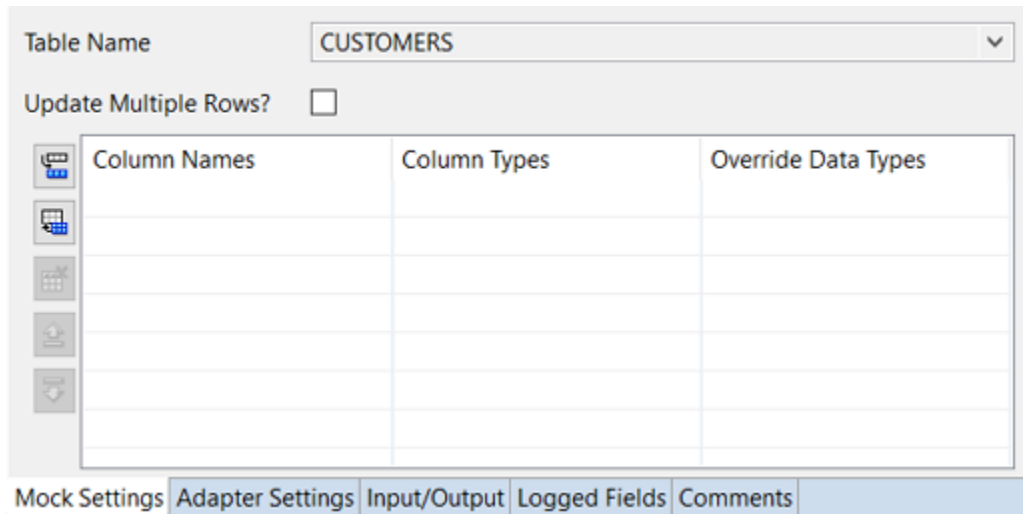
Use the files `build.xml` and `build.properties`.

```
ant deploy
```

6. Restart Integration Server.
7. Start Integration Server Administrator.
8. In Designer, create the adapter service.

A new adapter service of type *MyAdapter* is created. You can see the following in the *MockDBUpdate* tab:

- The labels are descriptive (as described in *ListResourceBundle* implementation).
- The table names are populated with mock data as described in *WmManagedConnection* implementation.
- The column names and column types update when the table name changes.



How to display adapter service signature?

1. Update the `com.wm.adk.cci.interaction.WmAdapterService` implementation class.
 - a. Create a class attribute, a set method, a constant, and a resource domain for each metadata parameter. For example, a class attribute `_fieldNames`, a corresponding set method `setFieldNames`, a corresponding constant `FIELD_NAMES_PARM`, and a corresponding resource domain `FIELD_NAMES_RD` (which needs to lookup) for the parameter table name.

Class Attribute Name	Class Attribute Set Method Name	Class Attribute Name Constant	Resource Domain Name Constant
<code>_fieldNames</code>	<code>setFieldNames</code>	<code>FIELD_NAMES_PARM</code>	<code>FIELD_NAMES_RD</code>
<code>_fieldTypes</code>	<code>setFieldTypes</code>	<code>FIELD_TYPES_PARM</code>	<code>FIELD_TYPES_RD</code>
None	<code>setSigIn</code>	<code>SIG_IN_PARM</code>	None
None	<code>setSigOut</code>	<code>SIG_OUT_PARM</code>	None

- b. In the `fillWmTemplateDescriptor` method, call `WmTemplateDescriptor.createGroup` method, `WmTemplateDescriptor.createFieldMap` method, and `WmTemplateDescriptor.createTuple` method for the new tab `Signature`.
- c. In the `fillWmTemplateDescriptor` method, call `WmTemplateDescriptor.setResourceDomain` method, and `WmTemplateDescriptor.setDescriptions` method.
- d. Create the private `Hashtable[] unpackRequest(WmRecord request)` method to process and convert the `webMethods`'s datatype `WmRecord` to `Hashtable[]`.
- e. Create the private `WmRecord packResonse(Hashtable[] response)` method to process and convert the `Hashtable[]` to `webMethods`'s datatype `WmRecord`.

- f. In the execute method, call `unpackRequest` method, and `packResonse` method.

For example, *MockDbUpdate* class:

```
package com.wm.MyAdapter.services;
import com.wm.adk.cci.interaction.WmAdapterService;
import com.wm.adk.cci.record.WmRecord;
import com.wm.adk.cci.record.WmRecordFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.WmTemplateDescriptor;
import com.wm.data.IData;
import com.wm.data.IDataCursor;
import com.wm.data.IDataFactory;
import com.wm.data.IDataUtil;
import java.util.Hashtable;
import java.util.Locale;
import javax.resource.ResourceException;
import com.wm.MyAdapter.MyAdapter;

public class MockDbUpdate extends WmAdapterService {
    //MockDB Group
    public static final String UPD_SETTINGS_GRP = "Mock Settings";
    public static final String TABLE_NAME_PARM = "tableName";
    public static final String COLUMN_NAMES_PARM = "columnNames";
    public static final String COLUMN_TYPES_PARM = "columnTypes";
    public static final String REPEATING_PARM = "repeating";
    public static final String OVERRIDE_TYPES_PARM = "overrideTypes";
    private String _tableName;
    private String[] _columnNames;
    private String[] _columnTypes;
    private boolean _repeating;
    private String[] _overrideTypes;
    public void setTableName(String val){ _tableName = val;}
    public void setColumnNames(String[] val){ _columnNames = val;}
    public void setColumnTypes(String[] val){ _columnTypes = val;}
    public void setRepeating(boolean val){ _repeating = val;}
    public void setOverrideTypes(String[] val){_overrideTypes = val;}
    public static final String TABLES_RD = "tablesRD";
    public static final String COLUMN_NAMES_RD = "columnNamesRD";
    public static final String COLUMN_TYPES_RD = "columnTypesRD";
    public static final String OVERRIDE_TYPES_RD = "overrideTypesRD";

    //MockDB Signature Group
    public static final String SIG_SETTINGS_GRP = "Signature";
    public static final String FIELD_NAMES_PARM = "fieldNames";
    public static final String FIELD_TYPES_PARM = "fieldTypes";
    public static final String SIG_IN_PARM = "sigIn";
    public static final String SIG_OUT_PARM = "sigOut";
    private String[] _fieldNames;
    private String[] _fieldTypes;
    public void setFieldNames(String[] val){ _fieldNames = val;}
    public void setFieldTypes(String[] val){ _fieldTypes = val;}
    public void setSigIn(String[] val){}
    public void setSigOut(String[] val){}
    public static final String FIELD_NAMES_RD = "fieldNamesRD";
    public static final String FIELD_TYPES_RD = "fieldTypesRD";

    public void fillWmTemplateDescriptor(WmTemplateDescriptor d,Locale l)
        throws ResourceException {
```

```

//MockDB Grouping and resource domain setup
d.createGroup(UPD_SETTINGS_GRP, new String [] {
    TABLE_NAME_PARM,
    REPEATING_PARM,
    COLUMN_NAMES_PARM,
    COLUMN_TYPES_PARM,
    OVERRIDE_TYPES_PARM}
);
d.createFieldMap(new String[] {
    COLUMN_NAMES_PARM,
    COLUMN_TYPES_PARM,
    OVERRIDE_TYPES_PARM},
true);
d.createTuple(new String[]{COLUMN_NAMES_PARM,COLUMN_TYPES_PARM});

d.setResourceDomain(TABLE_NAME_PARM, TABLES_RD, null);
d.setResourceDomain(COLUMN_NAMES_PARM, COLUMN_NAMES_RD,
    new String[]{TABLE_NAME_PARM});
d.setResourceDomain(COLUMN_TYPES_PARM, COLUMN_TYPES_RD,
    new String[]{TABLE_NAME_PARM});
d.setResourceDomain(OVERRIDE_TYPES_PARM, OVERRIDE_TYPES_RD, null);
//MockDB Signature Grouping and resource domain setup
d.createGroup(SIG_SETTINGS_GRP, new String [] {
    FIELD_NAMES_PARM,
    FIELD_TYPES_PARM,
    SIG_IN_PARM,
    SIG_OUT_PARM}
);
d.createFieldMap(new String [] {
    FIELD_NAMES_PARM,
    FIELD_TYPES_PARM,
    SIG_IN_PARM,
    SIG_OUT_PARM},
false);
d.createTuple(new String[]{FIELD_NAMES_PARM, FIELD_TYPES_PARM});

String [] fieldTupleDependencies = {TABLE_NAME_PARM,
    REPEATING_PARM,
    COLUMN_NAMES_PARM,
    COLUMN_TYPES_PARM,
    OVERRIDE_TYPES_PARM};
d.setResourceDomain(FIELD_NAMES_PARM, FIELD_NAMES_RD, fieldTupleDependencies);
d.setResourceDomain(FIELD_TYPES_PARM, FIELD_TYPES_RD, fieldTupleDependencies);
d.setResourceDomain(SIG_IN_PARM, WmTemplateDescriptor.INPUT_FIELD_NAMES,
    new String[] {FIELD_NAMES_PARM, FIELD_TYPES_PARM});
d.setResourceDomain(SIG_OUT_PARM, WmTemplateDescriptor.OUTPUT_FIELD_NAMES,
    new String[] {FIELD_NAMES_PARM, FIELD_TYPES_PARM});
//Call to setDescription
d.setDescriptions(MyAdapter.getInstance().
    getAdapterResourceBundleManager(), l);
}
public WmRecord execute(WmManagedConnection connection, WmRecord input)
throws ResourceException {
    Hashtable[] request = this.unpackRequest(input);
    return this.packResonse(request);
}
private Hashtable[] unpackRequest(WmRecord request) throws ResourceException {
    Hashtable data[] = null;
    IData mainIData = request.getIData();
    IDataCursor mainCursor = mainIData.getCursor();
}

```

```

try {
    String tableName = this._tableName;
    String[] columnNames = this._columnNames;
    if(mainCursor.first(tableName)) {
        IData[] recordIData;
        if(this._repeating) {
            recordIData = IDataUtil.getIDataArray (mainCursor,tableName);
            data = new Hashtable[recordIData.length];
        }
        else {
            recordIData = new IData[] {IDataUtil.getIData(mainCursor)};
            data = new Hashtable[1];
        }
        for(int rec=0;rec<recordIData.length;rec++) {
            IDataCursor recordCursor = recordIData[rec].getCursor();
            data[rec] = new Hashtable();
            for(int c = 0; c < columnNames.length;c++) {
                if(recordCursor.first(columnNames[c])) {
                    data[rec].put(tableName + "." + columnNames[c],
                        recordCursor.getValue());
                }
            }
            recordCursor.destroy();
        }
    }
    else {
        throw MyAdapter.getInstance().createAdapterException(9999,
            new String[] {"No Request Data"});
    }
}
catch (Throwable t) {
    throw MyAdapter.getInstance().createAdapterException(9999,
        new String[] {"Error unpacking request data"},t);
}
finally {
    mainCursor.destroy();
}
return data;
}

private WmRecord packResonse(Hashtable[] response) throws ResourceException {
    WmRecord data = null;
    try {
        IData[] recordIData = new IData[response.length];
        String tableName = this._tableName;
        String[] columnNames = this._columnNames;
        for(int rec = 0; rec < response.length; rec++) {
            recordIData[rec] = IDataFactory.create();
            IDataCursor recordCursor = recordIData[rec].getCursor();
            for(int col = 0; col < columnNames.length;col++) {
                IDataUtil.put(recordCursor,columnNames[col],
                    response[rec].get(tableName + "." +
                        columnNames[col]));
            }
            recordCursor.destroy();
        }
        IData mainIData = IDataFactory.create();
        IDataCursor mainCursor = mainIData.getCursor();
        if(this._repeating) {
            IDataUtil.put(mainCursor,tableName,recordIData);
        }
    }
}

```

```

else {
    IDataUtil.put(mainCursor, tableName, recordIDData[0]);
}
mainCursor.destroy();
data = WmRecordFactory.getFactory().createWmRecord("nameNotUsed");
data.setIDData(mainIDData);
}
catch (Throwable t) {
    throw MyAdapter.getInstance().createAdapterException(9999,
        new String[] {"Error packing response data"},t);
}
return data;
}
}

```

2. Update the WmManagedConnection implementation class.

- Add the data for the mock tables in the adapter service.
- Add the methods `adapterCheckValue`, `adapterResourceDomainLookup`, and `registerResourceDomain`

For example, *SimpleConnection* class

```

package com.wm.MyAdapter.connections;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.*;
import com.wm.adk.error.AdapterException;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.services.MockDbUpdate;
public class SimpleConnection extends WmManagedConnection {
    String hostName;
    int port;

    //Adapter Services variables
    private String[] mockTableNames = { "CUSTOMERS","ORDERS","LINE_ITEMS"};
    private String[][] mockColumnNames = {
        {"name","id", "ssn"},
        {"id","date","customer_id"},
        {"order_id","item_number","quantity","description"}
    };
    private String [][] mockDataTypes = {
        {"java.lang.String","java.lang.Integer", "java.lang.String"},
        {"java.lang.Integer", "java.util.Date", "java.lang.Integer"},
        {"java.lang.Integer", "java.lang.Integer", "java.lang.Integer",
        "java.lang.String"}
    };
    public SimpleConnection(String hostNameValue, int portValue)
    {
        super();
        hostName = hostNameValue;
        port = portValue;
        MyAdapter.getInstance().getLogger().logDebug(9999,
            "Simple Connection created with hostName = "
            + hostName + "and port = " +
            Integer.toString(port));
    }
    public void destroyConnection()
    {
        MyAdapter.getInstance().getLogger().logDebug(9999,"Simple Connection Destroyed");
    }
}

```

```

}

// The remaining methods support metadata for related services, etc.
// Implement content as needed.
public Boolean adapterCheckValue(String serviceName,
    String resourceDomainName,
    String[][] values,
    String testValue) throws AdapterException
{
    Boolean result = new Boolean(false);
    if(resourceDomainName.equals(MockDbUpdate.OVERRIDE_TYPES_RD))
    {
        try
        {
            Object o = Class.forName(testValue).getConstructor(
                new Class[] {String.class}).newInstance(new Object[]{"0"});
            result = new Boolean(true);
        }
        catch (Throwable t){}
    }
    return result;
}

public ResourceDomainValues[] adapterResourceDomainLookup(String serviceName,
    String resourceDomainName, String[][] values) throws AdapterException
{
    ResourceDomainValues[] results = null;
    //MockDB Group Lookup
    if(resourceDomainName.equals(MockDbUpdate.COLUMN_NAMES_RD) ||
        resourceDomainName.equals(MockDbUpdate.COLUMN_TYPES_RD))
    {
        String tableName = values[0][0];
        for(int x = 0; x < this.mockTableNames.length;x++)
        {
            if(this.mockTableNames[x].equals(tableName))
            {
                ResourceDomainValues columnsRdvs = new ResourceDomainValues(
                    MockDbUpdate.COLUMN_NAMES_RD,this.mockColumnNames[x]);
                columnsRdvs.setComplete(true);
                ResourceDomainValues typesRdvs = new ResourceDomainValues(
                    MockDbUpdate.COLUMN_TYPES_RD, this.mockDataTypes[x]);
                typesRdvs.setComplete(true);
                results = new ResourceDomainValues[] {columnsRdvs,typesRdvs};
                break;
            }
        }
        //MockDB Signature Group Lookup
        else if (resourceDomainName.equals(MockDbUpdate.FIELD_NAMES_RD) ||
            resourceDomainName.equals(MockDbUpdate.FIELD_TYPES_RD))
        {
            String tableName = values[0][0];
            boolean repeating = Boolean.valueOf(values[1][0]).booleanValue();
            String[] columnNames = values[2];
            String[] columnTypes = values[3];
            String[] overrideTypes = values[4];
            String[] fieldNames = new String[columnNames.length];
            String[] fieldTypes = new String[columnTypes.length];
            String optBrackets;
            if(repeating)
                optBrackets = "[]";
            else

```

```

    optBrackets = "";
    for (int i = 0; i < fieldNames.length; i++)
    {
        fieldNames[i] = tableName + optBrackets + "." + columnNames[i];
        fieldTypes[i] = columnTypes[i] + optBrackets;
        if(overrideTypes.length > i)
        {
            if (!overrideTypes[i].equals(""))
            {
                fieldTypes[i] = overrideTypes[i] + optBrackets;
            }
        }
    }
    results = new ResourceDomainValues[]{
        new ResourceDomainValues(MockDbUpdate.FIELD_NAMES_RD, fieldNames),
        new ResourceDomainValues(MockDbUpdate.FIELD_TYPES_RD, fieldTypes)};
}

return results;
}
public void registerResourceDomain(WmAdapterAccess access)
    throws AdapterException
{
    //MockDB Group Registering Resource Domain
    ResourceDomainValues tableRdvs = new ResourceDomainValues(
        MockDbUpdate.TABLES_RD, mockTableNames);
    tableRdvs.setComplete(true);
    access.addResourceDomain(tableRdvs);
    access.addResourceDomainLookup(MockDbUpdate.COLUMN_NAMES_RD, this);
    access.addResourceDomainLookup(MockDbUpdate.COLUMN_TYPES_RD, this);
    ResourceDomainValues rdvs = new ResourceDomainValues(
        MockDbUpdate.OVERRIDE_TYPES_RD, new String[] {""});
    rdvs.setComplete(false);
    rdvs.setCanValidate(true);
    access.addResourceDomain(rdvs);
    access.addCheckValue(MockDbUpdate.OVERRIDE_TYPES_RD, this);
    //MockDB Signature Group Registering Resource Domain
    access.addResourceDomainLookup(MockDbUpdate.FIELD_NAMES_RD, this);
    access.addResourceDomainLookup(MockDbUpdate.FIELD_TYPES_RD, this);
}
}
}

```

3. Update the resource bundle implementation class to add the display name and description of the adapter service.

In the example, update *MyAdapterResource* class:

```

package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.services.MockDbUpdate;
public class MyAdapterResource extends ListResourceBundle implements
MyAdapterConstants{
    ..
    ..
    static final Object[][] _contents = {
    ..
    ..
    //MockDB Group Resource Domain Values

```



```

    ,{MockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Mock Update Service"}
    ,{MockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Simulates a database update service"}
    ,{MockDbUpdate.UPD_SETTINGS_GRP + ADKGLOBAL.RESOURCEBUNDLEKEY_GROUP,
    MockDbUpdate.UPD_SETTINGS_GRP}
    ,{MockDbUpdate.TABLE_NAME_PARM +
    ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME, "Table Name"}
    ,{MockDbUpdate.TABLE_NAME_PARM +
    ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION, "Select Table Name"}
    ,{MockDbUpdate.COLUMN_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Column Names"}
    ,{MockDbUpdate.COLUMN_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Name of column updated by this service"}
    ,{MockDbUpdate.COLUMN_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Column Types"}
    ,{MockDbUpdate.COLUMN_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Default data type for column"}
    ,{MockDbUpdate.OVERRIDE_TYPES_PARM +
    ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME, "Override Data Types"}
    ,{MockDbUpdate.OVERRIDE_TYPES_PARM +
    ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION, "Type to override column default"}
    ,{MockDbUpdate.REPEATING_PARM +
    ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME, "Update Multiple Rows?"}
    ,{MockDbUpdate.REPEATING_PARM +
    ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Select if input will include multiple rows to update"}
    //MockDB Signature Group Resource Domain Values
    ,{MockDbUpdate.SIG_SETTINGS_GRP + ADKGLOBAL.RESOURCEBUNDLEKEY_GROUP,
    MockDbUpdate.SIG_SETTINGS_GRP}
    ,{MockDbUpdate.FIELD_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Field Names"}
    ,{MockDbUpdate.FIELD_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Name of Field"}
    ,{MockDbUpdate.FIELD_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Field Type"}
    ,{MockDbUpdate.FIELD_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Type of Field"}
    ,{MockDbUpdate.SIG_IN_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Input Signature"}
    ,{MockDbUpdate.SIG_IN_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Input Signature"}
    ,{MockDbUpdate.SIG_OUT_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Output Signature"}
    ,{MockDbUpdate.SIG_OUT_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Output Signature"}
    }
    protected Object[][] getContents() {
        // TODO Auto-generated method stub
        return _contents;
    }
}

```

4. Register the adapter service by updating your `fillResourceAdapterMetadataInfo` method in your `WmManagedConnectionFactory` implementation class.

In the example, *MockDbUpdate* class is registered using the `fillResourceAdapterMetadataInfo` method in the *SimpleConnectionFactory* connection factory class:

```
package com.wm.MyAdapter.connections;
import com.wm.adk.connection.WmManagedConnectionFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.info.ResourceAdapterMetadataInfo;
import com.wm.adk.metadata.WmDescriptor;
import com.wm.adk.error.AdapterException;
import java.util.Locale;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.MyAdapterConstants;
import com.wm.MyAdapter.services.MockDbUpdate;
public class SimpleConnectionFactory extends WmManagedConnectionFactory implements
MyAdapterConstants {
    ..
    ..
    ..
    public void fillResourceAdapterMetadataInfo(ResourceAdapterMetadataInfo info,
Locale locale) {
        info.addServiceTemplate(MockDbUpdate.class.getName());
    }
}
```

5. Execute the ANT script created in the adapter definition to compile, and deploy the adapter in Integration Server.

Use the files `build.xml` and `build.properties`.

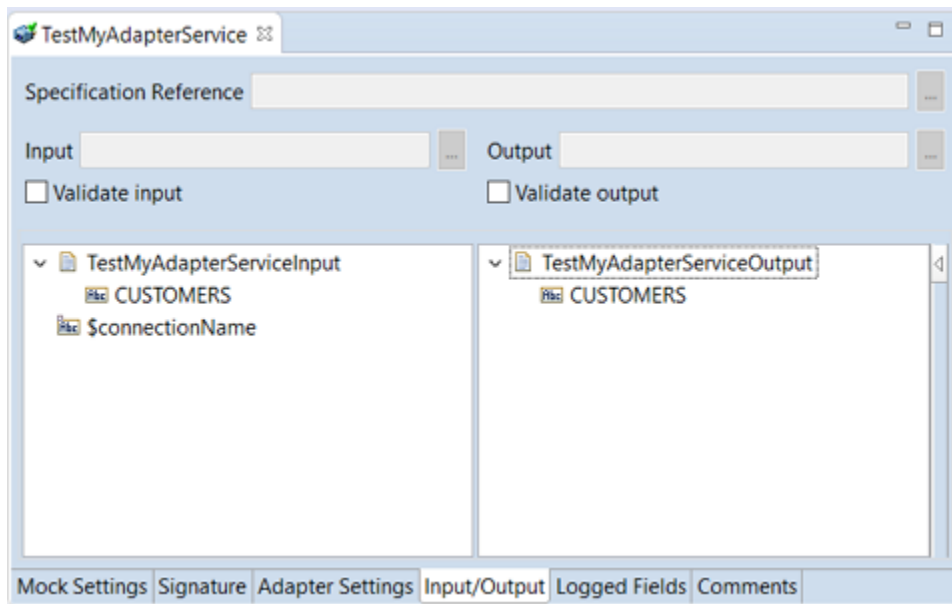
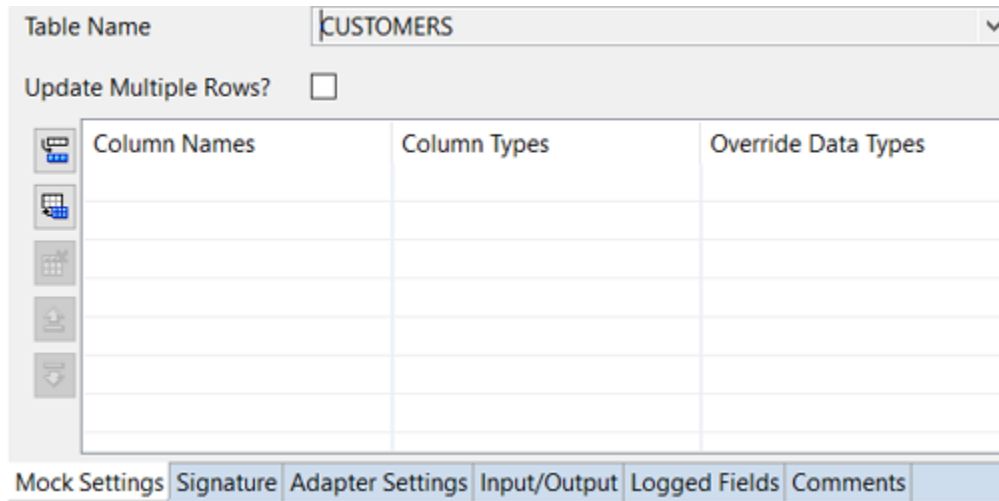
```
ant deploy
```

6. Restart Integration Server.
7. Start Integration Server Administrator.
8. In Designer, create the adapter service.

You will see the **Mock Settings**, and **Signature** tab

- a. In the **Mock Settings** tab, select a table and then select **Signature** tab. For example: In **Mock Settings** tab, select table `Customer` and then select **Signature** tab.

You will see the following in the **Mock Settings** tab, **Signature** tab, and **Input/Output** tab.



- b. In the *Mock Settings* tab, for a table, populate the rows using **Insert Row** or **Fill in all rows to the table** and then select *Signature* tab. For example: In the *Mock Settings* tab, select the table *Customer*, populate the rows using **Insert Row** or **Fill in all rows to the table** and then select *Signature* tab.

You will see the following in the **Mock Settings** tab, **Signature** tab, and **Input/Output** tab.

Table Name: CUSTOMERS

Update Multiple Rows?

Column Names	Column Types	Override Data Types
name	java.lang.String	
id	java.lang.Integer	
ssn	java.lang.String	

Mock Settings | Signature | Adapter Settings | Input/Output | Logged Fields | Comments

Field Names	Field Type	Input Signature	Output Signature
CUSTOMERS.name	java.lang.Type of Field	CUSTOMERS.name	CUSTOMERS.name
CUSTOMERS.id	java.lang.Integer	CUSTOMERS.id	CUSTOMERS.id
CUSTOMERS.ssn	java.lang.String	CUSTOMERS.ssn	CUSTOMERS.ssn

Mock Settings | Signature | Adapter Settings | Input/Output | Logged Fields | Comments

TestMyAdapterService

Specification Reference: _____

Input: _____ Output: _____

Validate input Validate output

TestMyAdapterServiceInput

- CUSTOMERS
 - name
 - id
 - ssn
- \$connectionName

TestMyAdapterServiceOutput

- CUSTOMERS
 - name
 - id
 - ssn

Mock Settings | Signature | Adapter Settings | Input/Output | Logged Fields | Comments

Note:

The **Signature** tab shown is for demonstration purposes only. In most cases, the fields in the **Signature** are hidden. Select the **Input/Output** tab to view the signature.

An adapter service of type *MyAdapter* is created with two user defined tabs *Mock Settings*, and *Signature*.

How to create a polling notification implementation?

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.

Ensure that you have:

- webMethods Integration Server 9.12 or later installed.
 - Designer 9.12 or later installed.
 - Integration Server Administrator access.
 - Java 1.8 or later installed.
 - Basic understanding of webMethods Integration Server, Designer, Integration Server Administrator, Java.
1. Start the editor to create Java files for adapter service implementation.
 2. Create directories corresponding to your Java package structure in the webMethods package you created using Designer. For example: `com\mycompany\adapter\myAdapter\notifications`. In the example, the folder created is `com\wm\MyAdapter\notifications`.

Note:

You must create your Java package and classes in the `adapterPackageName\code\source` folder in the webMethods package you created using Designer.

3. Create the `com.wm.adk.notification.WmPollingNotification` implementation class.

In the example, created a *SimpleNotification* class.

- a. Create a constant for grouping the metadata parameters.

In the example, `NOTIFICATION_SETUP_GROUP` is the constant.

- b. Create a class attribute, a set method, a constant, and a resource domain for each metadata parameter. For example, a constant `DIRECTORY_PARM`, a corresponding class attribute `_directory`, a corresponding set method `setDirectory`, and a resource domain `DIRECTORIES_RD` (which needs to lookup) for the parameter table name.

Class Attribute Name	Class Attribute Set Method Name	Class Attribute Name Constant	Resource Domain Name Constant
<i>_directory</i>	setDirectory	<i>DIRECTORY_PARM</i>	<i>DIRECTORIES_RD</i>
<i>_checkAdded</i>	setCheckAdded	<i>CHECK_ADDED_PARM</i>	None
<i>_checkDeleted</i>	setCheckDeleted	<i>CHECK_DELETED_PARM</i>	None
<i>_fieldNames</i>	setFieldNames	<i>SIG_FIELD_NAMES_PARM</i>	<i>FIELD_NAMES_RD</i>
<i>_fieldTypes</i>	setFieldTypes	<i>SIG_FIELD_TYPES_PARM</i>	<i>FIELD_TYPES_RD</i>
None	setSignature	<i>SIG_PARM</i>	None

- c. In the `fillWmTemplateDescriptor` method, call `WmTemplateDescriptor.createGroup` method, `WmTemplateDescriptor.createFieldMap` method, and `WmTemplateDescriptor.createTuple` method.
- d. In the `fillWmTemplateDescriptor` method, call `WmTemplateDescriptor.setHidden` method to hide the fields.
- e. In the `fillWmTemplateDescriptor` method, call `WmTemplateDescriptor.setResourceDomain` method, and `WmTemplateDescriptor.setDescriptions` method.
- f. In the `runNotification` method, call `doNotify` method.
- g. In the `adapterCheckValue` method, check if a lookup is performed on the `_directory` value and if the specified folder exists.
- h. In the `adapterResourceDomainLookup` method, add resource domain values if a lookup is performed on the `_fieldNames` and `_fieldTypes`.
- i. In the `registerResourceDomain` method, register resource domain lookup for the `_fieldNames` and `_fieldTypes` metadata parameters, and add resource domain values for the `_directory` metadata parameter.
- j. Create a `createNotice` method to add the information logged.

For example, the `SimpleNotification` class:

```
package com.wm.MyAdapter.notifications;
import com.wm.adk.cci.record.WmRecord;
import com.wm.adk.cci.record.WmRecordFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.error.AdapterException;
import com.wm.adk.metadata.ResourceDomainValues;
import com.wm.adk.metadata.WmAdapterAccess;
import com.wm.adk.metadata.WmTemplateDescriptor;
import com.wm.adk.notification.WmPollingNotification;
import java.io.File;
```

```
import java.util.ArrayList;
import java.util.Locale;
import javax.resource.ResourceException;
import com.wm.MyAdapter.MyAdapter;
```

```
public class SimpleNotification extends WmPollingNotification
{
    public static final String NOTIFICATION_SETUP_GROUP = "SimpleNotification";
    public static final String DIRECTORY_PARM = "directory";
    public static final String CHECK_ADDED_PARM = "checkAdded";
    public static final String CHECK_DELETED_PARM = "checkDeleted";
    public static final String SIG_FIELD_NAMES_PARM = "fieldNames";
    public static final String SIG_FIELD_TYPES_PARM = "fieldTypes";
    public static final String SIG_PARM = "signature";
    public static final String DIRECTORIES_RD =
        "SimpleNotification.directories.rd";
    public static final String FIELD_NAMES_RD =
        "SimpleNotification.fieldNames.rd";
    public static final String FIELD_TYPES_RD =
        "SimpleNotification.fieldTypes.rd";
    private String _directory;
    private boolean _checkAdded;
    private boolean _checkDeleted;
    private String[] _fieldNames;
    private String[] _fieldTypes;
    public void setDirectory(String val){_directory = val;}
    public void setCheckAdded(boolean val){_checkAdded = val;}
    public void setCheckDeleted(boolean val){_checkDeleted = val;}
    public void setFieldNames(String[] val){_fieldNames = val;}
    public void setFieldTypes(String[] val){_fieldTypes = val;}
    public void setSignature(String[] val){}
    private ArrayList _fileList = new ArrayList();
    public SimpleNotification(){}
```

```
public void fillWmTemplateDescriptor(WmTemplateDescriptor descriptor, Locale l)
    throws ResourceException
{
    descriptor.createGroup(NOTIFICATION_SETUP_GROUP,
        new String[]{DIRECTORY_PARM, CHECK_ADDED_PARM, CHECK_DELETED_PARM,
            SIG_FIELD_NAMES_PARM, SIG_FIELD_TYPES_PARM, SIG_PARM});
    descriptor.createFieldMap(
        new String[]{SIG_FIELD_NAMES_PARM, SIG_FIELD_TYPES_PARM, SIG_PARM},
        false);
    descriptor.setHidden(SIG_FIELD_NAMES_PARM);
    descriptor.setHidden(SIG_FIELD_TYPES_PARM);
    descriptor.setHidden(SIG_PARM);
    descriptor.createTuple(
        new String[]{SIG_FIELD_NAMES_PARM, SIG_FIELD_TYPES_PARM});
    descriptor.setResourceDomain(DIRECTORY_PARM, DIRECTORIES_RD, null);
    descriptor.setResourceDomain(SIG_FIELD_NAMES_PARM, FIELD_NAMES_RD, null);
    descriptor.setResourceDomain(SIG_FIELD_TYPES_PARM, FIELD_TYPES_RD, null);
    descriptor.setResourceDomain(SIG_PARM, WmTemplateDescriptor.OUTPUT_FIELD_NAMES,
        new String[]{SIG_FIELD_NAMES_PARM, SIG_FIELD_TYPES_PARM});
    descriptor.setDescriptions(
        MyDemoAdapter.getInstance().getAdapterResourceBundleManager(),l);
}
```

```
public void runNotification() throws ResourceException
{
    File thisDir = new File(_directory);
```

```

File [] newList = thisDir.listFiles();
ArrayList scratchCopy = new ArrayList(this._fileList);
for (int nlIndex = 0;nlIndex < newList.length;nlIndex++)
{
    String name = newList[nlIndex].getName();
    if(newList[nlIndex].isFile())
    {
        if(scratchCopy.contains(name))
        {
            scratchCopy.remove(name);
        }
        else
        {
            this._fileList.add(name);
            if(this._checkAdded)
            {
                this.doNotify(createNotice(name, _directory, true, false));
            }
        }
    }
    else
    {
        scratchCopy.remove(name);
    }
}
// now anything left in the scratch copy is missing from the directory
String[] deadList = new String[scratchCopy.size()];
scratchCopy.toArray(deadList);
for(int dlIndex = 0; dlIndex < deadList.length;dlIndex++)
{
    this._fileList.remove(deadList[dlIndex]);
    if(this._checkDeleted)
    {
        this.doNotify(createNotice(deadList[dlIndex], _directory, false, true));
    }
}
}
}

```

```

public Boolean adapterCheckValue(WmManagedConnection connection,
    String resourceDomainName, String[][] values, String testValue)
    throws AdapterException
{
    boolean result = true;
    if(resourceDomainName.equals(DIRECTORIES_RD))
    {
        File testDir = new File(testValue);
        if (!testDir.exists())
        {
            result = false;
        }
        else if(!testDir.isDirectory())
        {
            result = false;
        }
    }
    return new Boolean(result);
}

```

```

public ResourceDomainValues[] adapterResourceDomainLookup(
    WmManagedConnection connection, String resourceDomainName,

```



```
String[][] values) throws AdapterException
{
    ResourceDomainValues[] results = null;
    if (resourceDomainName.equals(FIELD_NAMES_RD) ||
        resourceDomainName.equals(FIELD_TYPES_RD))
    {
        ResourceDomainValues names =
            new ResourceDomainValues(FIELD_NAMES_RD,new String[] {
                "FileName", "Path","isAdded","isDeleted"});
        ResourceDomainValues types =
            new ResourceDomainValues(FIELD_TYPES_RD,new String[] {
                "java.lang.String", "java.lang.String",
                "java.lang.Boolean","java.lang.Boolean"});
        results = new ResourceDomainValues[] {names,types};
    }
    return results;
}
}
```

```
public void registerResourceDomain( WmManagedConnection connection,
    WmAdapterAccess access) throws AdapterException
{
    access.addResourceDomainLookup(this.getClass().getName(),
        FIELD_NAMES_RD,connection);
    access.addResourceDomainLookup(this.getClass().getName(),
        FIELD_TYPES_RD,connection);
    ResourceDomainValues rd = new ResourceDomainValues(DIRECTORIES_RD,
        new String[] {""});
    rd.setComplete(false);
    rd.setCanValidate(true);
    access.addResourceDomain(rd);
    access.addCheckValue(DIRECTORIES_RD, connection);
}
}
```

```
private WmRecord createNotice(String file, String dir, boolean isAdded,
    boolean isDeleted)
{
    WmRecord notice =
        WmRecordFactory.getFactory().createWmRecord("notUsed");
    notice.put("FileName",file);
    notice.put("Path",dir);
    notice.put("isAdded",new Boolean(isAdded));
    notice.put("isDeleted", new Boolean(isDeleted));
    return notice;
}
}
```

4. Update the `WmManagedConnection` implementation class to add the polling notification metadata parameters with resource domain lookup.

In the example, update `SimpleConnection` class's `registerResourceDomain` method as follows:

```
package com.wm.MyAdapter.connection;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.*;
import com.wm.adk.error.AdapterException;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.services.MockDbUpdate;
import com.wm.MyAdapter.notifications.SimpleNotification;
public class SimpleConnection extends WmManagedConnection {
```

```

..
..
public void registerResourceDomain(WmAdapterAccess access)
    throws AdapterException
{
    ..
    ..
    //Simple Notification Registering Resource Domain
    access.addResourceDomainLookup(SimpleNotification.DIRECTORIES_RD, this);
    access.addResourceDomainLookup(SimpleNotification.FIELD_NAMES_RD, this);
    access.addResourceDomainLookup(SimpleNotification.FIELD_TYPES_RD, this);
}

```

5. Update the resource bundle implementation class to add the display name, and description of the polling notification class and the fields in the polling notification class.

In the example, update *MyAdapterResource* class's `Object[][] _contents` as follows:

```

package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.notifications.SimpleNotification;
public class MyAdapterResource extends ListResourceBundle implements
MyAdapterConstants{
    ..
    ..
    static final Object[][] _contents = {
        ..
        ..
        //Polling Notifications
        ,{SimpleNotification.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Simple Polling Notification"}
        ,{SimpleNotification.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Looks for file updates to a specified directory"}
        ,{SimpleNotification.NOTIFICATION_SETUP_GROUP + ADKGLOBAL.RESOURCEBUNDLEKEY_GROUP,
        "Simple Notification Settings"}
        ,{SimpleNotification.DIRECTORY_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Directory Path"}
        ,{SimpleNotification.DIRECTORY_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Directory to monitor"}
        ,{SimpleNotification.CHECK_ADDED_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Notify on Add"}
        ,{SimpleNotification.CHECK_ADDED_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Check if notification should be generated when file added"}
        ,{SimpleNotification.CHECK_DELETED_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Notify on Delete"}
        ,{SimpleNotification.CHECK_DELETED_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Check if notification should be generated when file deleted"}
    }
    protected Object[][] getContents() {
        // TODO Auto-generated method stub
        return _contents;
    }
}

```

6. Register the polling notification type in the adapter by updating your `fillAdapterTypeInfo` method in your `WmAdapter` implementation class.

In the example, the polling notification class *SimpleNotification* is registered using the `addNotificationType` method in the adapter implementation class's *MyAdapter.fillAdapterTypeInfo* method:

```
package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.notifications.*;
..
..
public class MyAdapter extends WmAdapter implements MyAdapterConstants {
..
..
public void fillAdapterTypeInfo(AdapterTypeInfo info, Locale locale)
{
    ..
    ..
    info.addNotificationType(SimpleNotification.class.getName());
}
}
```

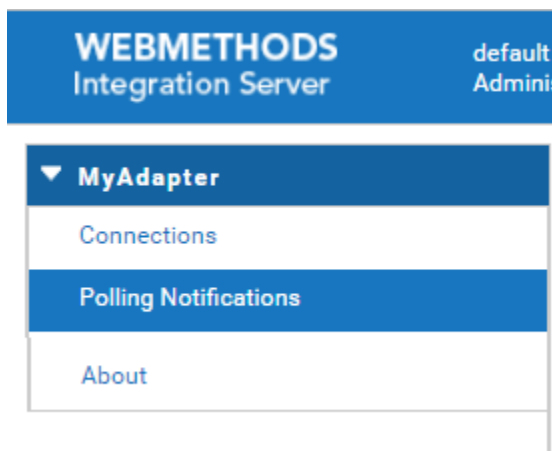
7. Execute the ANT script created in adapter definition to compile, and deploy the adapter in Integration Server.

Use the files `build.xml` and `build.properties`.

```
ant deploy
```

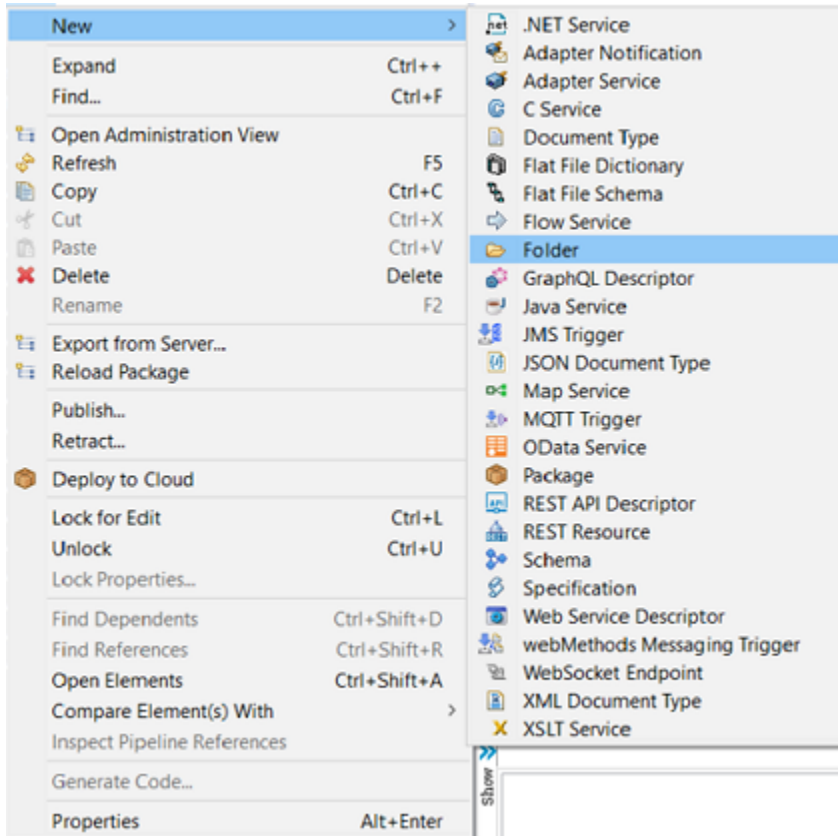
8. Restart Integration Server.
9. In Integration Server Administrator, navigate to **Adapters > MyAdapter**.

Polling Notifications appears in the menu.

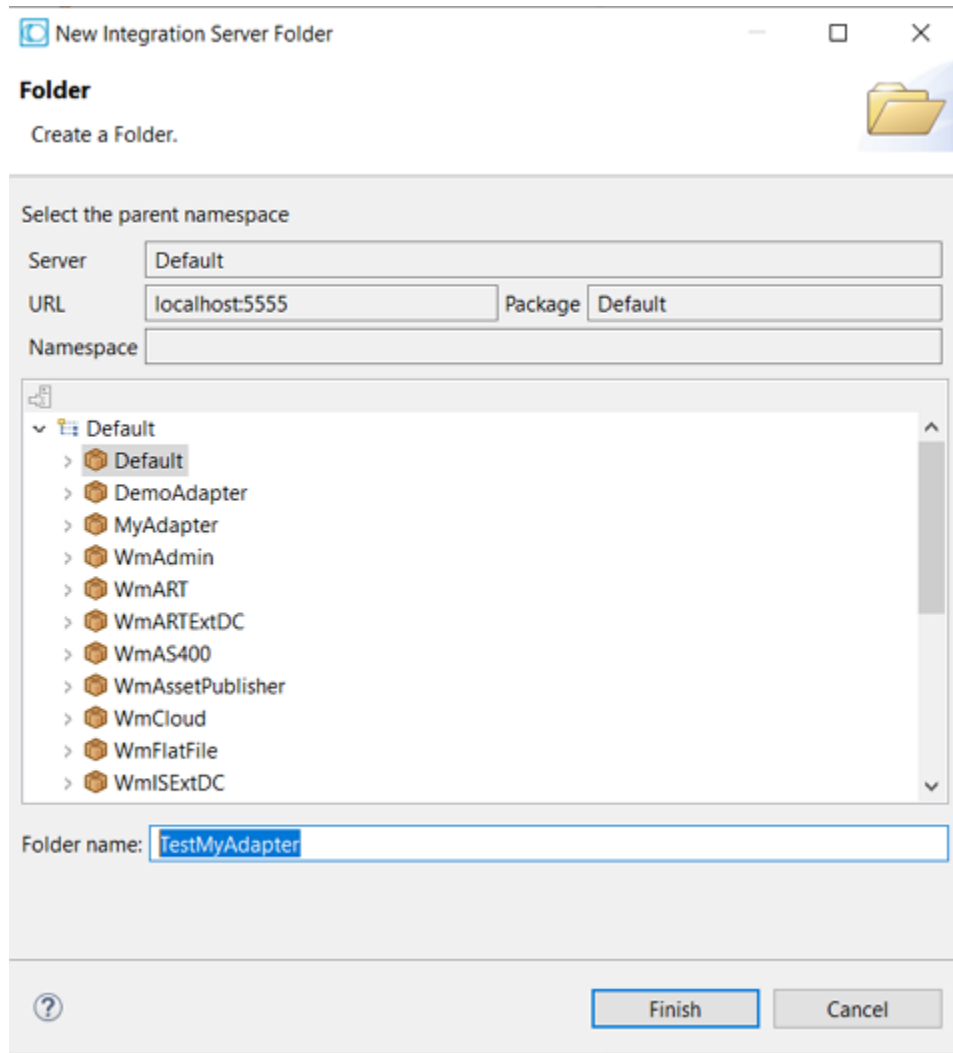


10. In Designer, create the **Adapter Notification**.
 - a. In **Package Navigator**, select the **Default** package.

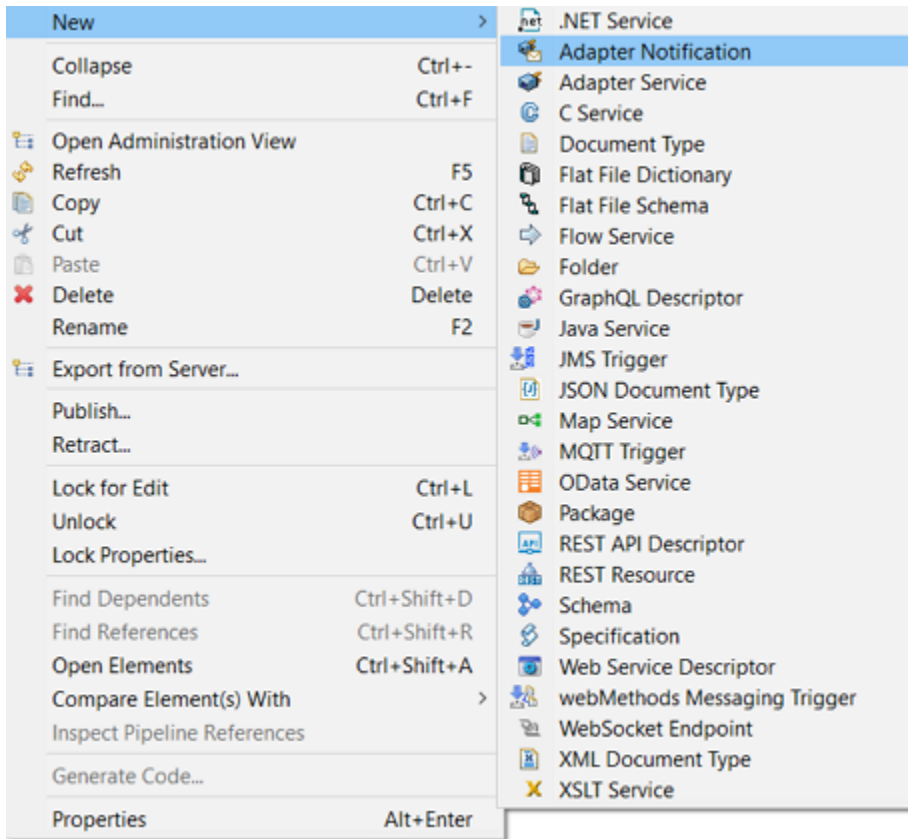
- b. Select **File > New > Folder**.



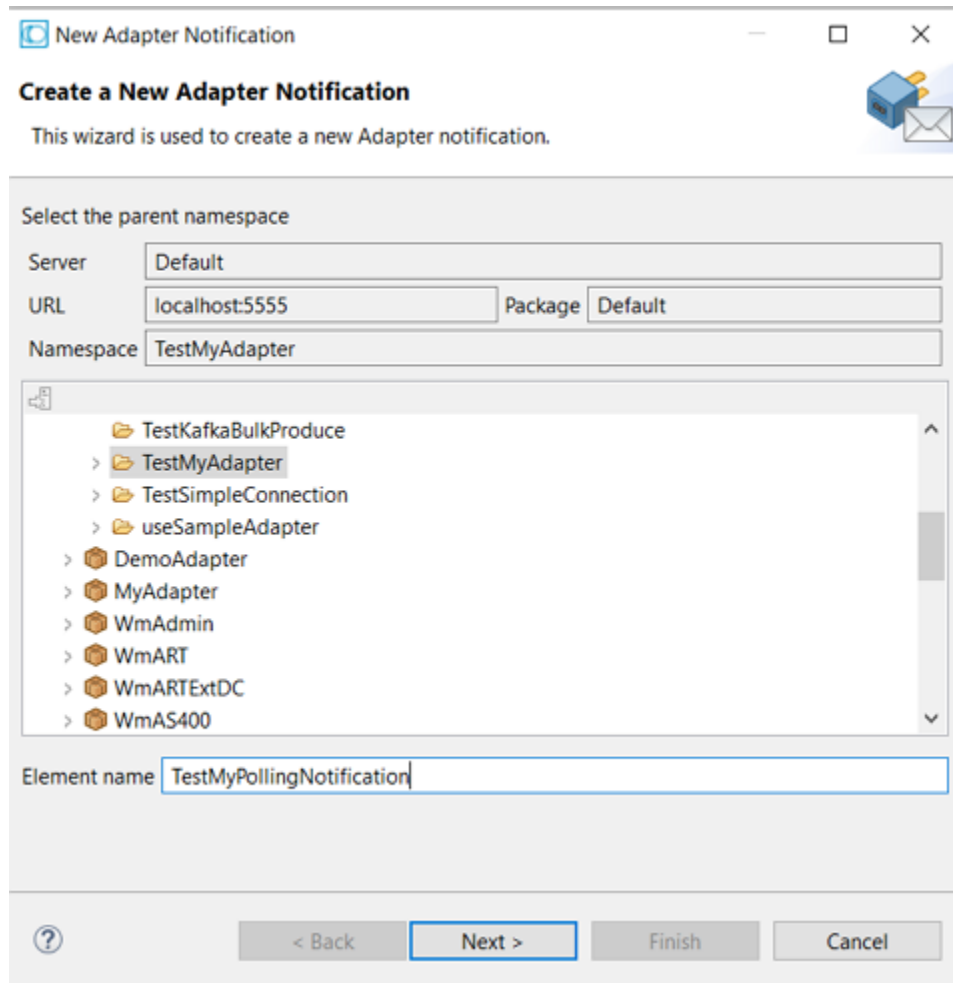
- c. Enter the **Folder name**. For example: *TestMyAdapter*.



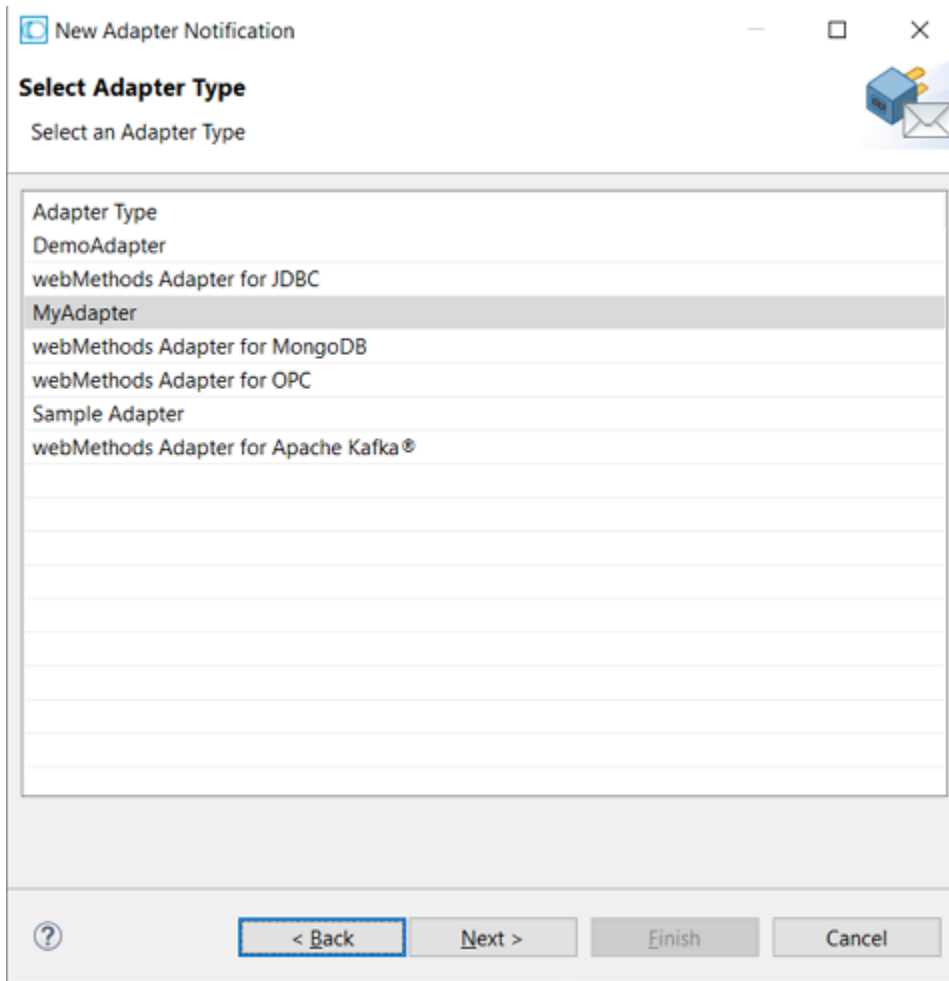
- d. In **Package Navigator**, select the **Default > TestMyAdapter**.
- e. Select **File > New > Adapter Notification**.



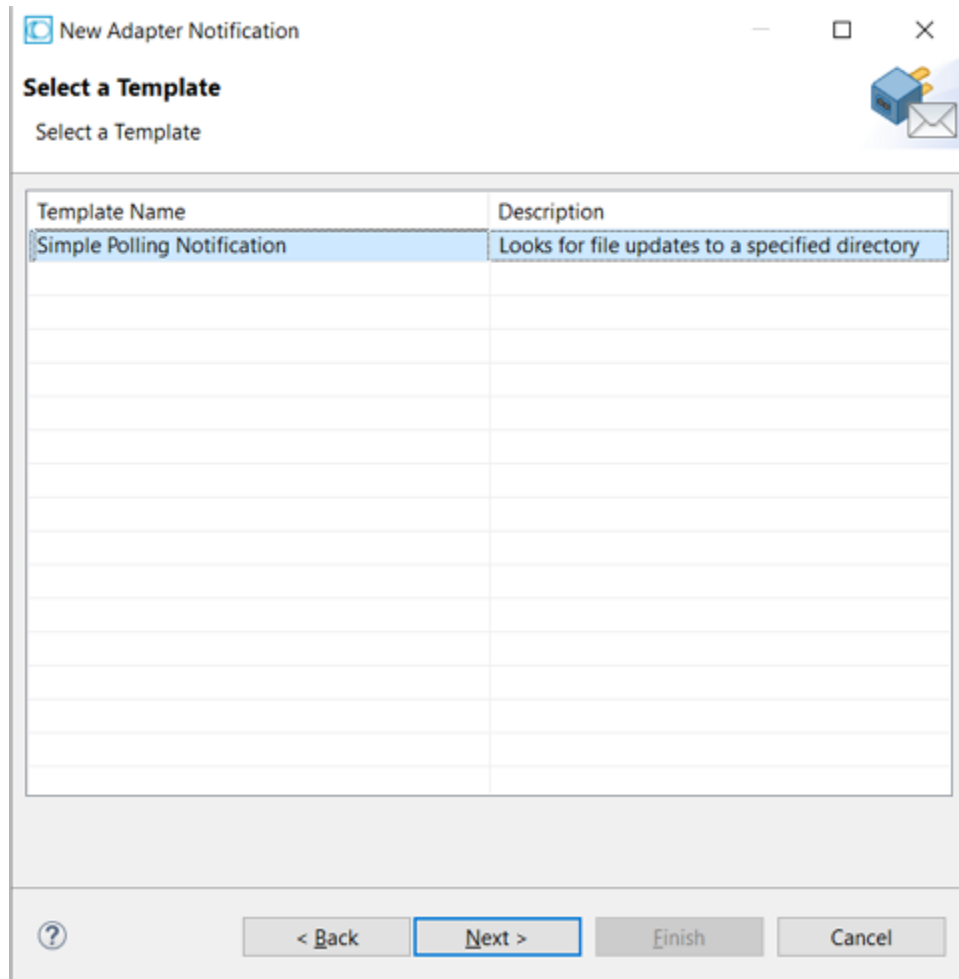
- f. In the **Create a New Adapter Notification** screen, enter the **Element name** and click **Next**. For example: *TestMyPollingNotification*.



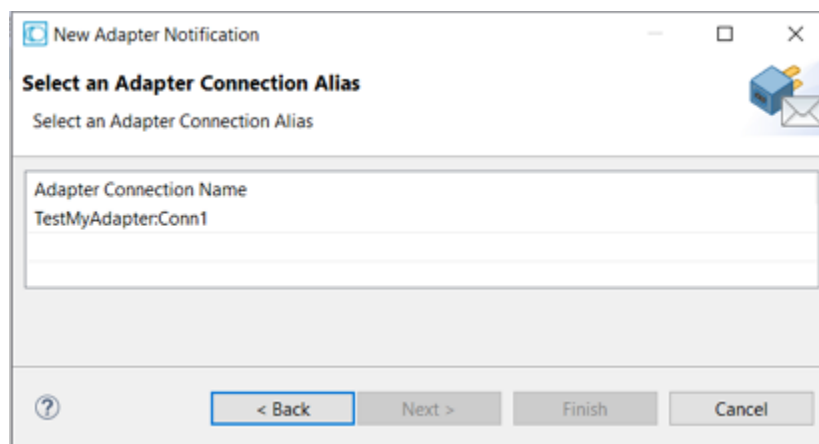
- g. In the **Select Adapter Type** screen, select an adapter type for which you want to create the polling notification. For example: *MyAdapter*.



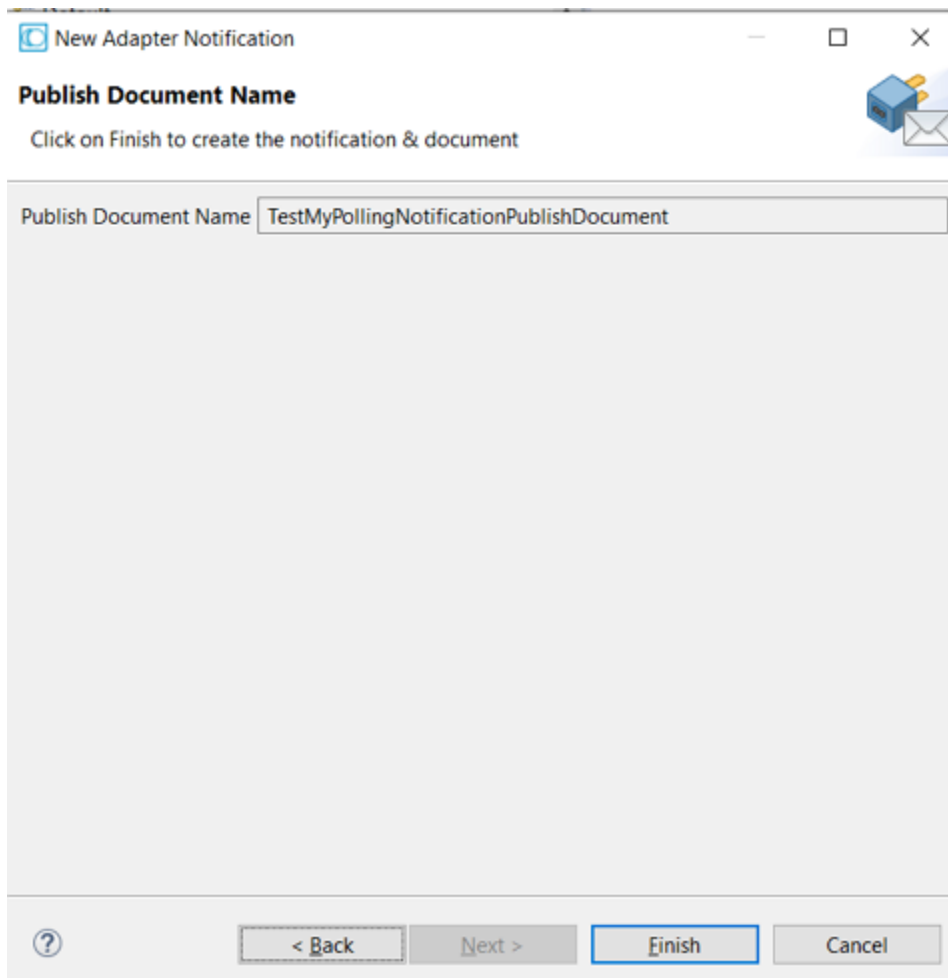
- h. In the **Select a Template** screen, select a polling notification template, and click **Finish**. For example: **Simple Polling Notification**.



- i. In the **Select an Adapter Connection Alias** screen, select an adapter connection. For example: **TestMyAdapter:Conn1**.

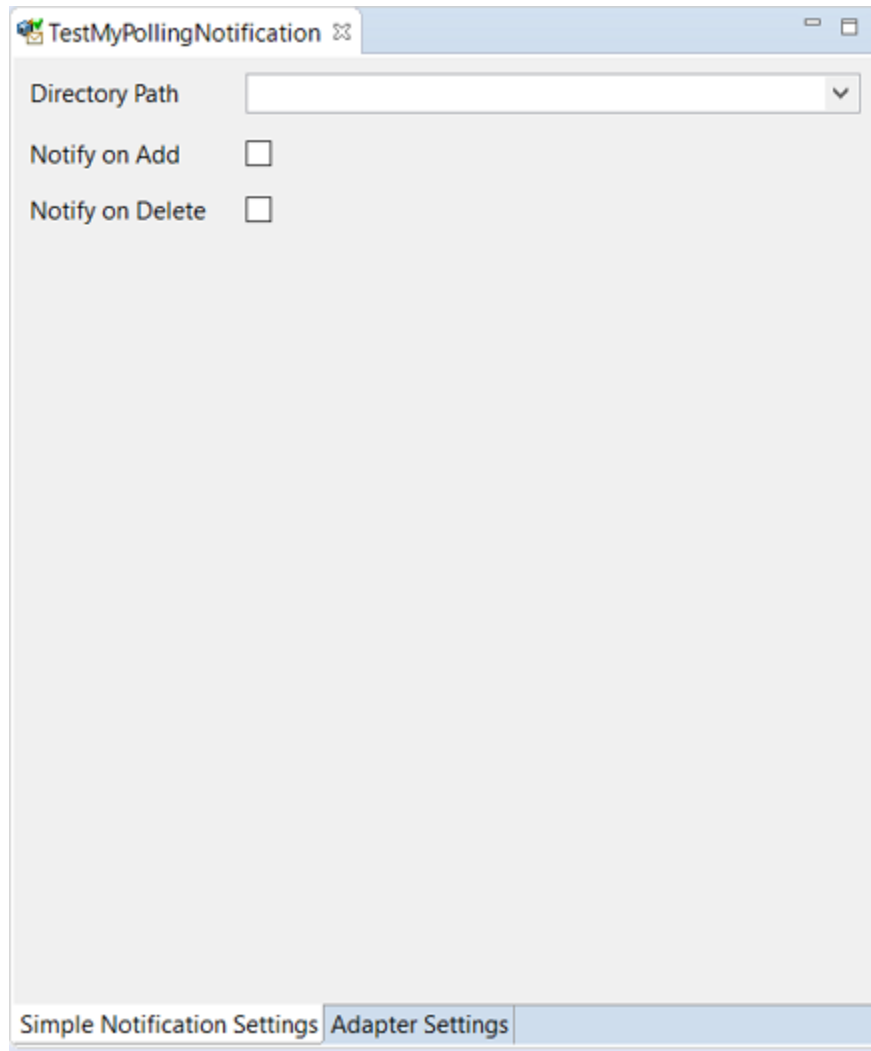


- j. In the **Publish Document Name** screen, select **Finish**.



In Designer, you can see the following two items created:

- a. A new adapter notification *TestMyPollingNotification* is created with two tabs: *Simple Notification Settings*, and *Adapter Settings*.



Adapter Properties

Adapter Name: MyAdapter

Adapter Listener Name: TestMyAdapter:Conn1

Adapter Notification Template: Simple Polling Notification

Publish Document to

webMethods Messaging Provider

Connection alias name: DEFAULT(IS_LOCAL_CONNECTION) ▾

JMS Provider

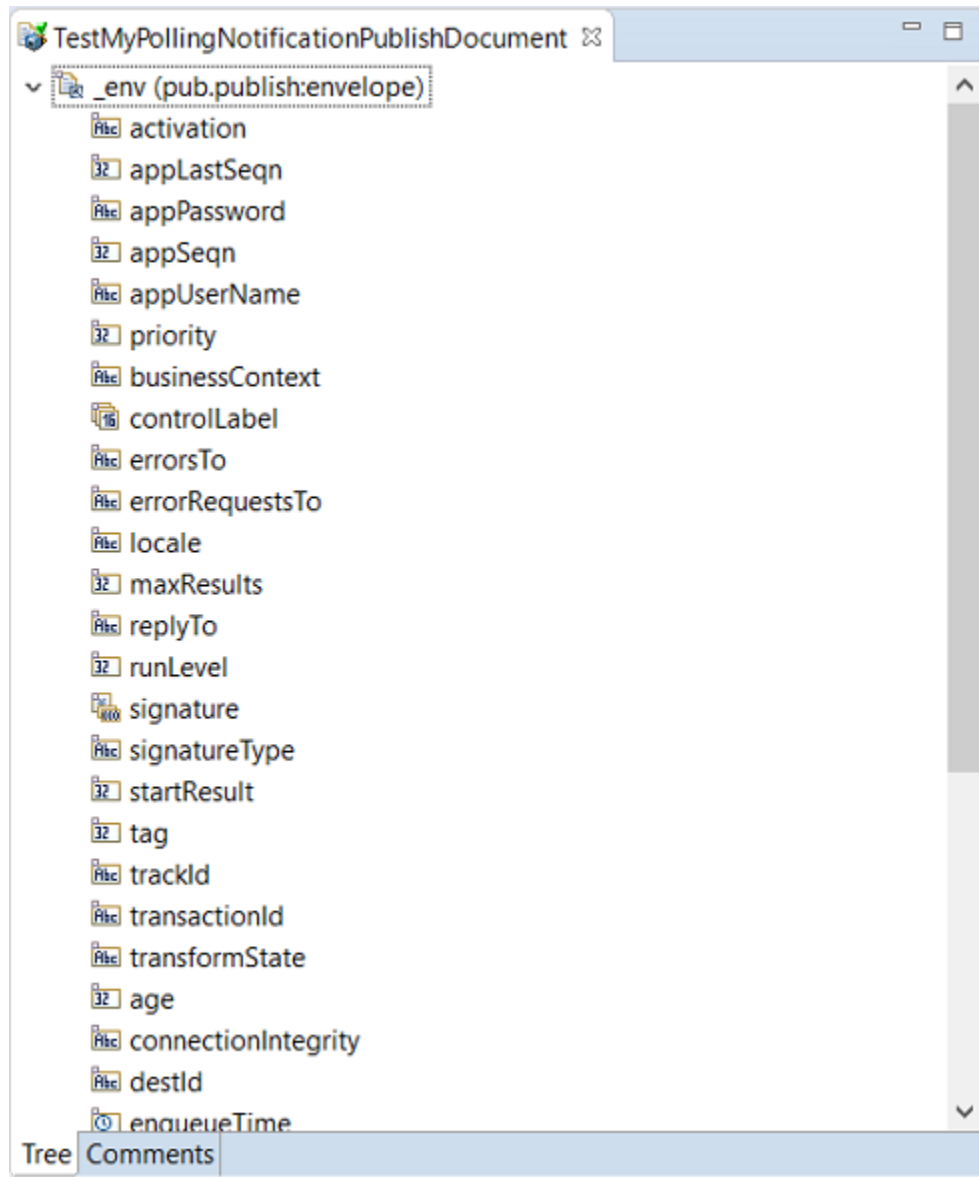
Connection alias name:

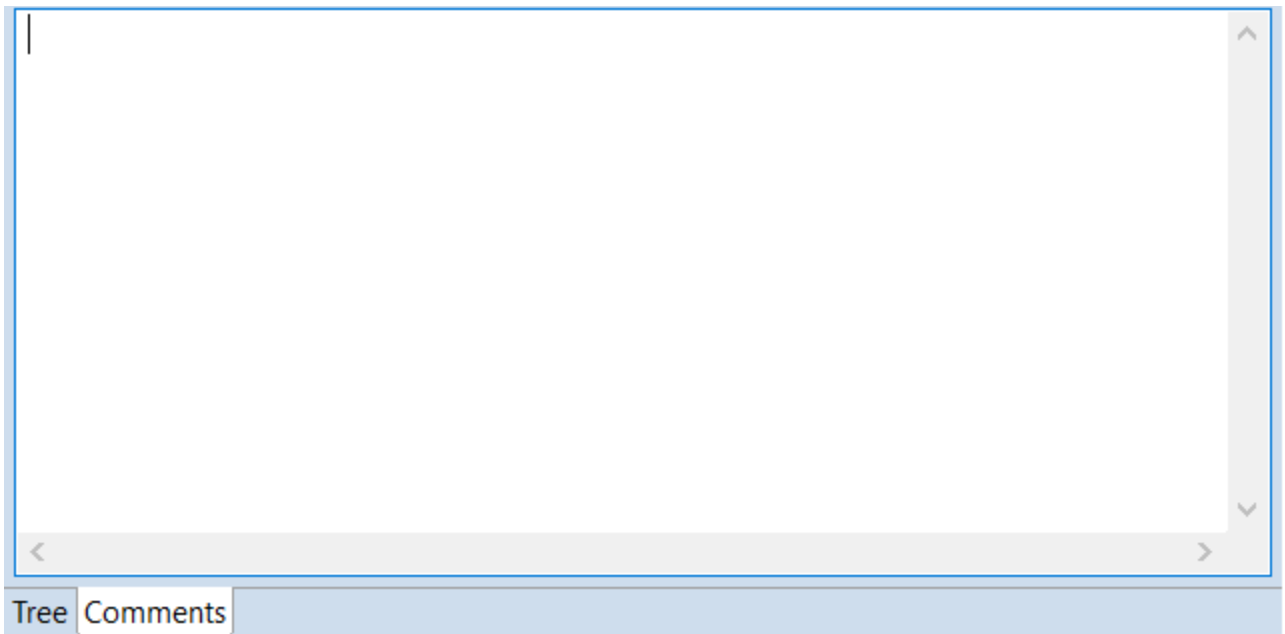
Destination name :

Destination type: Queue ▾

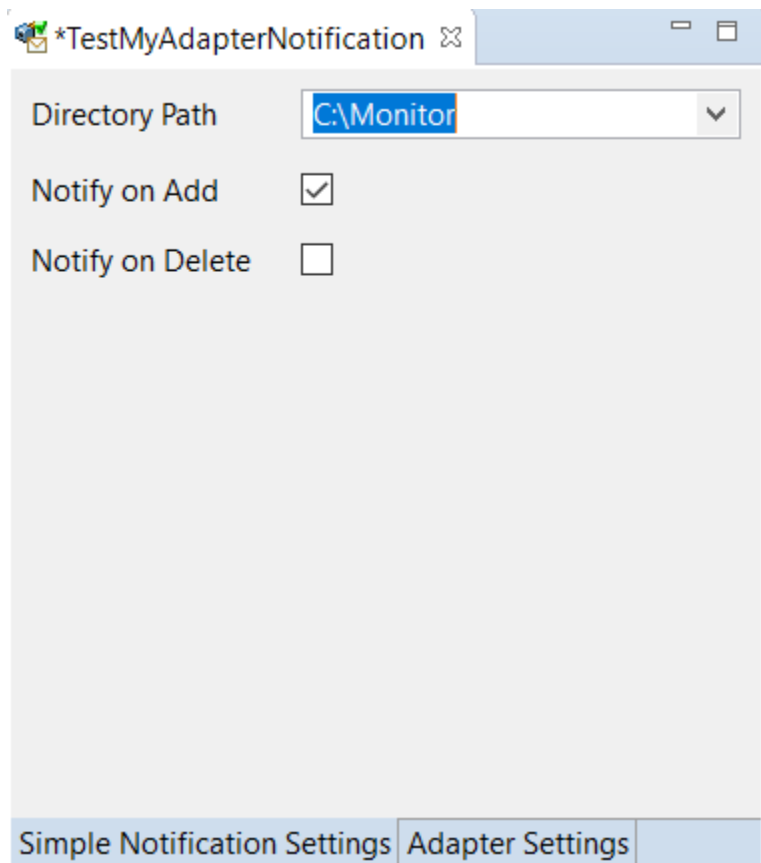
Simple Notification Settings Adapter Settings

- b. A new Document Type *TestMyPollingNotificationPublishDocument* is created with two tabs: **Tree**, and **Comments**.





11. In Designer, select a **Directory Path** to monitor and the polling event **Notify on Add** and **Notify on Delete** to monitor.

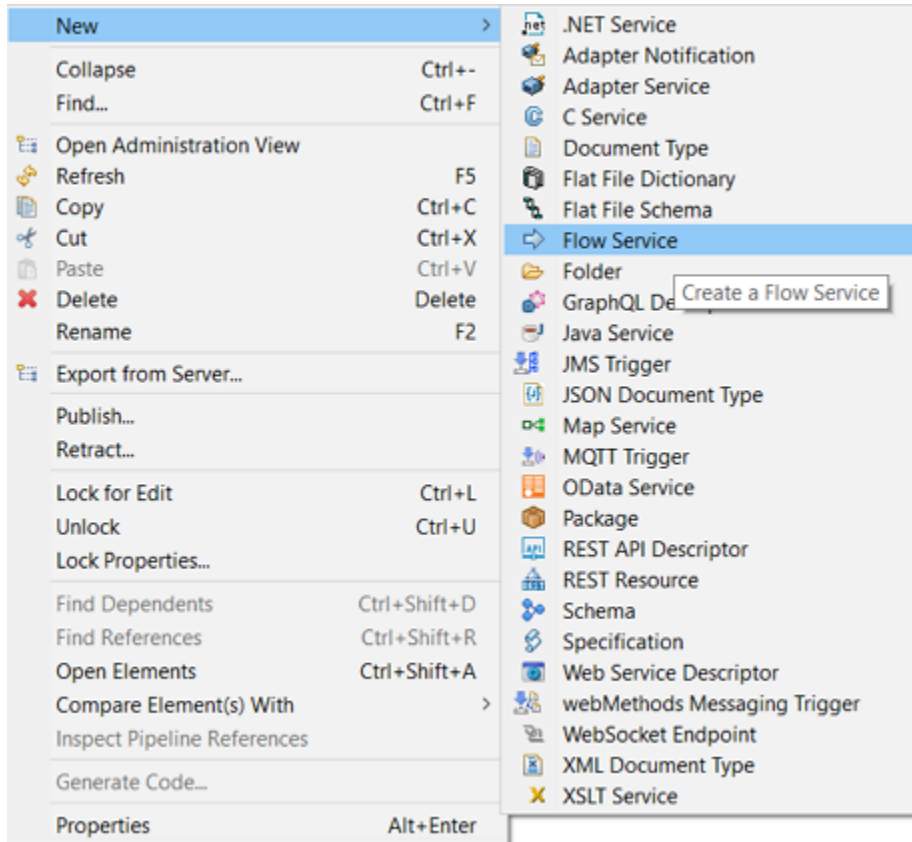


12. Select **File > Save**.

13. Create a Flow Service for the Polling Notification Node using Designer.

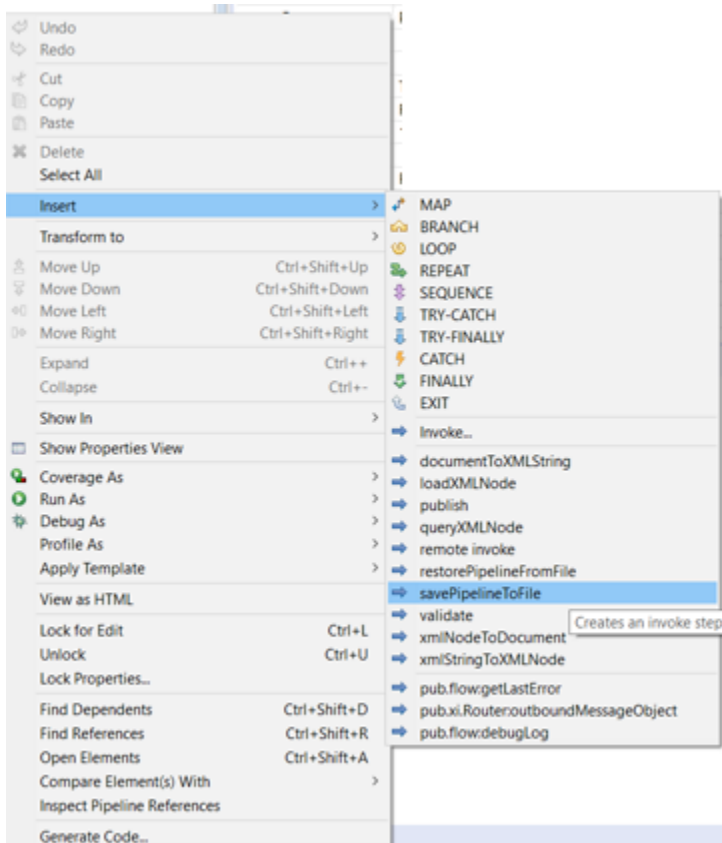
a. Navigate to the folder **Default > TestMyAdapter**.

b. Select **New > Flow Service**.

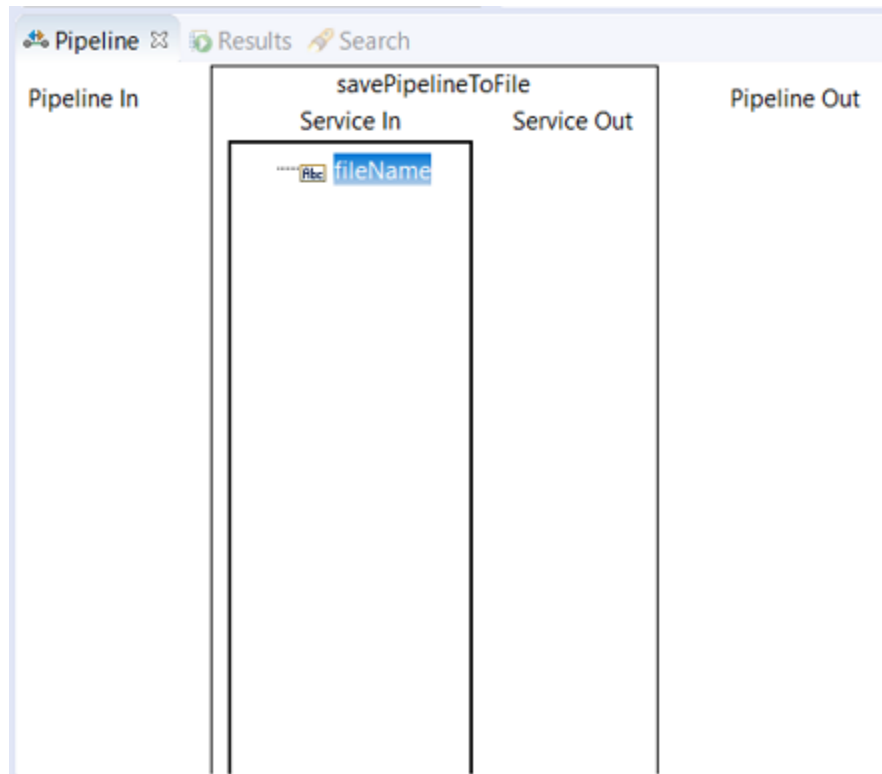


c. In the **Create a New Flow Service** screen, add *TestMyAdapterFlowService* in the **Element name** field and click **Finish**.

d. In the **Flow Service > Tree** tab, right-click and select **Insert > savePipelineToFile**.



- e. In the **Flow Service, Tree** tab, select the **savePipelineToFile** method. You can see the **Service In > fileName** in the **Pipeline** tab.



- f. Update **Service In > fileName** in the **Pipeline** tab. In this example, the value is *MonitorPollingNotificationPipeline.log*.

The screenshot shows a dialog box titled 'Enter input for 'fileName''. The dialog has a close button (X) in the top right corner. Below the title, there is a section for 'Enter Input for 'fileName'' with a toolbar containing icons for grid, list, and other views, and a checkbox for 'Include empty values for String Types'. Below this is a table with two columns: 'Name' and 'Value'. The table has one row with the following data:

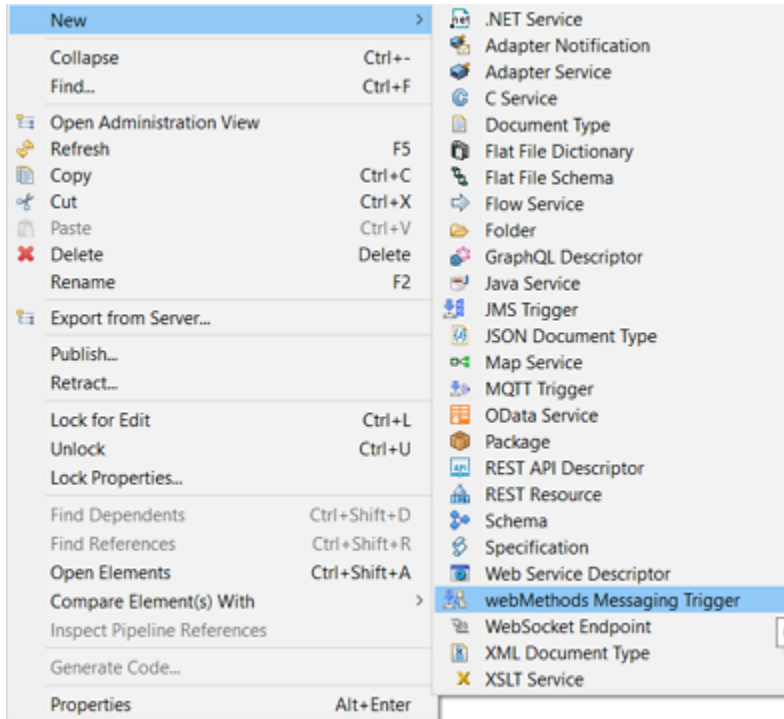
Name	Value
<input type="checkbox"/> fileName	MonitorPollingNotificationPipeline.log

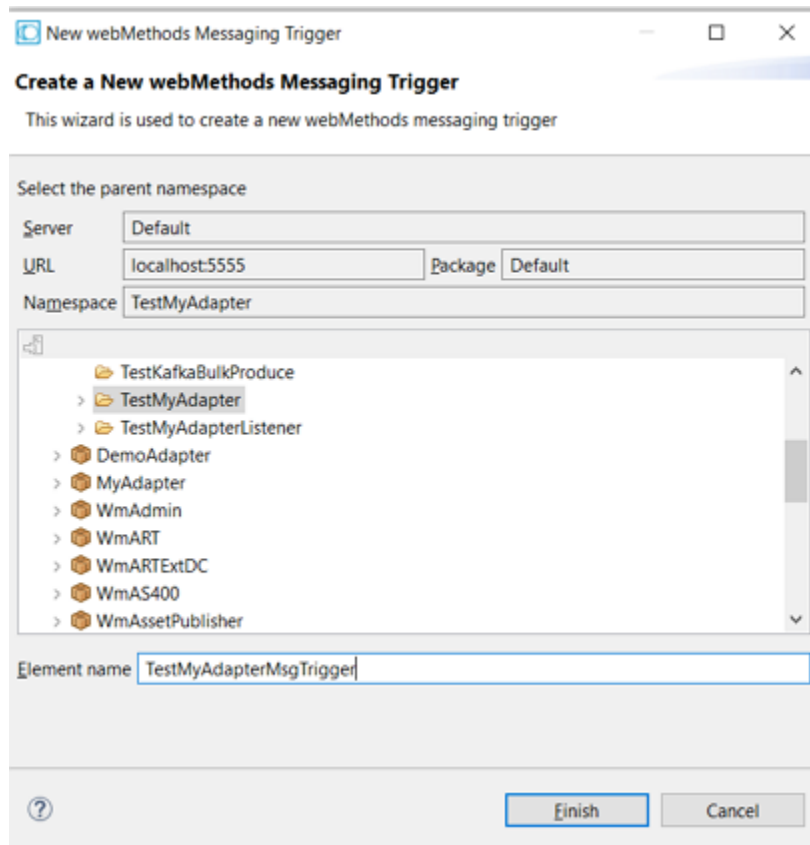
Below the table, there are three checkboxes: 'Overwrite pipeline value' (checked), 'Perform pipeline variable substitution' (unchecked), and 'Perform global variable substitution' (unchecked). At the bottom of the dialog, there are 'OK' and 'Cancel' buttons, and a help icon (?) on the left.


- g. Click **OK**, and save the flow service.

14. Create a trigger using Designer.

- a. Navigate to the folder **Default > TestMyAdapter**.
- b. Create the **webMethods Messaging Trigger** and select **Finish**. In the example, *TestMyAdapterMsgTrigger* is created.





- c. In the **webMethods Messaging Trigger**, **Trigger Settings** tab, **Condition Detail** section, perform the following:
- In the trigger editor, in the Conditions section, accept the default **Condition1**.
 - In the Condition detail section, in the **Service** field, select or type the flow service name *TestMyAdapterFlowService*.
 - Click  to insert the Document Type *TestMyPollingNotificationPublishDocument*.

Conditions

Name	Service	Document Types	Join Type
Condition1	TestMyAdapter:TestMyAdapterFlowService	TestMyPollingNotificationPublishDocument	N/A

Condition detail

Name:

Service:

Document Type	Connection Alias	Filter	Provider Filter (UM only)
TestMyAdapter:TestMyPollingNotificationPublishDocument	DEFAULT(IS_LOCAL_CONNECTION)		





Join type: All (AND) Any (OR) Only one (XOR)

Trigger Settings | Comments

d. Save the messaging trigger.

15. Schedule and enable the polling notification.

a. In Integration Server Administrator, select **Adapters > MyAdapter > Polling Notifications**, the list of polling notifications appear.

DemoAdapter Polling Notifications						
Notification Name ▲ ▼	Package Name ▲ ▼	State ▲ ▼	Edit Schedule	View Schedule	Publish Events	Run As User
TestMyAdapter:TestMyAdapterNotification	Default	Disabled ▼				Administrator 

b. Select **Edit Schedule** for **TestMyPollingNotification**.

c. Modify the schedule parameters for the **TestMyPollingNotification**.

Details for TestMyAdapter:TestMyAdapterNotification	
Interval: (seconds)	<input type="text" value="10"/>
Overlap:	<input type="checkbox"/>
Immediate:	<input checked="" type="checkbox"/>
Cluster settings	
Coordination mode:	Standby
Max process time: (seconds)	180
Max setup time: (seconds)	180

- d. Select **Save Settings**.
- e. In **Adapters > MyAdapter > Polling Notifications** screen, select **State** as **Enabled**.

Notification Name ▲ ▼	Package Name ▲ ▼	State ▲ ▼	Edit Schedule	View Schedule	Publish Events	Run As User
TestMyAdapter:TestMyAdapterNotification	Default	<input type="button" value="Enabled"/>			No	Administrator

1

16. Add a file to the folder that is monitored. In this example, the folder is C:\Monitor and file is added Testing-1.txt.

The file *MonitorPollingNotificationPipeline.log* is created in *Integration Server_directory / instances/<instance_name>/pipeline*. This file contains the following entry for the file added in C:\Monitor:

```
<?xml version="1.0" encoding="UTF-8"?>
<IDataXMLCoder version="1.0">
  <record javaclass="com.wm.data.ISMemDataImpl">
    <value name="fileName">MonitorPollingNotificationPipeline.log</value>
    <record name="TestMyAdapter:TestMyAdapterNotificationPublishDocument"
  javaclass="com.wm.data.ISMemDataImpl">
    <value name="FileName">Testing-1.txt</value>
    <value name="Path">C:\Monitor</value>
    <jboolean name="isAdded">true</jboolean>
    <jboolean name="isDeleted">>false</jboolean>
    <record name="_env" javaclass="com.wm.data.ISMemDataImpl">
      <value name="locale"></value>
      <value name="activation">wm6bfd23a95-1c84-4e3e-b687-5858453777bc</value>
      <value name="businessContext">wm6:bfd23a95-1c84-4e3e-b687-5858453777bc\snull\
snull:wm6bfd23a95-1c84-4e3e-b687-5858453777bc:null:IS_61:null</value>
      <value name="uuid">wm:c3fb4d30-2f23-11ec-8723-000000000002</value>
      <value name="trackId">wm:c3fb4d30-2f23-11ec-8723-000000000002</value>
      <value name="pubId">islocalpubid</value>
    </record>
  </record>
</IDataXMLCoder>
```

```

    <Date name="enqueueTime" type="java.util.Date">Sun Oct 17 13:55:20 IST
2021</Date>
    <Date name="recvTime" type="java.util.Date">Sun Oct 17 13:55:20 IST 2021</Date>
    <number name="age" type="java.lang.Integer">0</number>
  </record>
</record>
</record>
</IDataXMLCoder>

```

How to create an adapter listener implementation?

An adapter connection connects to an adapter resource. This chapter describes how to create an adapter connection implementation.

Pre-requisites:

- webMethods Integration Server 9.12 or later installed.
- Designer 9.12 or later installed.
- Integration Server Administrator access.
- Java 1.8 or later installed.
- Basic understanding of webMethods Integration Server, Designer, Integration Server Administrator, Java.

1. Start the editor to create Java files for adapter listener implementation.
2. Create directories corresponding to your Java package structure in the webMethods package you created using Designer. For example: `com\mycompany\adapter\myAdapter\listeners`. In the example, the folder created is `com\wm\MyAdapter\Listeners`.

Note:

You must create your Java package and classes in the `adapterPackageName\code\source` folder in the webMethods package you created using Designer.

3. Create the `com.wm.adk.notification.WmConnectedListener` implementation class.

In the example, created a *SimpleListener* class.

- a. Create a class attribute, a set method, a constant, and a resource domain for each metadata parameter. For example, a constant `FILE_NAME_PARM`, a corresponding class attribute `_fileName`, a corresponding set method `setFileName`.

Class Attribute Name	Class Attribute Set Method Name	Class Attribute Name Constant	Resource Domain Name Constant
<code>_fileName</code>	<code>setFileName</code>	<code>FILE_NAME_PARM</code>	None

- b. In the `fillWmDescriptor` method, call `WmDescriptor.setRequired` method, and `WmDescriptor.setDescriptions` method.
- c. In the `listenerStartup` method, call `retrieveConnection` method to retrieve connection object and use it to read the file.
- d. Implement the `waitForData` method.
- e. In the `listenerShutdown` method, close the `FileReader` object.
- f. Implement the `processNotificationResults` method.

For example: the *SimpleListener* class:

```
package com.wm.MyAdapter.listeners;
import com.wm.adk.error.AdapterException;
import com.wm.adk.metadata.WmDescriptor;
import com.wm.adk.notification.WmConnectedListener;
import com.wm.adk.notification.NotificationResults;
import java.io.FileReader;
import java.util.Locale;
import javax.resource.ResourceException;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.connections.SimpleConnection;

public class SimpleListener extends WmConnectedListener
{
    public static final String FILE_NAME_PARM = "fileName";
    private String _fileName = null;
    public void setFileName(String val){_fileName = val;}
    private FileReader _reader = null;
    private StringBuffer workingBuffer = new StringBuffer();
    private String _lastDataObject = null;
    public void fillWmDescriptor(WmDescriptor descriptor, Locale locale)
        throws ResourceException
    {
        descriptor.setRequired(FILE_NAME_PARM);
        descriptor.setDescriptions(
            MyAdapter.getInstance().getAdapterResourceBundleManager(), locale);
    }

    public void listenerStartup() throws ResourceException
    {
        try
        {
            //_reader = ((SimpleConnection)retrieveConnection()).getReader();
            _reader = ((SimpleConnection)retrieveConnection()).getReader(_fileName);
            while(_reader.ready())
            {
                _reader.read(); // move to the end of the stream
            }
        }
        catch (Throwable t)
        {
            throw MyAdapter.getInstance().createAdapterException(100,t);
        }
    }
}
```

```

}
public Object waitForData() throws ResourceException
{
    try
    {
        if(_reader.ready())
        {
            do
            {
                int i = _reader.read();
                if (i != -1)
                {
                    char c = (char)i;
                    workingBuffer.append(c);
                    if(c == '\n')
                    {
                        _lastDataObject = new String(workingBuffer);
                        workingBuffer = new StringBuffer();
                        break;
                    }
                }
            }
            else
            {
                break;
            }
        } while (_reader.ready());
    }
    catch (Throwable t)
    {
        throw MyAdapter.getInstance().createAdapterException(100,t);
    }
    return _lastDataObject;
}

```

```

public void listenerShutdown()
{
    try
    {
        _reader.close();
    }
    catch(Throwable t){}
}
public void processNotificationResults(NotificationResults results)
    throws ResourceException
{
    if(results != null)
    {
        if(results.hadError())
        {
            MyAdapter.getLogger().logError(9999,
                "Error processing: " + this._lastDataObject +
                " errorInfo = " + (results.getErrorInfo() == null ? "" :
results.getErrorInfo().toString()));
        }
    }
    else
    {
        MyAdapter.getLogger().logError( 9999,
            "No notification available to process:" +

```



```

        this._lastDataObject);
    }
}
}

```

4. Add a `getReader` method in the `WmManagedConnection` implementation class.

```

package com.wm.MyAdapter.connections;
..
..
import java.io.FileReader;
import java.io.FileNotFoundException;
..
..
public class SimpleConnection extends WmManagedConnection {
..
..
    public FileReader getReader(String fileName) throws AdapterException
    {
        FileReader _reader = null;

        try {
            _reader = new FileReader(fileName);
        }
        catch(Exception e) {
            throw MyAdapter.getInstance().createAdapterException(100,e);
        }
        return _reader;
    }
}

```

5. Update the resource bundle implementation class to add the display name and description of the listener class and the fields in the listener class.

In the example, update `MyAdapterResource` class's `Object[][] _contents` as follows:

```

package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.listeners.SimpleListener;
..
..
public class MyAdapterResource extends ListResourceBundle implements
MyAdapterConstants{
    ..
    ..
    static final Object[][] _contents = {
        ..
        ..
        //Listener
        ,{SimpleListener.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Simple Listener"}
        ,{SimpleListener.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Use to monitor log files"}
        ,{SimpleListener.FILE_NAME_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Log File Name"}
    }
    protected Object[][] getContents() {

```

```
// TODO Auto-generated method stub
return _contents;
}
}
```

6. Register the listener type to the adapter by updating your `fillAdapterTypeInfo` method in your `WmAdapter` implementation class.

```
package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.listeners.SimpleListener;
..
..
public class MyAdapter extends WmAdapter implements MyAdapterConstants {
..
..
public void fillAdapterTypeInfo(AdapterTypeInfo info, Locale locale)
{
..
..
info.addListenerType(SimpleListener.class.getName());
}
}
```

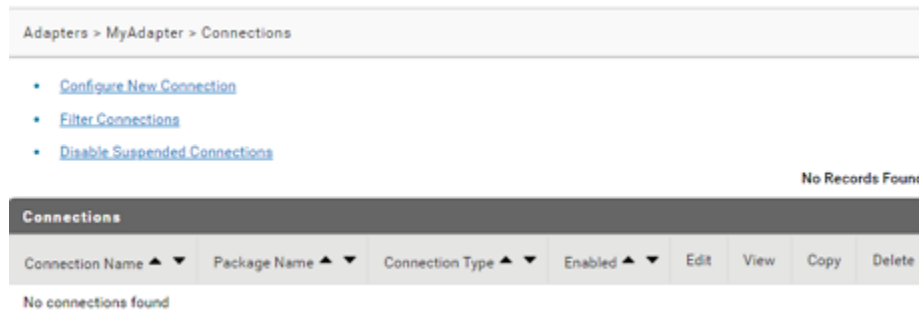
7. Execute the ANT script created in adapter definition to compile, and deploy the adapter in Integration Server.

Use the files `build.xml` and `build.properties`.

```
ant deploy
```

8. Restart Integration Server.
9. Start Integration Server Administrator.
10. In Integration Server Administrator, select **Adapters > MyAdapter**.

The adapters' connection configured are listed.



11. In the **Adapters > MyAdapter > Connections** screen, select **Configure New Connections**.

The adapters' connection types are listed.

Adapters > MyAdapter > Connection Types

- [Return to MyAdapter Connections](#)

Connection Types	
Connection Type	Description
Simple Connection	Simple framework for demonstration purposes

12. In the **Adapters > MyAdapter > Connection Types** screen, select the **Connection Type Simple Connection**.

The adapters' connection properties to configure are listed:

Adapters > MyAdapter > Configure Connection Type

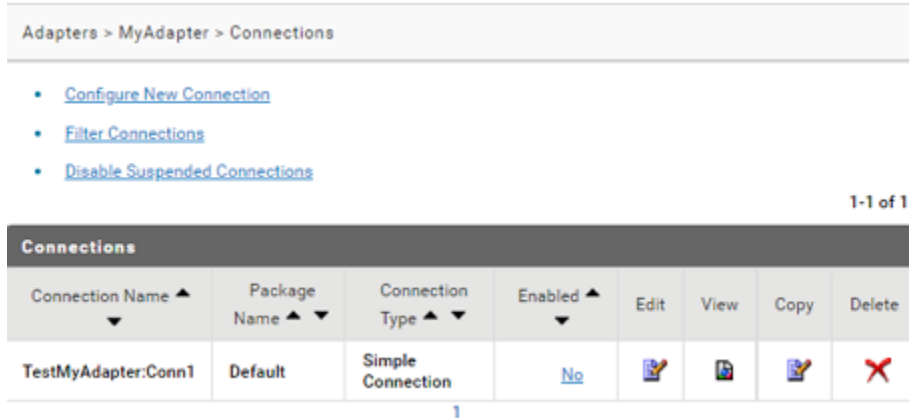
- [Return to MyAdapter Connection Types](#)

Configure Connection Type > MyAdapter

Package	Default ▾
Folder Name	<input type="text"/>
Connection Name	<input type="text"/>
Connection Properties	
Host Name	<input type="text"/>
Port	5555 ▾
Connection Management Properties	
Enable Connection Pooling	true ▾
Minimum Pool Size	<input type="text" value="1"/>
Maximum Pool Size	<input type="text" value="10"/>
Pool Increment Size	<input type="text" value="1"/>
Block Timeout (msec)	<input type="text" value="1000"/>
Expire Timeout (msec)	<input type="text" value="1000"/>
Startup Retry Count	<input type="text" value="0"/>
Startup Backoff Timeout (sec)	<input type="text" value="10"/>
<input type="button" value="Save Connection"/> <input type="button" value="Test Connection"/>	

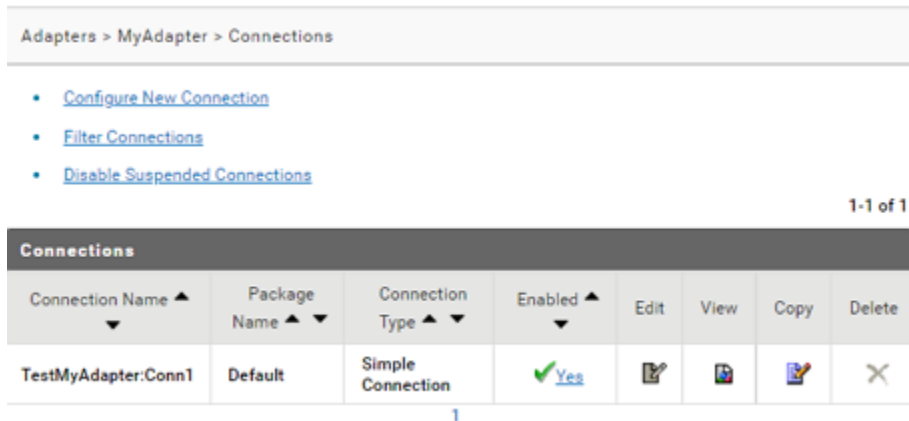
13. In the **Adapters > MyAdapter > Configure Connection Type** screen, add the details and **Save Connection**.

The **Adapters > MyAdapter > Connections** screen lists the connection you added.



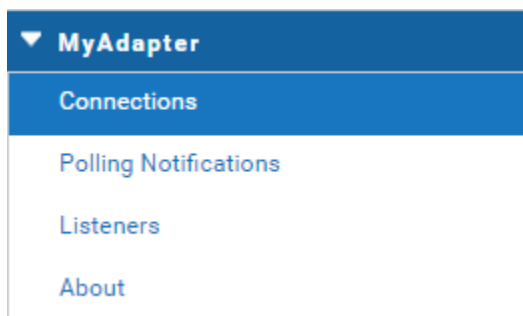
14. In the **Adapters > MyAdapter > Connections** screen, click **No** in the **Enabled** column.

The **Enabled** column now shows **Yes**.



15. In Integration Server Administrator, select **Adapters > MyAdapter**.

The adapters' menu lists **Listeners**.



16. In the **Adapters > MyAdapter > Listeners** screen, select **Configure new listener**.

The adapters' listener types are listed.

Adapters > MyAdapter > Listener Types

- [Return to MyAdapter Listeners](#)

Listener Types	
Listener Type	Description
Simple Listener	Use to monitor log files

17. In the **Adapters > MyAdapter > Configure Listener Type** screen, select the **Listener Type Simple Listener**.

The adapters' **Simple Listener** properties to configure are listed:

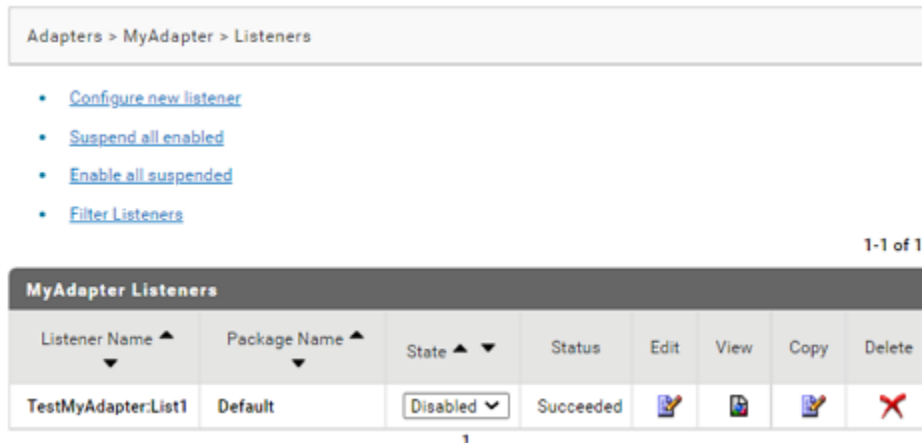
Adapters > MyAdapter > Configure Listener Type

- [Return to MyAdapter Listener Types](#)

Configure Listener Type > MyAdapter	
Package	Default ▾
Folder Name	<input type="text"/>
Listener Name	<input type="text"/>
Connection name	TestMyAdapter.Conn1 ▾
Retry Limit	<input type="text" value="5"/>
Retry Backoff Timeout	<input type="text" value="10"/>
Listener Properties	
Log File Name	<input type="text"/>

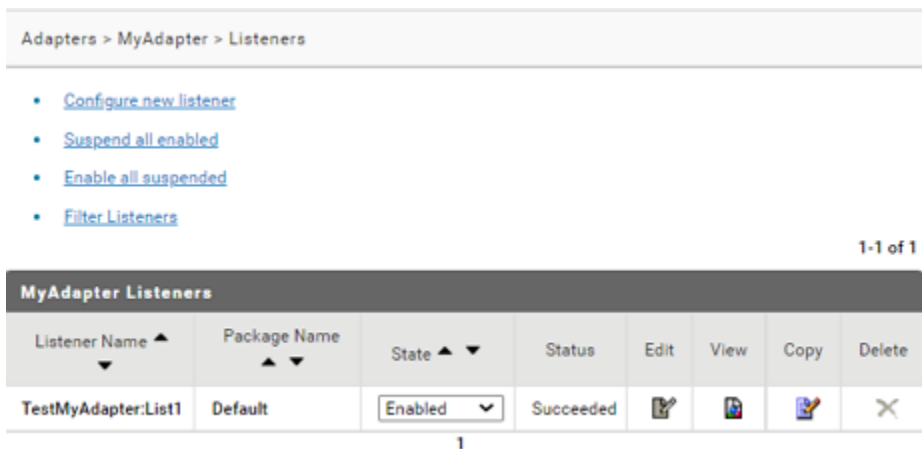
18. In the **Adapters > My Adapter > Configure Listener Type** screen, add the details (File path of the log on which you want to add the listener. For example: C:\softwareag\107\IntegrationServer\instances\default\logs\server.log) and **Save Listener**.

The **Adapters > My Adapter > Listener** screen lists the listener you added.



19. In the **Adapters > MyAdapter > Listener** screen, select **Enabled** in the **Status** column.

The **Listener** column now shows **Enabled**.



20. Create a class by extending `com.wm.adk.notification.WmAsyncListenerNotification` base class.

In the example, created a `SessionLogListenerNotification` class.

- a. Create a class attribute, a set method, a constant, and a resource domain for each metadata parameter. For example, a constant `FIELD_NAMES_PARM`, a corresponding class attribute `_fileName`, a corresponding set method `setFileName`.

Class Attribute Name	Class Attribute Set Method Name	Class Attribute Name Constant	Resource Domain Name Constant
<code>_fieldNames</code>	<code>setFieldNames</code>	<code>FIELD_NAMES_PARM</code>	<code>FIELD_NAMES_RD</code>
<code>_fieldTypes</code>	<code>setFieldTypes</code>	<code>FIELD_TYPES_PARM</code>	<code>FIELD_TYPES_RD</code>
<code>_uses</code>	<code>setUses</code>	<code>USES_PARM</code>	None
None	<code>setSignature</code>	<code>SIG_PARM</code>	None

- b. In the `fillWmTemplateDescriptor` method, call `WmTemplateDescriptor.createGroup` method, `WmTemplateDescriptor.createFieldMap` method, and `WmTemplateDescriptor.createTuple` method.
- c. In the `fillWmTemplateDescriptor` method, call `WmTemplateDescriptor.setResourceDomain` method, and `WmTemplateDescriptor.setDescriptions` method.

Note:

Use of `WmTemplateDescriptor.setResourceDomain` for `SIG_PARM`:

```
descriptor.setResourceDomain(SIG_PARM,
    WmTemplateDescriptor.OUTPUT_FIELD_NAMES,new String[]{
        FIELD_NAMES_PARM,FIELD_TYPES_PARM}, USES_PARM);
```

- d. Implement the `adapterCheckValue` method to return `true`.
- e. Implement the `adapterResourceDomainLookup` method.
- f. In the `registerResourceDomain` method, call `addResourceDomainLookup` method for `FIELD_NAMES_RD`, and `FIELD_TYPES_RD`.
- g. Implement the `supports` method to parse the data and return `true`.
- h. Implement the `runNotification` method.

For example, the `SessionLogListenerNotification` class:

```
package com.wm.MyAdapter.listeners;
import com.wm.adk.error.AdapterException;
import com.wm.adk.metadata.WmDescriptor;
import com.wm.adk.notification.WmAsyncListenerNotification;
import com.wm.adk.notification.NotificationResults;
import com.wm.adk.notification.AsyncNotificationResults;
import com.wm.adk.notification.NotificationEvent;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.*;
import com.wm.adk.cci.record.WmRecord;
import com.wm.adk.cci.record.WmRecordFactory;
import javax.resource.ResourceException;
import java.util.Locale;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.StringTokenizer;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.MyAdapterConstants;
import com.wm.MyAdapter.connections.SimpleConnection;

public class SessionLogListenerNotification extends WmAsyncListenerNotification
implements MyAdapterConstants
{
    private String[] _fieldNames = null;
    private String[] _fieldTypes = null;
    private boolean[] _uses = null;
    public void setFieldNames(String[] val){_fieldNames = val;}
```

```

public void setFieldTypes(String[] val){_fieldTypes = val;}
public void setUses (boolean[] val){_uses = val;}
public void setSignature(String[] val){}
public static final String NOTIFICATION_SETUP_GROUP =
"SessionLogListenerNotification.setup";
public static final String FIELD_NAMES_PARM = "fieldNames";
public static final String FIELD_TYPES_PARM = "fieldTypes";
public static final String USES_PARM = "uses";
public static final String SIG_PARM = "signature";
public static final String FIELD_NAMES_RD =
"SessionLogListenerNotification.fieldNames.rd";
public static final String FIELD_TYPES_RD =
"SessionLogListenerNotification.fieldTypes.rd";
public static final String[] _sigFieldNames = {
"timeStamp",
"component",
"rootContext",
"parentContext",
"currentContext",
"server",
"eventCode",
"user",
"sessionName",
"RPCs",
"age"};
private Object[] _parsedValues = new Object[_sigFieldNames.length];

```

```

public void fillWmTemplateDescriptor(WmTemplateDescriptor descriptor,Locale l)
throws ResourceException
{
String[] parms = new String[] {FIELD_NAMES_PARM,
FIELD_TYPES_PARM,
USES_PARM,
SIG_PARM};
descriptor.createGroup(NOTIFICATION_SETUP_GROUP, parms);
descriptor.createFieldMap(parms, false);
descriptor.createTuple(new String[]{FIELD_NAMES_PARM, FIELD_TYPES_PARM});
descriptor.setResourceDomain(FIELD_NAMES_PARM, FIELD_NAMES_RD, null);
descriptor.setResourceDomain(FIELD_TYPES_PARM, FIELD_TYPES_RD, null);
descriptor.setResourceDomain(SIG_PARM,
WmTemplateDescriptor.OUTPUT_FIELD_NAMES,new String[]{
FIELD_NAMES_PARM,FIELD_TYPES_PARM}, USES_PARM);
descriptor.setDescriptions( MyAdapter.getInstance().
getAdapterResourceBundleManager(), l);
}
public Boolean adapterCheckValue(WmManagedConnection connection,
String resourceDomainName, String[][] values,
String testValue) throws AdapterException
{
return true;
}
public ResourceDomainValues[] adapterResourceDomainLookup(
WmManagedConnection connection, String resourceDomainName,
String[][] values) throws AdapterException
{
ResourceDomainValues[] results = null;
if (resourceDomainName.equals(FIELD_NAMES_RD)
|| resourceDomainName.equals(FIELD_TYPES_RD))
{
ResourceDomainValues names = new ResourceDomainValues(

```



```

    FIELD_NAMES_RD, _sigFieldNames);
ResourceDomainValues types = new ResourceDomainValues(
    FIELD_TYPES_RD,new String[] {
        Date.class.getName(), //timestamp
        String.class.getName(), // component
        String.class.getName(), // rootContext
        String.class.getName(), // parentContext
        String.class.getName(), // currentContext
        String.class.getName(), // server
        Integer.class.getName(), // eventCode
        String.class.getName(), // user
        String.class.getName(), // sessionId
        Integer.class.getName(), // RPCs
        Long.class.getName() // age
    });
    results = new ResourceDomainValues[] {names,types};
}
return results;
}
public void registerResourceDomain(WmManagedConnection connection,
    WmAdapterAccess access) throws AdapterException
{
    access.addResourceDomainLookup(this.getClass().getName(),
        FIELD_NAMES_RD, connection);
    access.addResourceDomainLookup(this.getClass().getName(),
        FIELD_TYPES_RD, connection);
}
public boolean supports(Object data) throws ResourceException
{
    boolean result = false;
    try
    {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd H:mm:ss zzz");
        String sData = (String)data;
        this._parsedValues[0] = sdf.parse(sData.substring(0,48));
        StringTokenizer st = new StringTokenizer(sData.substring(49)," ",false);
        this._parsedValues[1] = st.nextToken();
        this._parsedValues[2] = st.nextToken();
        this._parsedValues[3] = st.nextToken();
        this._parsedValues[4] = st.nextToken();
        /*
        st.nextToken(); // skip the session ID
        this._parsedValues[5] = st.nextToken();
        this._parsedValues[6] = new Integer(st.nextToken());
        this._parsedValues[7] = st.nextToken();
        this._parsedValues[8] = st.nextToken();
        this._parsedValues[9] = new Integer(st.nextToken());
        this._parsedValues[10] = new Long(st.nextToken());
        */
        result = true;
    }
    catch(Throwable t){}
    return result;
}

public NotificationResults runNotification(NotificationEvent event)
    throws ResourceException
{
    NotificationResults result = null;
    WmRecord notice = WmRecordFactory.getFactory().createWmRecord("notUsed");

```

```

for(int i = 0; i < _sigFieldNames.length;i++)
{
    if (_uses[i])
    {
        notice.put(_sigFieldNames[i],_parsedValues[i]);
    }
}
this.doNotify(notice);
result = new AsyncNotificationResults(this.nodeName(),true,null);
return result;
}
}

```

21. Update the `WmManagedConnection` implementation class, `SimpleConnection`.

- a. Update the `WmManagedConnection.registerResourceDomain` method to register the listener notification metadata parameters for lookup.

In the example, update `SimpleConnection.registerResourceDomain` method as follows:

```

package com.wm.MyAdapter.connections;
..
..
import com.wm.MyAdapter.listeners.SessionLogListenerNotification;
..
..
public class SimpleConnection extends WmManagedConnection {
..
..
public void registerResourceDomain(WmAdapterAccess access)
    throws AdapterException
{
    ..
    ..
    //SessionLog Listener Notification Registering Resource Domain
    access.addResourceDomainLookup(SessionLogListenerNotification.FIELD_NAMES_RD,
    this);
    access.addResourceDomainLookup(SessionLogListenerNotification.FIELD_TYPES_RD,
    this);
}
}
}

```

- b. Update the `adapterResourceDomainLookup` method to add the resource domain lookup of the listener notification metadata parameters.

In the example, update `SimpleConnection.adapterResourceDomainLookup` method as follows:

```

package com.wm.MyAdapter.connections;
..
..
import java.util.Date;
..
import com.wm.MyAdapter.listeners.SessionLogListenerNotification;
..
..
public class SimpleConnection extends WmManagedConnection {
..
..

```

```

..
public ResourceDomainValues[] adapterResourceDomainLookup(String serviceName,
String resourceDomainName, String[][] values) throws AdapterException
{
    ..
    ..
    //Listener notification
    else if
    (resourceDomainName.equals(SessionLogListenerNotification.FIELD_NAMES_RD)
    || resourceDomainName.equals(SessionLogListenerNotification.FIELD_TYPES_RD))
    {
        ResourceDomainValues names = new ResourceDomainValues(
            SessionLogListenerNotification.FIELD_NAMES_RD,
            SessionLogListenerNotification._sigFieldNames);
        ResourceDomainValues types = new ResourceDomainValues(
            SessionLogListenerNotification.FIELD_TYPES_RD, new String[] {
                Date.class.getName(), //timestamp
                String.class.getName(), // component
                String.class.getName(), // rootContext
                String.class.getName(), // parentContext
                String.class.getName(), // currentContext
                String.class.getName(), // server
                Integer.class.getName(), // eventCode
                String.class.getName(), // user
                String.class.getName(), // sessionName
                Integer.class.getName(), // RPCs
                Long.class.getName() // age
            });
        results = new ResourceDomainValues[] {names,types};
    }
}

```

22. Update the resource bundle implementation class to add the display name and description of the listener notification class and the fields in the listener notification class.

In the example, update *MyAdapterResource* class's `Object[][] _contents` as follows:

```

package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.listeners.SessionLogListenerNotification;
..
..
public class MyAdapterResource extends ListResourceBundle implements
MyAdapterConstants{
    ..
    ..
    static final Object[][] _contents = {
        ..
        ..
        //SessionLog Listener Notification
        ,{SessionLogListenerNotification.class.getName() +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "SessionLog Listener Notification"}
        ,{SessionLogListenerNotification.class.getName() +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Use SessionLog Listener Notification to monitor log files"}
        ,{SessionLogListenerNotification.FIELD_NAMES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,

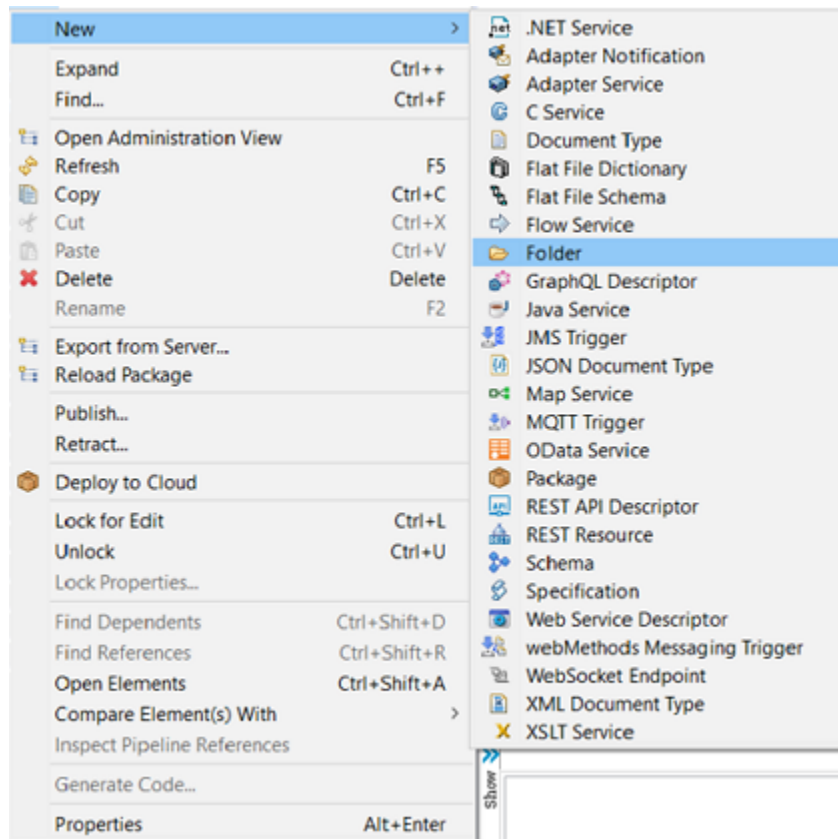
```

```
"Field Name"}
,{SessionLogListenerNotification.FIELD_NAMES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
"Field Name To Check"}
,{SessionLogListenerNotification.FIELD_TYPES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
"Field Types"}
,{SessionLogListenerNotification.FIELD_TYPES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
"Field Types To Check"}
}
protected Object[][] getContents() {
// TODO Auto-generated method stub
return _contents;
}
}
```

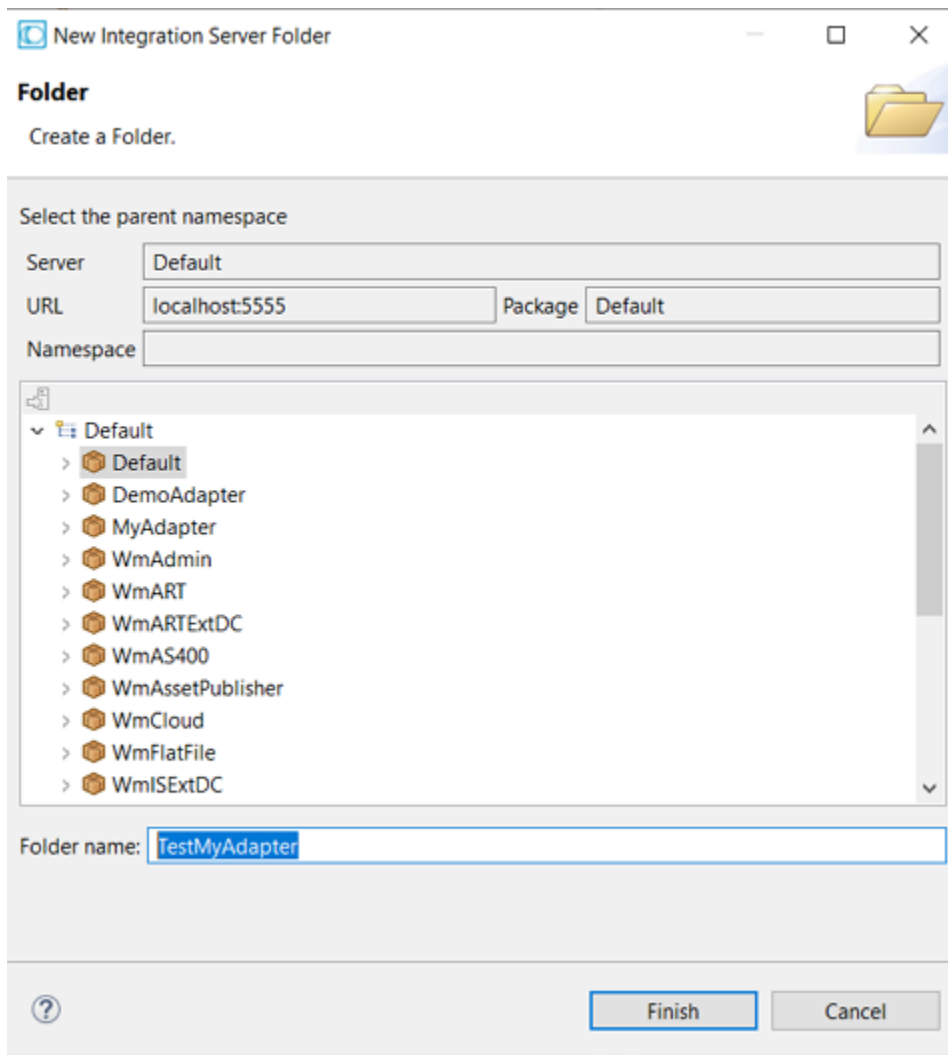
23. Register the listener notification type in the adapter by updating your `fillAdapterTypeInfo` method in your `WmAdapter` implementation class.

```
package com.wm.MyAdapter;
..
..
import com.wm.MyAdapter.listeners.SessionLogListenerNotification;
..
..
public class MyAdapter extends WmAdapter implements MyAdapterConstants {
..
..
public void fillAdapterTypeInfo(AdapterTypeInfo info, Locale locale)
{
..
..
info.addNotificationType(SessionLogListenerNotification.class.getName());
}
}
```

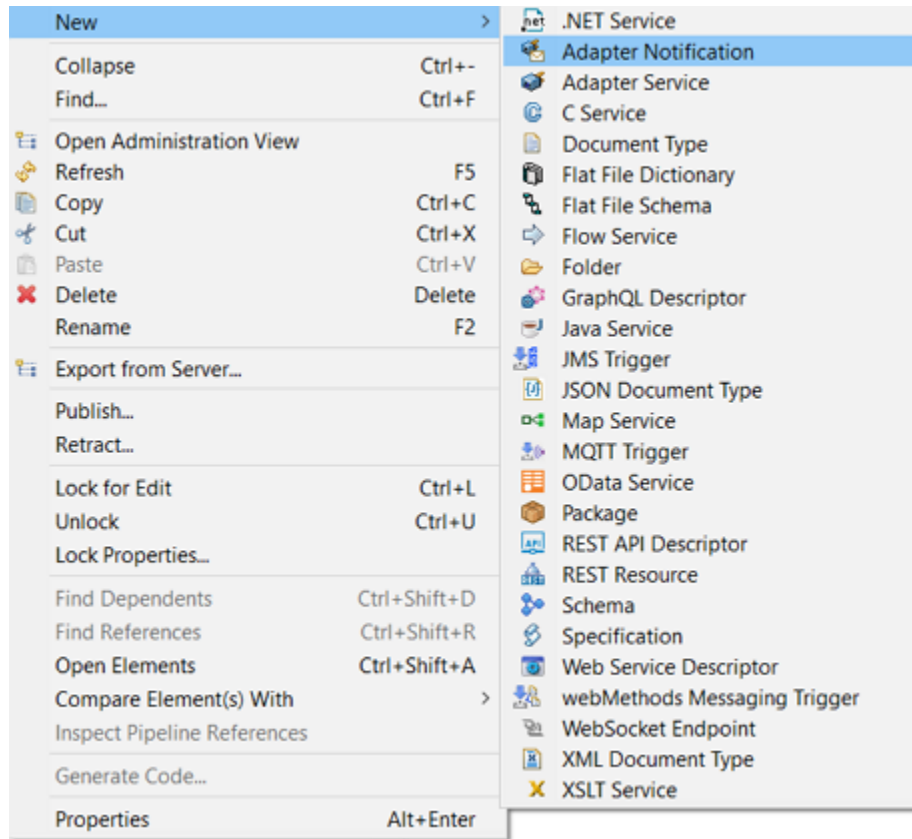
24. In Designer, create the **Adapter Notification**.
- In **Package Navigator**, select the **Default** package.
 - Select **File > New > Folder**.



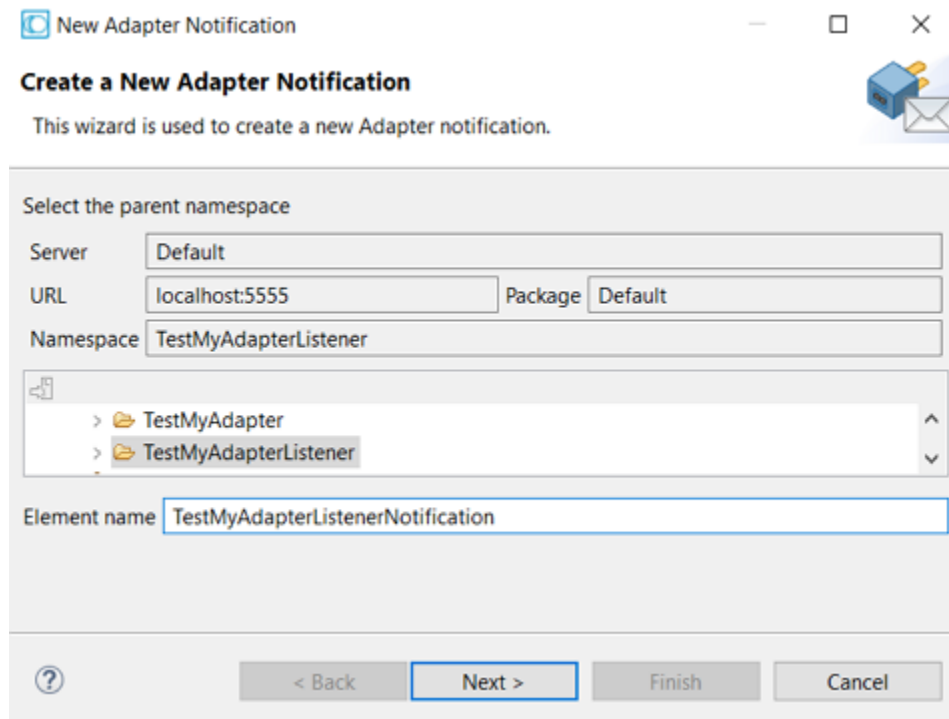
- c. Enter the **Folder name**. For example: *TestMyAdapterListener*.



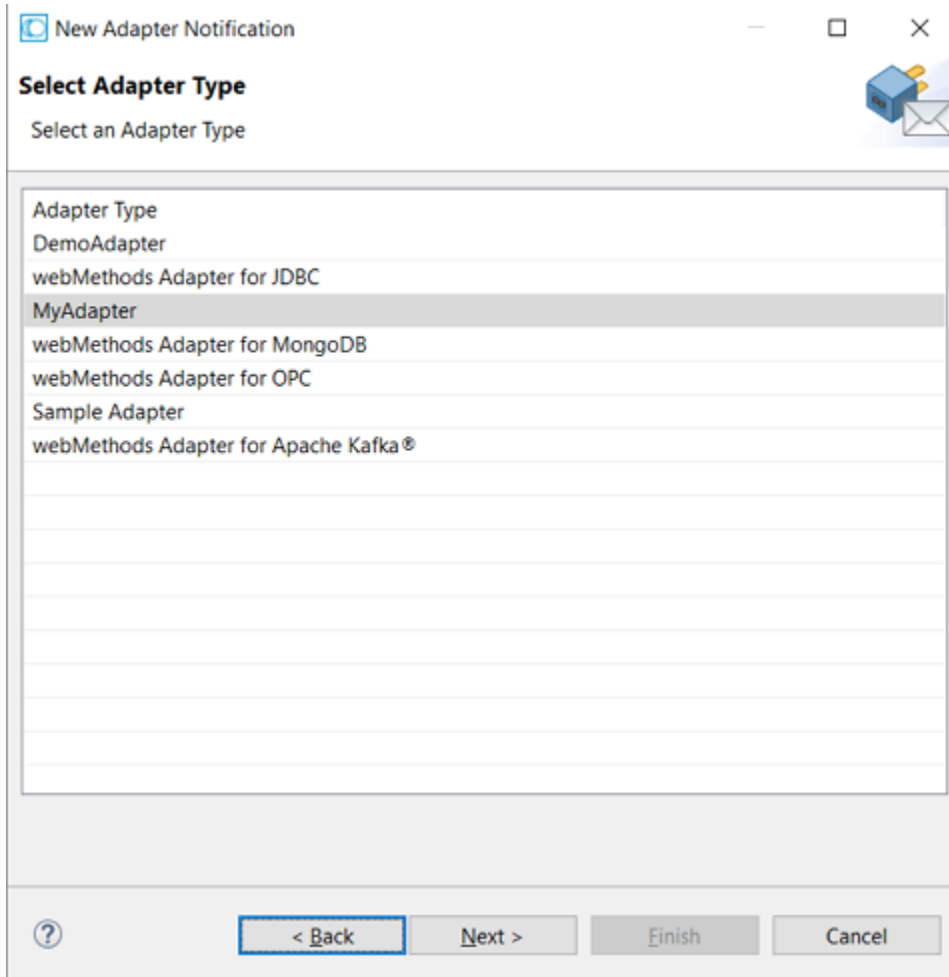
- d. In the **Package Navigator**, select the **Default > TestMyAdapter**.
- e. Select **File > New > Adapter Notification**.



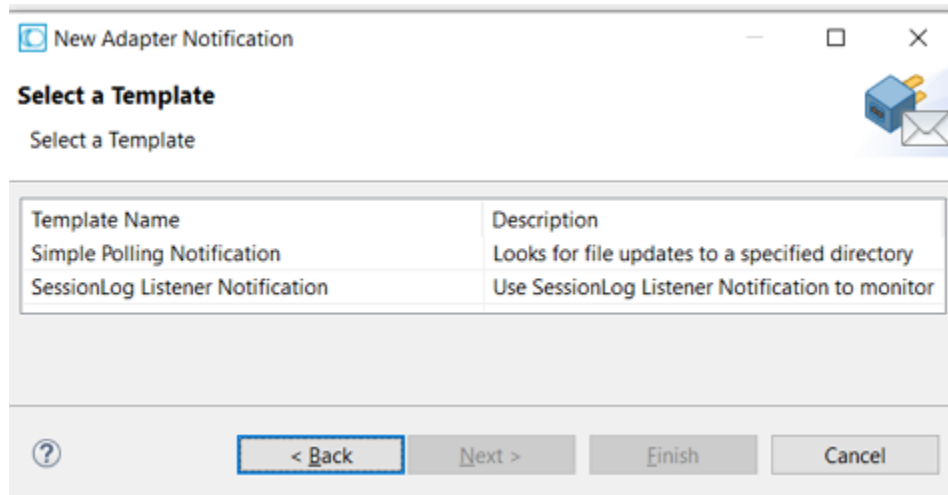
- f. Enter the **Element name** and click **Next**. For example: *TestMyAdapterListenerNotification*.



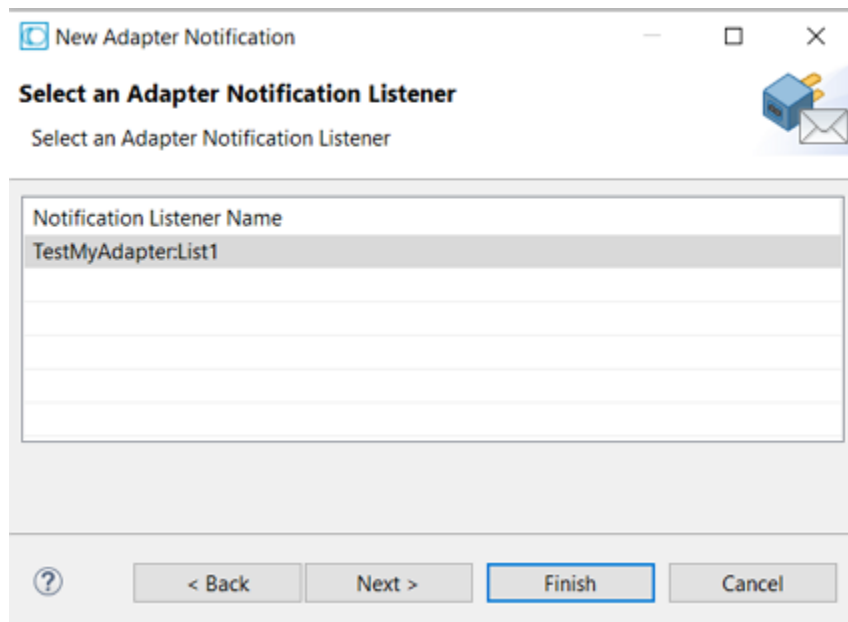
- g. In the **Select Adapter Type** screen, select an adapter type for which you want to create the service. For example: *MyAdapter*.



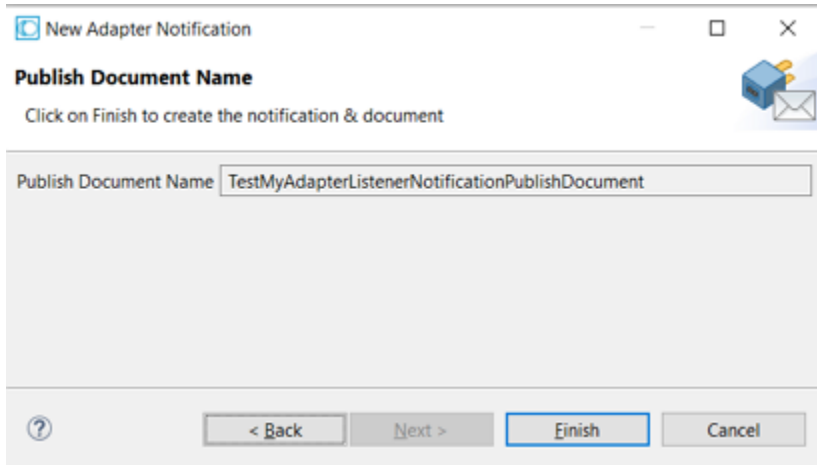
- h. In the **Select a Template** screen, select a listener notification template and click **Finish**. For example: ***SessionLog Listener Notification***.



- i. In the **Select an Adapter Notification Listener** screen, select an adapter listener. For example: *TestMyAdapter:List1*.



- j. In the **Publish Document Name** screen, select **Finish**.

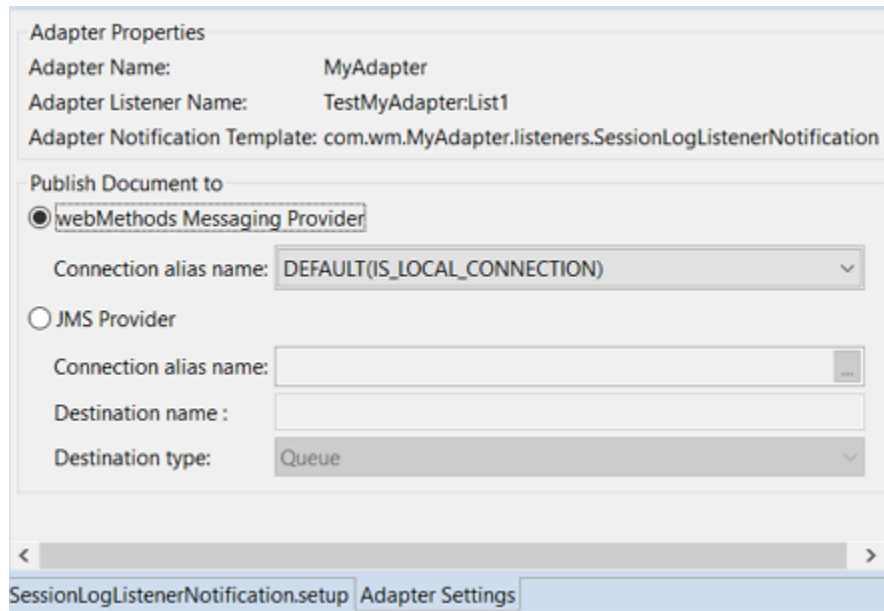


In Designer, you can see the following two items created:

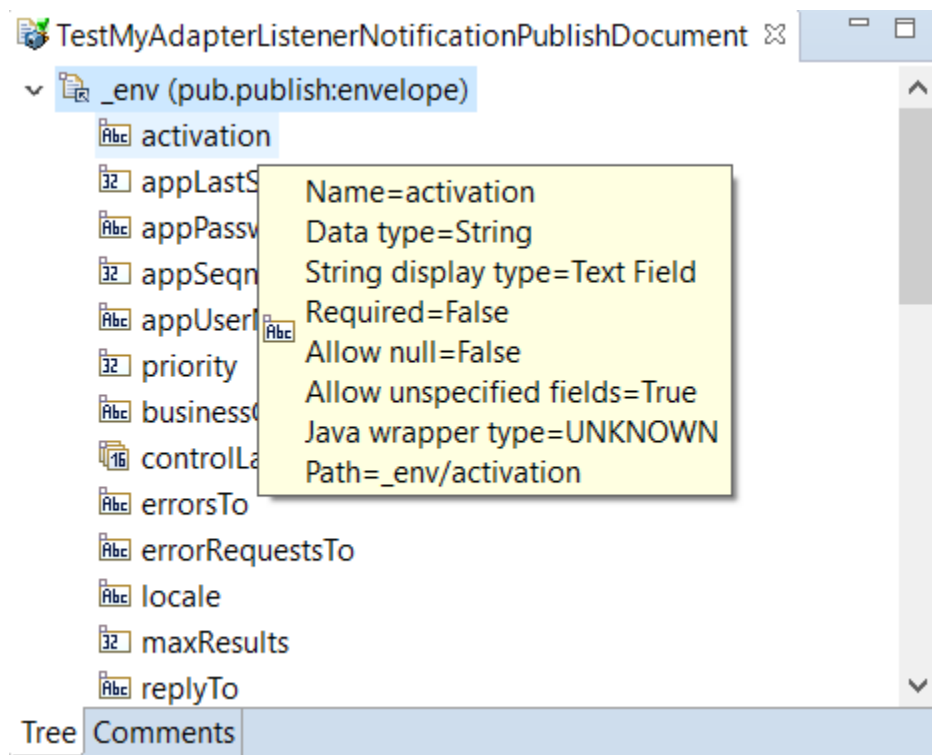
- a. A new adapter notification *TestMyAdapterListenerNotification* is created with two tabs: *SessionLogListenerNotification.setup*, and *Adapter Settings*.

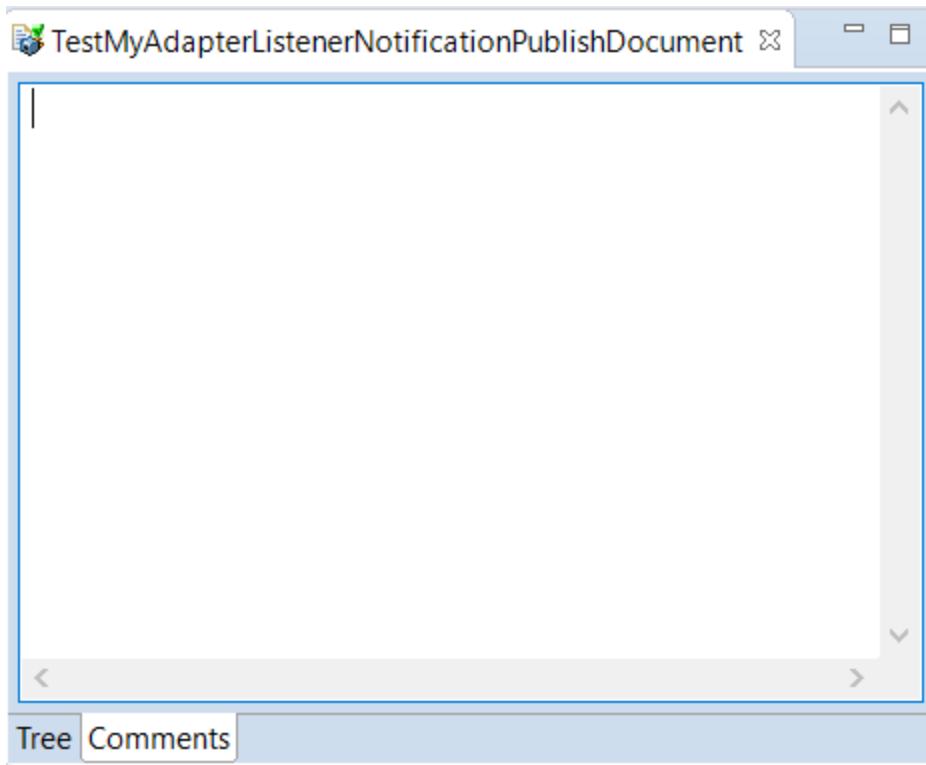
Field Name	Field Types	uses	signature
timeStamp	java.util.Date	<input type="checkbox"/>	
component	java.lang.String	<input type="checkbox"/>	
rootContext	java.lang.String	<input type="checkbox"/>	
parentContext	java.lang.String	<input type="checkbox"/>	
currentContext	java.lang.String	<input type="checkbox"/>	
server	java.lang.String	<input type="checkbox"/>	
eventCode	java.lang.Integer	<input type="checkbox"/>	
user	java.lang.String	<input type="checkbox"/>	
sessionName	java.lang.String	<input type="checkbox"/>	
RPCs	java.lang.Integer	<input type="checkbox"/>	
age	java.lang.Long	<input type="checkbox"/>	

SessionLogListenerNotification.setup Adapter Settings

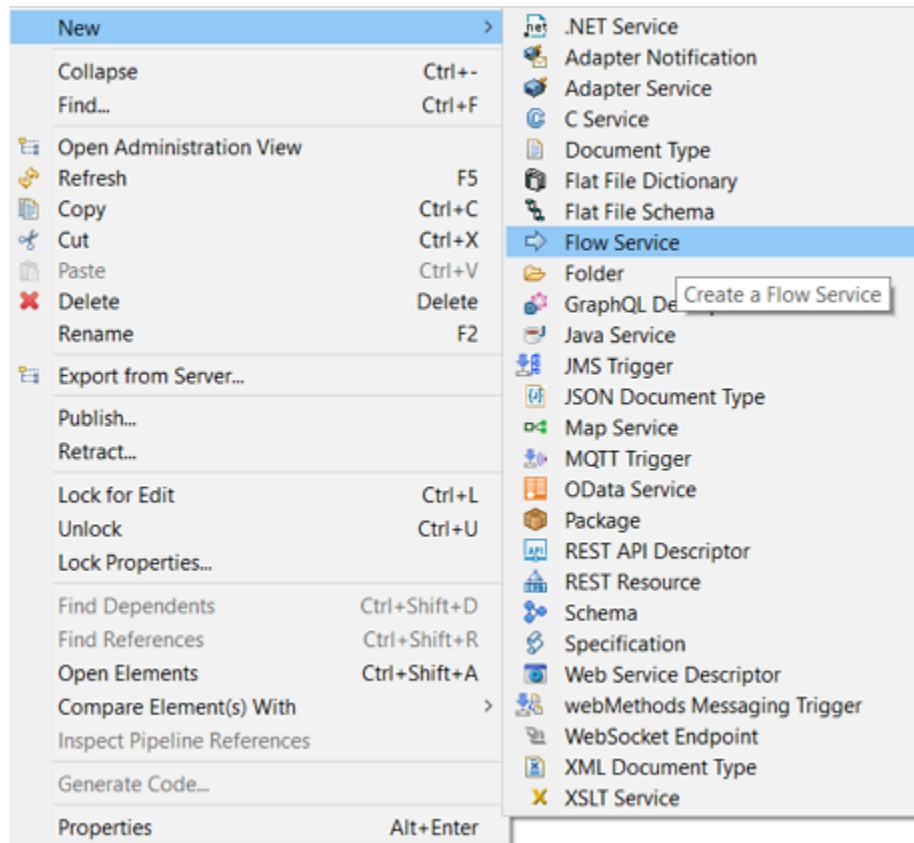


- b. A new Document Type *TestMyAdapterListenerNotificationPublishDocument* is created with two tabs: **Tree**, and **Comments**.

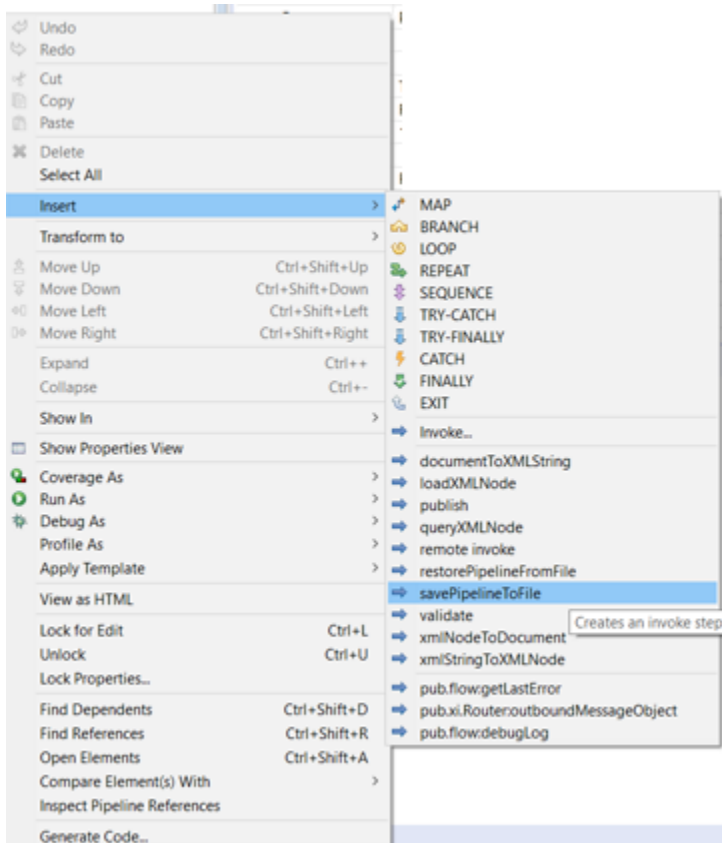




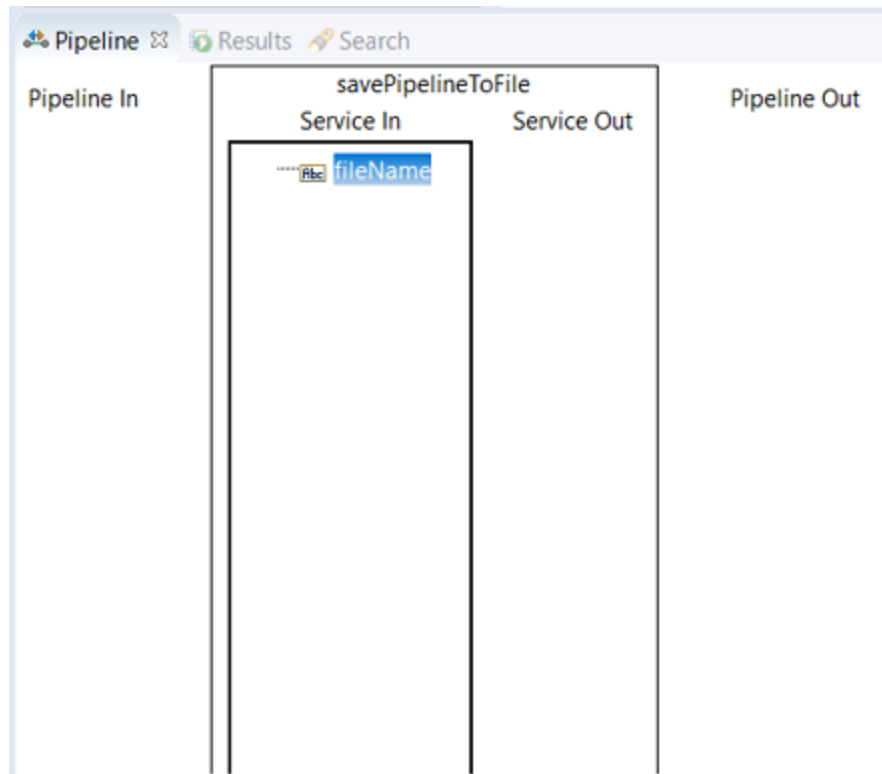
25. Create a Flow Service for the Listener Notification Node using Designer.
 - a. Navigate to the folder **Default > TestMyAdapter**.
 - b. Create the **Flow Service**.



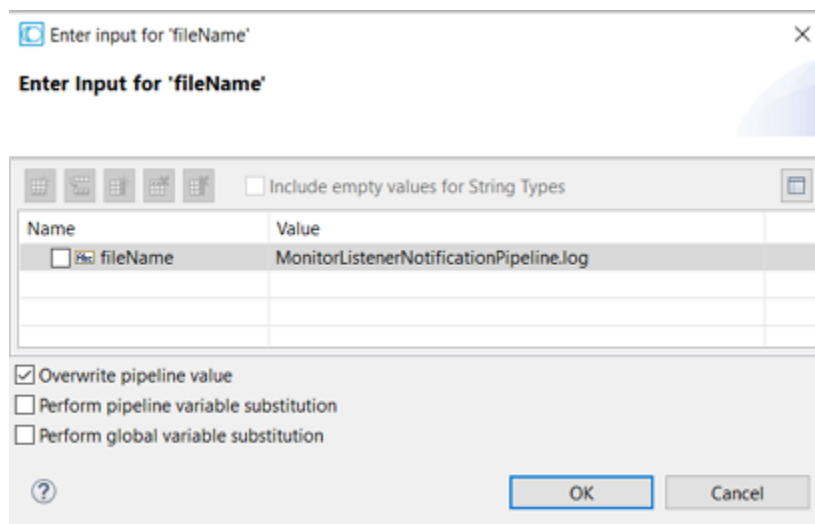
- c. In the **Create a New Flow Service** screen, add *TestMyAdapterFlowService* in the **Element name** field and click **Finish**.
- d. In the **Flow Service > Tree** tab, right-click and select **Insert > savePipelineToFile**.



- e. In the **Flow Service > Tree** tab, select the **savePipelineToFile** method. You can see the **Service In > fileName** in the **Pipeline** tab.



- f. Update **Service In > fileName** in the **Pipeline** tab. In this example, the value is *MonitorListenerNotificationPipeline.log*.

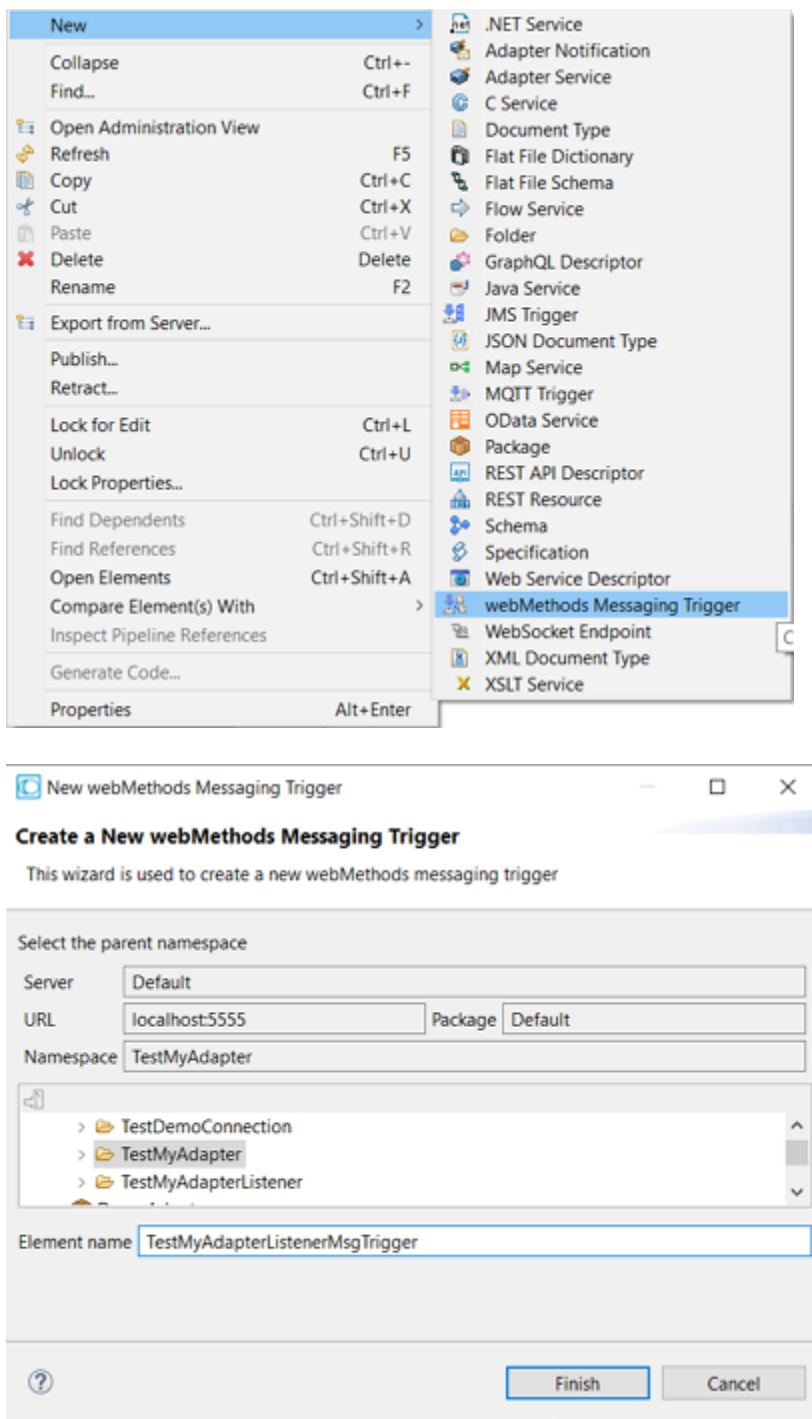


- g. Click **OK**, and save the flow service.


26. Create a trigger using Designer.

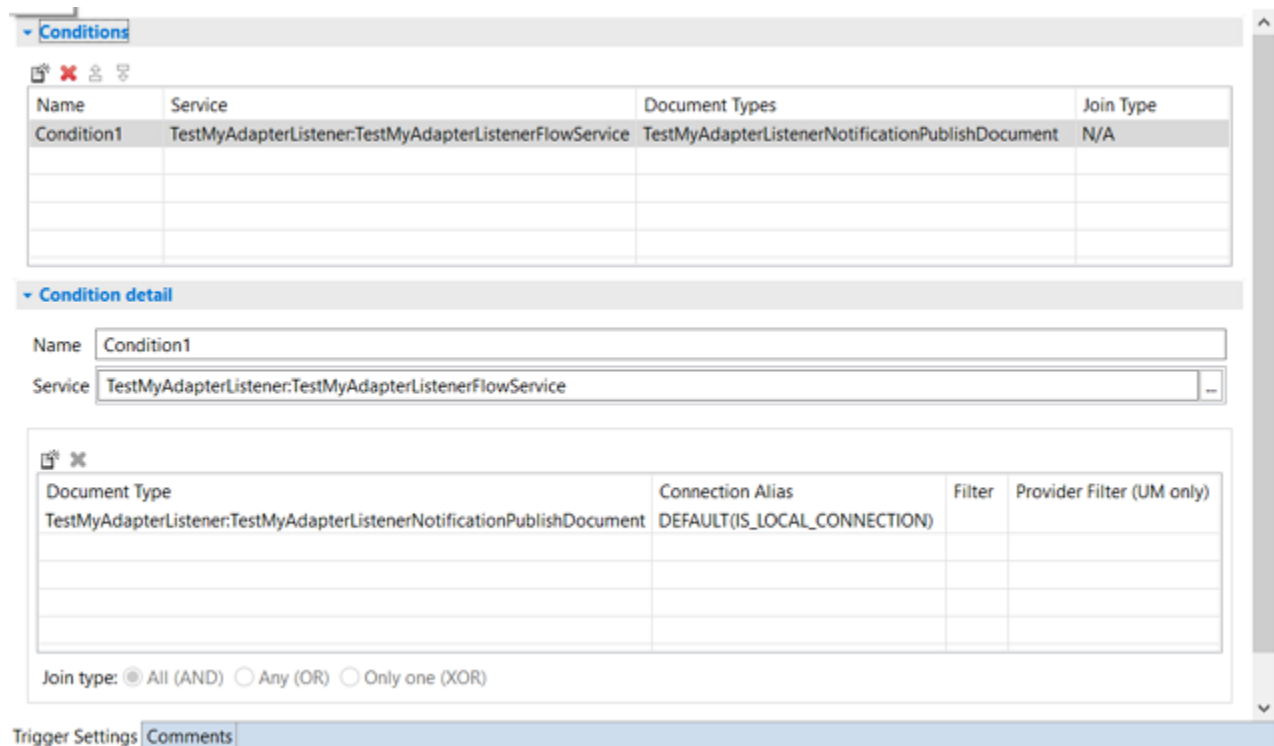
- a. Navigate to the folder **Default > TestMyAdapter**.

- b. Create the **webMethods Messaging Trigger** and select **Finish**. In the example, *TestMyAdapterListenerMsgTrigger* is created.



- c. In the **webMethods Messaging Trigger**, **Trigger Settings** tab, **Condition Detail** section, perform the following:
- In the trigger editor, in the Conditions section, accept the default **Condition1**.

- In the Condition detail section, in the **Service** field, select or type the flow service name *TestMyAdapterFlowService*.
- Click  to insert the Document Type *TestMyAdapterListenerNotificationPublishDocument*.



Name	Service	Document Types	Join Type
Condition1	TestMyAdapterListener:TestMyAdapterListenerFlowService	TestMyAdapterListenerNotificationPublishDocument	N/A

Condition detail

Name: Condition1

Service: TestMyAdapterListener:TestMyAdapterListenerFlowService

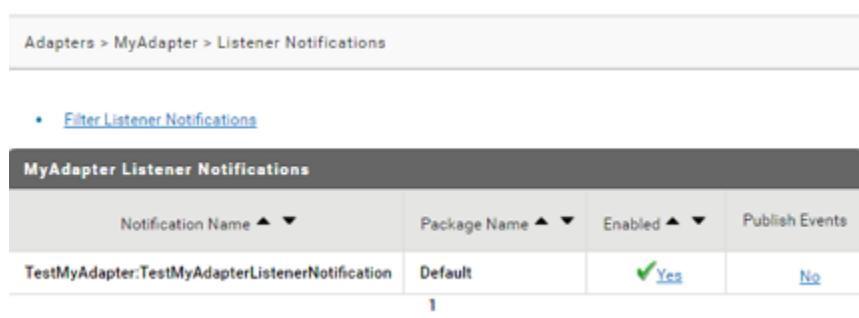
Document Type	Connection Alias	Filter	Provider Filter (UM only)
TestMyAdapterListener:TestMyAdapterListenerNotificationPublishDocument	DEFAULT(IS_LOCAL_CONNECTION)		

Join type: All (AND) Any (OR) Only one (XOR)

d. Save the messaging trigger.

27. Enable the listener notification.

- a. Start Integration Server Administrator
- b. In **Adapters > MyAdapter > Listener Notifications** screen, click **No** in the **Enabled** column for the listener notification.



Adapters > MyAdapter > Listener Notifications

• [Filter Listener Notifications](#)

Notification Name	Package Name	Enabled	Publish Events
TestMyAdapter:TestMyAdapterListenerNotification	Default	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> No

The file *MonitorListenerNotificationPipeline.log* is created in *Integration Server_directory / instances/<instance_name>/pipeline/* folder. This file contains one entry each time the file added in the listener is updated:

```
<?xml version="1.0" encoding="UTF-8"?>
<IDataXMLCoder version="1.0">
  <record javaclass="com.wm.data.ISMemDataImpl">
    <value name="fileName">MonitorListenerNotificationPipeline.log</value>
    <record
name="TestMyAdapterListener:TestMyAdapterListenerNotificationPublishDocument"
javaclass="com.wm.data.ISMemDataImpl">
      <Date name="timeStamp" type="java.util.Date">Thu Oct 21 12:34:45 IST 2021</Date>
      <value name="component"></value>
      <value name="rootContext">Unable</value>
      <record name="_env" javaclass="com.wm.data.ISMemDataImpl">
        <value name="locale"></value>
        <value name="activation">wm624455fb0-b66d-4344-b92f-cee92639b14c</value>
        <value
name="businessContext">wm6:2281c61b-a0dd-4dc2-bb31-6eaea2052a1c\snull\snull:
wm624455fb0-b66d-4344-b92f-cee92639b14c:null:IS_61:null</value>
        <value name="uuid">wm:14c48f70-323e-11ec-b7fd-000000000152</value>
        <value name="trackId">wm:14c48f70-323e-11ec-b7fd-000000000152</value>
        <value name="pubId">islocalpubid</value>
        <Date name="enqueueTime" type="java.util.Date">Thu Oct 21 12:41:15 IST
2021</Date>
        <Date name="recvTime" type="java.util.Date">Thu Oct 21 12:41:15 IST 2021</Date>
        <number name="age" type="java.lang.Integer">0</number>
      </record>
    </record>
  </record>
</IDataXMLCoder>
```

A Alternative Approaches to Metadata

■ Overview	308
■ Implementing Metadata Parameters Using External Classes	308
■ An Alternative Approach to Organizing Resource Domains	308
■ Using Resource Bundles with Resource Domain Values	325

Overview

This chapter describes other capabilities supported by the ADK that are useful, but not required, for implementing an adapter.

Implementing Metadata Parameters Using External Classes

A basic model specifying all adapter metadata parameters is described in [“Metadata Model for Connection” on page 63](#). Integration Server derives a parameter's name and data type from the name of the accessor methods defined in the class in which the parameter is defined.

If a class contains a conforming accessor method that uses an object data type not existing in the listed data types, the adapter interprets that object as being an external container for the metadata parameters. For more information about the data types supported, see *paramType* parameter in [“Overview” on page 344](#).

In this example, the `WmAdapterService` class contains an accessor method:

```
SetParameters(MyServiceParameters value);
```

- Integration Server considers *MyServiceParameters*, an external class that contains accessor methods.
- Integration Server introspects the *MyServiceParameters* class, and derives metadata parameter names from it.

Note:

MyServiceParameters must support a default (no argument) constructor.

- This feature changes the name of the parameter string used in the descriptor methods as well as the resource bundle by prefixing it with the name derived from the method that implements the indirection.

If *MyServiceParameters* class includes a `setFoo(String value)` method, then

- The string that referenced this parameter is `parameters.foo`.
- The *parameters* value is derived from `setParameters` method.
- The *foo* value is derived from `setFoo` method.

An Alternative Approach to Organizing Resource Domains

The model described in this section provides an alternative way of organizing resource domain information, such that the resource domain implementation is contained within each adapter service or notification class that uses the resource domain, rather than within your `WmManagedConnection` implementation. To implement this approach you must perform the following:

- Create an interface defining the methods for resource domain handling. The following methods are defined:

- adapterResourceDomainLookup
- adapterCheckValue
- registerResourceDomain
- Update the connection factory to use a string array containing the class name for each service or notification type added to the adapter.
- Remove the methods listed in the resource domain handler interface from connection implementation class.
- Implement the resource domain handler interface in the service and notification classes which enables the classes to manage their own resource domain functionality.

Note:

The ResourceDomainHandler interface is not delivered as part of the ADK.

1. Create an interface for handling the resource domain.

In this example, a ResourceDomainHandler is created in the com.wm.MyAdapter package.

```
package com.wm.MyAdapter;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.*;
import com.wm.adk.error.*;
public interface ResourceDomainHandler
{
    /**
     * Implements resource domain lookups using the provided connection. Refer to
     * the method of the same name in com.wm.adk.connection.WmManagedConnection.
     **
     * @param connection
     * @param resourceDomainName
     * @param values
     * @return ResourceDomainValues[]
     * @throws AdapterException
     */
    public ResourceDomainValues[] adapterResourceDomainLookup(
        WmManagedConnection connection, String resourceDomainName,
        String[][] values) throws AdapterException;
    /**
     * Implements Adapter check values using the provided connection. Refer to
     * the method of the same name in com.wm.adk.connection.WmManagedConnection.
     **
     * @param connection
     * @param resourceDomainName
     * @param values
     * @param testValue
     * @return Boolean
     * @throws AdapterException
     */
    public Boolean adapterCheckValue( WmManagedConnection connection,
        String resourceDomainName,
        String[][] values, String testValue) throws AdapterException;
    /**
     * Implements resource domain registrations specific to a particular service
```

```

* or notification. Refer to the method of the same name in
* com.wm.adk.connection.WmManagedConnectionFactory.
**
@param connection
* @param access
* @throws AdapterException
*/
public void registerResourceDomain(WmManagedConnectionFactory connection,
    WmAdapterAccess access) throws AdapterException;
}

```

2. Update the connection factory implementation class to create a list of services and notifications that implement the interface, register them, and pass to the connections that the connection factory creates.

In this example a class `SimpleConnectionFactory` contains the following:

- Create a *supportedServiceTemplates* string array containing the class name of the adapter service templates.
- Create a *supportedNotificationTemplates* string array containing the class name of the adapter notification templates.
- Update `fillResourceAdapterMetadataInfo` method using *supportedServiceTemplates* to register the adapter service templates, and *supportedNotificationTemplates* to register the adapter notification templates.
- Update `createManagedConnectionObject` method to pass the list of services and notifications implementing the resource domain handler interface to the connection class.

```

package com.wm.MyAdapter.connections;
import com.wm.adk.connection.WmManagedConnectionFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.info.ResourceAdapterMetadataInfo;
import com.wm.adk.metadata.WmDescriptor;
import com.wm.adk.error.AdapterException;
import java.util.Locale;
import java.util.ArrayList;
import java.util.Arrays;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.MyAdapterConstants;
import com.wm.MyAdapter.services.MockDbUpdate;
import com.wm.MyAdapter.services.UIMockDbUpdate;
public class SimpleConnectionFactory extends WmManagedConnectionFactory implements
    MyAdapterConstants {
    private String hostName;
    private int port;

    private static final String[] supportedServiceTemplates = {
        MockDbUpdate.class.getName(),
        UIMockDbUpdate.class.getName()
    };

    /*
    private static final String[] supportedNotificationTemplates = {
        SimpleNotification.class.getName(),
        SessionLogListenerNotification.class.getName()
    };
    */
}

```

```

};
*/

public void setHostName(String hostNameValue){hostName = hostNameValue;}
public void setPort(int portValue){port = portValue;}
public SimpleConnectionFactory(){super();}
public WmManagedConnection createManagedConnectionObject(
    javax.security.auth.Subject subject,
    javax.resource.spi.ConnectionRequestInfo cxRequestInfo)
{
    ArrayList templateList = new ArrayList(Arrays.asList(supportedServiceTemplates));
    //Use the following to add notifications or other services
    //templateList.addAll(Arrays.asList(supportedNotificationTemplates));
    String[] listArg = new String[templateList.size()];
    templateList.toArray(listArg);
    return new SimpleConnection(hostName,port,listArg);
}
public void fillWmDescriptor(WmDescriptor d,Locale l) throws
    AdapterException
{
    d.createGroup(GROUP_SIMPLE_CONNECTION,
        new String[]{SIMPLE_SERVER_HOST_NAME, SIMPLE_SERVER_PORT_NUMBER});
    d.setValidValues(SIMPLE_SERVER_PORT_NUMBER, new String[] {"5555","1555","4000"});
    d.setDescriptions(
        MyAdapter.getInstance().getAdapterResourceBundleManager(),l);
}
public void fillResourceAdapterMetadataInfo(ResourceAdapterMetadataInfo info,
    Locale locale) {

    for (int i = 0; i < supportedServiceTemplates.length;i++)
    {
        info.addServiceTemplate(supportedServiceTemplates[i]);
    }
    //Use the following to add notifications or other services
    /*
    for (int i = 0; i < supportedNotificationTemplates.length;i++)
    {
        info.addNotificationTemplate(supportedNotificationTemplates[i]);
    }
    */
}
}
}

```

3. Update the connection implementation class to use the service name and forward the requests to the appropriate service or notification class.

In this example, the SimpleConnection class contains the following:

- Create the registerResourceDomain method and forward the requests to the appropriate service or notification class.
- Create the adapterResourceDomainLookup method, use the service name to identify the class name, and forward the requests to the appropriate service or notification class.
- Create the adapterCheckValue method, use the service name to identify the class name, and forward the requests to the appropriate service or notification class.

```
package com.wm.MyAdapter.connections;
```

```

import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.*;
import com.wm.adk.error.AdapterException;
import com.wm.MyAdapter.MyAdapter;
import com.wm.MyAdapter.services.MockDbUpdate;
import com.wm.MyAdapter.ResourceDomainHandler;
public class SimpleConnection extends WmManagedConnection {
    String hostName;
    int port;
    private String[] resourceHandlerList;
    public SimpleConnection(String hostNameValue, int portValue, String[]
resourceHandlerListValue)
    {
        super();
        hostName = hostNameValue;
        port = portValue;
        resourceHandlerList = resourceHandlerListValue;
        MyAdapter.getInstance().getLogger().logDebug(9999,
            "Simple Connection created with hostName = "
            + hostName + "and port = " + Integer.toString(port));
    }
    public void destroyConnection()
    {
        MyAdapter.getInstance().getLogger().logDebug(9999,"Simple Connection Destroyed");
    }
    public void registerResourceDomain(WmAdapterAccess access)
        throws AdapterException
    {
        try {
            Class serviceClass;
            ResourceDomainHandler serviceObject;
            for (int i = 0;i < resourceHandlerList.length;i++ ) {
                serviceClass = Class.forName(resourceHandlerList[i]);
                serviceObject = (ResourceDomainHandler)serviceClass.newInstance();
                serviceObject.registerResourceDomain(this,access);
            }
        }
        catch (Throwable t) {
            throw MyAdapter.getInstance().createAdapterException(9999,t);
        }
    }
    public ResourceDomainValues[] adapterResourceDomainLookup(String serviceName,
String resourceDomainName, String[][] values) throws AdapterException
    {
        Class serviceClass;
        ResourceDomainHandler serviceObject;
        try {
            serviceClass = Class.forName(serviceName);
            serviceObject = (ResourceDomainHandler)serviceClass.newInstance();
        }
        catch (Throwable t) {
            throw MyAdapter.getInstance().createAdapterException(9999,t);
        }
        return serviceObject.adapterResourceDomainLookup(this,resourceDomainName,values);
    }
    public Boolean adapterCheckValue(String serviceName,
String resourceDomainName,
String[][] values,
String testValue) throws AdapterException
    {

```



```

Class serviceClass;
ResourceDomainHandler serviceObject;
try {
    serviceClass = Class.forName(serviceName);
    serviceObject = (ResourceDomainHandler)serviceClass.newInstance();
}
catch (Throwable t) {
    throw MyAdapter.getInstance().createAdapterException(9999,t);
}
return serviceObject.adapterCheckValue(this, resourceDomainName, values, testValue);
}
}

```

4. Create two adapter service templates classes.

In this example, the two adapter service templates created are:

■ UIMockDbUpdate class:

```

package com.wm.MyAdapter.services;
import com.wm.adk.cci.interaction.WmAdapterService;
import com.wm.adk.cci.record.WmRecord;
import com.wm.adk.cci.record.WmRecordFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.WmTemplateDescriptor;
import com.wm.data.IData;
import com.wm.data.IDataCursor;
import com.wm.data.IDataFactory;
import com.wm.data.IDataUtil;
import java.util.Hashtable;
import java.util.Locale;
import javax.resource.ResourceException;
import com.wm.MyAdapter.MyAdapter;
//Alternate
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.*;
import com.wm.adk.error.AdapterException;
import com.wm.MyAdapter.ResourceDomainHandler;
//
public class UIMockDbUpdate extends WmAdapterService implements
ResourceDomainHandler{
    //Adapter Services variables
    private String[] UI_mockTableNames ={
"UI_CUSTOMERS","UI_ORDERS","UI_LINE_ITEMS"};
    private String[][] UI_mockColumnNames ={
    {"name","id", "ssn"},
    {"id","date","customer_id"},
    {"order_id","item_number","quantity","description"}
    };
    private String [][] UI_mockDataTypes = {
    {"java.lang.String","java.lang.Integer", "java.lang.String"},
    {"java.lang.Integer", "java.util.Date", "java.lang.Integer"},
    {"java.lang.Integer", "java.lang.Integer", "java.lang.Integer"},
    "java.lang.String"}
    };
    //MockDB Group
    public static final String UI_UPD_SETTINGS_GRP = "UI Mock Settings";
    public static final String UI_TABLE_NAME_PARM = "baseTableName";
    public static final String UI_COLUMN_NAMES_PARM = "baseColumnNames";

```

```

public static final String UI_COLUMN_TYPES_PARM = "baseColumnTypes";
public static final String UI_REPEATING_PARM = "baseRepeating";
public static final String UI_OVERRIDE_TYPES_PARM = "baseOverrideTypes";
private String baseTableName;
private String[] baseColumnNames;
private String[] baseColumnTypes;
private boolean baseRepeating;
private String[] baseOverrideTypes;
public void setBaseTableName(String val){ baseTableName = val;}
public void setBaseColumnNames(String[] val){ baseColumnNames = val;}
public void setBaseColumnTypes(String[] val){ baseColumnTypes = val;}
public void setBaseRepeating(boolean val){ baseRepeating = val;}
public void setBaseOverrideTypes(String[] val){baseOverrideTypes = val;}
public static final String UI_TABLES_RD = "baseTablesRD";
public static final String UI_COLUMN_NAMES_RD = "baseColumnNamesRD";
public static final String UI_COLUMN_TYPES_RD = "baseColumnTypesRD";
public static final String UI_OVERRIDE_TYPES_RD = "baseOverrideTypesRD";
public void fillWmTemplateDescriptor(WmTemplateDescriptor d,Locale l)
throws ResourceException
{
//UIMockDB Grouping and resource domain setup
d.createGroup(UI_UPD_SETTINGS_GRP, new String [] {
    UI_TABLE_NAME_PARM,
    UI_REPEATING_PARM,
    UI_COLUMN_NAMES_PARM,
    UI_COLUMN_TYPES_PARM,
    UI_OVERRIDE_TYPES_PARM}
);
d.createFieldMap(new String[] {
    UI_COLUMN_NAMES_PARM,
    UI_COLUMN_TYPES_PARM,
    UI_OVERRIDE_TYPES_PARM},
true);
d.createTuple(new String[]{UI_COLUMN_NAMES_PARM, UI_COLUMN_TYPES_PARM});

d.setResourceDomain(UI_TABLE_NAME_PARM, UI_TABLES_RD, null);
d.setResourceDomain(UI_COLUMN_NAMES_PARM, UI_COLUMN_NAMES_RD,
new String[]{UI_TABLE_NAME_PARM});
d.setResourceDomain(UI_COLUMN_TYPES_PARM, UI_COLUMN_TYPES_RD,
new String[]{UI_TABLE_NAME_PARM});
d.setResourceDomain(UI_OVERRIDE_TYPES_PARM,UI_OVERRIDE_TYPES_RD,null);
//Call to setDescription
d.setDescription(MyAdapter.getInstance().
getAdapterResourceBundleManager(),l);
}
public WmRecord execute(WmManagedConnection connection, WmRecord input)
throws ResourceException
{
Hashtable[] request = unpackRequest(input);
return packResonse(request);
}
private Hashtable[] unpackRequest(WmRecord request) throws ResourceException
{
Hashtable data[] = null;
IData mainIData = request.getIData();
IDataCursor mainCursor = mainIData.getCursor();
try
{
String tableNameValue = baseTableName;
String[] columnNamesValue = baseColumnNames;

```

```

if(mainCursor.first(tableNameValue))
{
IData[] recordIData;
if(baseRepeating)
{
recordIData = IDataUtil.getIDataArray (mainCursor,tableNameValue);
data = new Hashtable[recordIData.length];
}
else
{
recordIData = new IData[] {IDataUtil.getIData(mainCursor)};
data = new Hashtable[1];
}
for(int rec=0;rec<recordIData.length;rec++)
{
IDataCursor recordCursor = recordIData[rec].getCursor();
data[rec] = new Hashtable();
for(int c = 0; c < columnNamesValue.length;c++)
{
if(recordCursor.first(columnNamesValue[c]))
{
data[rec].put(tableNameValue + "." + columnNamesValue[c],
recordCursor.getValue());
}
}
recordCursor.destroy();
}
}
else
{
throw MyAdapter.getInstance().createAdapterException(9999,
new String[] {"No Request Data"});
}
}
catch (Throwable t)
{
throw MyAdapter.getInstance().createAdapterException(9999,
new String[] {"Error unpacking request data"},t);
}
finally
{
mainCursor.destroy();
}
return data;
}
private WmRecord packResonse(Hashtable[] response) throws ResourceException
{
WmRecord data = null;
try
{
IData[] recordIData = new IData[response.length];
String tableNameValue = baseTableName;
String[] columnNamesValue = baseColumnNames;
for(int rec = 0; rec < response.length; rec++)
{
recordIData[rec] = IDataFactory.create();
IDataCursor recordCursor = recordIData[rec].getCursor();
for(int col = 0; col < columnNamesValue.length;col++)
{
IDataUtil.put(recordCursor,columnNamesValue[col],

```

```

        response[rec].get(tableNameValue + "." +
            columnNamesValue[col]));
    }
    recordCursor.destroy();
}
IData mainIData = IDataFactory.create();
IDataCursor mainCursor = mainIData.getCursor();
if(baseRepeating)
{
    IDataUtil.put(mainCursor, tableNameValue, recordIData);
}
else
{
    IDataUtil.put(mainCursor, tableNameValue, recordIData[0]);
}
mainCursor.destroy();
data = WmRecordFactory.getFactory().createWmRecord("nameNotUsed");
data.setIData(mainIData);
}
catch (Throwable t)
{
    throw MyAdapter.getInstance().createAdapterException(9999,
        new String[] {"Error packing response data"},t);
}
return data;
}

//Alternate Methods
public void registerResourceDomain(WmManagedConnection connection,
WmAdapterAccess access)
    throws AdapterException
{
    //UIMockDB Group Registering Resource Domain
    ResourceDomainValues tableRdvs = new ResourceDomainValues(
        UIMockDbUpdate.UI_TABLES_RD, UI_mockTableNames);
    tableRdvs.setComplete(true);
    access.addResourceDomain(tableRdvs);
    access.addResourceDomainLookup(UIMockDbUpdate.UI_COLUMN_NAMES_RD,connection);
    access.addResourceDomainLookup(UIMockDbUpdate.UI_COLUMN_TYPERD_RD,connection);
    ResourceDomainValues rdvs = new ResourceDomainValues(
        UIMockDbUpdate.UI_OVERRIDE_TYPERD_RD, new String[] {""});
    rdvs.setComplete(false);
    rdvs.setCanValidate(true);
    access.addResourceDomain(rdvs);
    access.addCheckValue(UIMockDbUpdate.UI_OVERRIDE_TYPERD_RD,connection);
}

public ResourceDomainValues[] adapterResourceDomainLookup(
    WmManagedConnection connection,
    String resourceDomainName, String[][] values) throws AdapterException
{
    ResourceDomainValues[] results = null;

    if(resourceDomainName.equals(UIMockDbUpdate.UI_COLUMN_NAMES_RD) ||
        resourceDomainName.equals(UIMockDbUpdate.UI_COLUMN_TYPERD_RD))
    {
        String tableName = values[0][0];
        for(int x = 0; x < UI_mockTableNames.length;x++)
        {

```

```

        if(UI_mockTableNames[x].equals(tableName))
        {
            ResourceDomainValues columnsRdvs = new ResourceDomainValues(
                UIMockDbUpdate.UI_COLUMN_NAMES_RD,UI_mockColumnNames[x]);
            columnsRdvs.setComplete(true);
            ResourceDomainValues typesRdvs = new ResourceDomainValues(
                UIMockDbUpdate.UI_COLUMN_TYPES_RD, UI_mockDataTypes[x]);
            typesRdvs.setComplete(true);
            results = new ResourceDomainValues[] {columnsRdvs,typesRdvs};
            break;
        }
    }
}
return results;
}

public Boolean adapterCheckValue(WmManagedConnection connection,
    String resourceDomainName,
    String[][] values,
    String testValue) throws AdapterException
{
    Boolean result = new Boolean(false);
    if(resourceDomainName.equals(UIMockDbUpdate.UI_OVERRIDE_TYPES_RD))
    {
        try
        {
            Object o = Class.forName(testValue).getConstructor(
                new Class[] {String.class}).newInstance(new Object[]{"0"});
            result = new Boolean(true);
        }
        catch (Throwable t){}
    }
    return result;
}
}

```

■ **MockDbUpdate class:**

```

package com.wm.MyAdapter.services;
import com.wm.adk.cci.interaction.WmAdapterService;
import com.wm.adk.cci.record.WmRecord;
import com.wm.adk.cci.record.WmRecordFactory;
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.WmTemplateDescriptor;
import com.wm.data.IData;
import com.wm.data.IDataCursor;
import com.wm.data.IDataFactory;
import com.wm.data.IDataUtil;
import java.util.Hashtable;
import java.util.Locale;
import javax.resource.ResourceException;
import com.wm.MyAdapter.MyAdapter;
//Alternate
import com.wm.adk.connection.WmManagedConnection;
import com.wm.adk.metadata.*;
import com.wm.adk.error.AdapterException;
import com.wm.MyAdapter.ResourceDomainHandler;
//
public class MockDbUpdate extends WmAdapterService implements
ResourceDomainHandler {

```

```

//Adapter Services variables
private String[] mockTableNames ={ "CUSTOMERS","ORDERS","LINE_ITEMS"};
private String[][] mockColumnNames ={
    {"name","id", "ssn"},
    {"id","date","customer_id"},
    {"order_id","item_number","quantity","description"}
};
private String [][] mockDataTypes = {
    {"java.lang.String","java.lang.Integer", "java.lang.String"},
    {"java.lang.Integer", "java.util.Date", "java.lang.Integer"},
    {"java.lang.Integer", "java.lang.Integer", "java.lang.Integer"},
    {"java.lang.String"}
};
//MockDB Group
public static final String UPD_SETTINGS_GRP = "Mock Settings";
public static final String TABLE_NAME_PARM = "tableName";
public static final String COLUMN_NAMES_PARM = "columnNames";
public static final String COLUMN_TYPES_PARM = "columnTypes";
public static final String REPEATING_PARM = "repeating";
public static final String OVERRIDE_TYPES_PARM = "overrideTypes";
private String tableName;
private String[] columnNames;
private String[] columnTypes;
private boolean repeating;
private String[] overrideTypes;
public void setTableName(String val){ tableName = val;}
public void setColumnNames(String[] val){ columnNames = val;}
public void setColumnTypes(String[] val){ columnTypes = val;}
public void setRepeating(boolean val){ repeating = val;}
public void setOverrideTypes(String[] val){overrideTypes = val;}
public static final String TABLES_RD = "tablesRD";
public static final String COLUMN_NAMES_RD = "columnNamesRD";
public static final String COLUMN_TYPES_RD = "columnTypesRD";
public static final String OVERRIDE_TYPES_RD = "overrideTypesRD";
//MockDB Signature Group
public static final String SIG_SETTINGS_GRP = "Signature";
public static final String FIELD_NAMES_PARM = "fieldNames";
public static final String FIELD_TYPES_PARM = "fieldTypes";
public static final String SIG_IN_PARM = "sigIn";
public static final String SIG_OUT_PARM = "sigOut";
private String[] fieldNames;
private String[] fieldTypes;
public void setFieldNames(String[] val){ fieldNames = val;}
public void setFieldTypes(String[] val){ fieldTypes = val;}
public void setSigIn(String[] val){}
public void setSigOut(String[] val){}
public static final String FIELD_NAMES_RD = "fieldNamesRD";
public static final String FIELD_TYPES_RD = "fieldTypesRD";
public void fillWmTemplateDescriptor(WmTemplateDescriptor d,Locale l)
    throws ResourceException
{
    //MockDB Grouping and resource domain setup
    d.createGroup(UPD_SETTINGS_GRP, new String [] {
        TABLE_NAME_PARM,
        REPEATING_PARM,
        COLUMN_NAMES_PARM,
        COLUMN_TYPES_PARM,
        OVERRIDE_TYPES_PARM}
    );
    d.createFieldMap(new String[] {

```

```

        COLUMN_NAMES_PARM,
        COLUMN_TYPES_PARM,
        OVERRIDE_TYPES_PARM},
        true);
d.createTuple(new String[]{COLUMN_NAMES_PARM,COLUMN_TYPES_PARM});

d.setResourceDomain(TABLE_NAME_PARM, TABLES_RD, null);
d.setResourceDomain(COLUMN_NAMES_PARM, COLUMN_NAMES_RD,
    new String[]{TABLE_NAME_PARM});
d.setResourceDomain(COLUMN_TYPES_PARM, COLUMN_TYPES_RD,
    new String[]{TABLE_NAME_PARM});
d.setResourceDomain(OVERRIDE_TYPES_PARM, OVERRIDE_TYPES_RD, null);
//MockDB Signature Grouping and resource domain setup
d.createGroup(SIG_SETTINGS_GRP, new String [] {
    FIELD_NAMES_PARM,
    FIELD_TYPES_PARM,
    SIG_IN_PARM,
    SIG_OUT_PARM}
);
d.createFieldMap(new String [] {
    FIELD_NAMES_PARM,
    FIELD_TYPES_PARM,
    SIG_IN_PARM,
    SIG_OUT_PARM},
    false);
d.createTuple(new String[]{FIELD_NAMES_PARM, FIELD_TYPES_PARM});

String [] fieldTupleDependencies = {TABLE_NAME_PARM,
    REPEATING_PARM,
    COLUMN_NAMES_PARM,
    COLUMN_TYPES_PARM,
    OVERRIDE_TYPES_PARM};
d.setResourceDomain(FIELD_NAMES_PARM, FIELD_NAMES_RD, fieldTupleDependencies);
d.setResourceDomain(FIELD_TYPES_PARM, FIELD_TYPES_RD, fieldTupleDependencies);
d.setResourceDomain(SIG_IN_PARM, WmTemplateDescriptor.INPUT_FIELD_NAMES,
    new String[] {FIELD_NAMES_PARM, FIELD_TYPES_PARM});
d.setResourceDomain(SIG_OUT_PARM, WmTemplateDescriptor.OUTPUT_FIELD_NAMES,
    new String[] {FIELD_NAMES_PARM, FIELD_TYPES_PARM});
//Call to setDescription
d.setDescriptions(MyAdapter.getInstance().
    getAdapterResourceBundleManager(), l);
}
public WmRecord execute(WmManagedConnection connection, WmRecord input)
    throws ResourceException
{
    Hashtable[] request = unpackRequest(input);
    return packResonse(request);
}
private Hashtable[] unpackRequest(WmRecord request) throws ResourceException
{
    Hashtable data[] = null;
    IData mainIData = request.getIData();
    IDataCursor mainCursor = mainIData.getCursor();
    try
    {
        {
            String tableNameValue = tableName;
            String[] columnNamesValue = columnNames;
            if(mainCursor.first(tableNameValue))
            {
                IData[] recordIData;

```

```

    if(repeating)
    {
        recordIData = IDataUtil.getIDataArray (mainCursor,tableNameValue);
        data = new Hashtable[recordIData.length];
    }
    else
    {
        recordIData = new IData[] {IDataUtil.getIData(mainCursor)};
        data = new Hashtable[1];
    }
    for(int rec=0;rec<recordIData.length;rec++)
    {
        IDataCursor recordCursor = recordIData[rec].getCursor();
        data[rec] = new Hashtable();
        for(int c = 0; c < columnNamesValue.length;c++)
        {
            if(recordCursor.first(columnNamesValue[c]))
            {
                data[rec].put(tableNameValue + "." + columnNamesValue[c],
                    recordCursor.getValue());
            }
        }
        recordCursor.destroy();
    }
}
else
{
    throw MyAdapter.getInstance().createAdapterException(9999,
        new String[] {"No Request Data"});
}
}
catch (Throwable t)
{
    throw MyAdapter.getInstance().createAdapterException(9999,
        new String[] {"Error unpacking request data"},t);
}
finally
{
    mainCursor.destroy();
}
return data;
}
private WmRecord packResonse(Hashtable[] response) throws ResourceException
{
    WmRecord data = null;
    try
    {
        IData[] recordIData = new IData[response.length];
        String tableNameValue = tableName;
        String[] columnNamesValue = columnNames;
        for(int rec = 0; rec < response.length; rec++)
        {
            recordIData[rec] = IDataFactory.create();
            IDataCursor recordCursor = recordIData[rec].getCursor();
            for(int col = 0; col < columnNamesValue.length;col++)
            {
                IDataUtil.put(recordCursor,columnNamesValue[col],
                    response[rec].get(tableNameValue + "." +
                    columnNamesValue[col]));
            }
        }
    }
}

```



```

        recordCursor.destroy();
    }
    IData mainIData = IDataFactory.create();
    IDataCursor mainCursor = mainIData.getCursor();
    if(repeating)
    {
        IDataUtil.put(mainCursor,tableNameValue,recordIData);
    }
    else
    {
        IDataUtil.put(mainCursor,tableNameValue,recordIData[0]);
    }
    mainCursor.destroy();
    data = WmRecordFactory.getFactory().createWmRecord("nameNotUsed");
    data.setIData(mainIData);
}
catch (Throwable t)
{
    throw MyAdapter.getInstance().createAdapterException(9999,
        new String[] {"Error packing response data"},t);
}
return data;
}

public void registerResourceDomain(WmManagedConnection connection,
WmAdapterAccess access)
throws AdapterException
{
    //MockDB Group Registering Resource Domain
    ResourceDomainValues tableRdvs = new ResourceDomainValues(
        MockDbUpdate.TABLES_RD,mockTableNames);
    tableRdvs.setComplete(true);
    access.addResourceDomain(tableRdvs);
    access.addResourceDomainLookup(MockDbUpdate.COLUMN_NAMES_RD,connection);
    access.addResourceDomainLookup(MockDbUpdate.COLUMN_TYPES_RD,connection);
    ResourceDomainValues rdvs = new ResourceDomainValues(
        MockDbUpdate.OVERRIDE_TYPES_RD, new String[] {""});
    rdvs.setComplete(false);
    rdvs.setCanValidate(true);
    access.addResourceDomain(rdvs);
    access.addCheckValue(MockDbUpdate.OVERRIDE_TYPES_RD,connection);

    //MockDB Signature Group Registering Resource Domain
    access.addResourceDomainLookup(MockDbUpdate.FIELD_NAMES_RD,connection);
    access.addResourceDomainLookup(MockDbUpdate.FIELD_TYPES_RD,connection);
}

public ResourceDomainValues[] adapterResourceDomainLookup(
    WmManagedConnection connection,
    String resourceDomainName, String[][] values) throws AdapterException
{
    ResourceDomainValues[] results = null;

    //MockDB Group Lookup
    if(resourceDomainName.equals(MockDbUpdate.COLUMN_NAMES_RD) ||
        resourceDomainName.equals(MockDbUpdate.COLUMN_TYPES_RD))
    {
        String tableName = values[0][0];
        for(int x = 0; x < mockTableNames.length;x++)
        {

```

```

    if(mockTableNames[x].equals(tableName))
    {
        ResourceDomainValues columnsRdvs = new ResourceDomainValues(
            MockDbUpdate.COLUMN_NAMES_RD, mockColumnNames[x]);
        columnsRdvs.setComplete(true);
        ResourceDomainValues typesRdvs = new ResourceDomainValues(
            MockDbUpdate.COLUMN_TYPES_RD, mockDataTypes[x]);
        typesRdvs.setComplete(true);
        results = new ResourceDomainValues[] {columnsRdvs, typesRdvs};
        break;
    }
}
//MockDB Signature Group Lookup
else if (resourceDomainName.equals(MockDbUpdate.FIELD_NAMES_RD) ||
    resourceDomainName.equals(MockDbUpdate.FIELD_TYPES_RD))
{
    String tableName = values[0][0];
    boolean repeating = Boolean.valueOf(values[1][0]).booleanValue();
    String[] columnNames = values[2];
    String[] columnTypes = values[3];
    String[] overrideTypes = values[4];
    String[] fieldNames = new String[columnNames.length];
    String[] fieldTypes = new String[columnTypes.length];
    String optBrackets;
    if(repeating)
        optBrackets = "[";
    else
        optBrackets = "";
    for (int i = 0; i < fieldNames.length; i++)
    {
        fieldNames[i] = tableName + optBrackets + "." + columnNames[i];
        fieldTypes[i] = columnTypes[i] + optBrackets;
        if(overrideTypes.length > i)
        {
            if (!overrideTypes[i].equals(""))
            {
                fieldTypes[i] = overrideTypes[i] + optBrackets;
            }
        }
    }
    results = new ResourceDomainValues[]{
        new ResourceDomainValues(MockDbUpdate.FIELD_NAMES_RD, fieldNames),
        new ResourceDomainValues(MockDbUpdate.FIELD_TYPES_RD, fieldTypes)};
}
return results;
}

public Boolean adapterCheckValue(WmManagedConnection connection,
    String resourceDomainName,
    String[][] values,
    String testValue) throws AdapterException
{
    Boolean result = new Boolean(false);
    if(resourceDomainName.equals(MockDbUpdate.OVERRIDE_TYPES_RD))
    {
        try
        {
            Object o = Class.forName(testValue).getConstructor(
                new Class[] {String.class}).newInstance(new Object[]{"0"});

```

```

    result = new Boolean(true);
  }
  catch (Throwable t){}
}
return result;
}
}

```

- Corresponding MyAdapterResource class:

```

package com.wm.MyAdapter;
import java.util.ListResourceBundle;
import com.wm.adk.ADKGLOBAL;
import com.wm.MyAdapter.connections.SimpleConnectionFactory;
import com.wm.MyAdapter.services.MockDbUpdate;
import com.wm.MyAdapter.services.UIMockDbUpdate;
public class MyAdapterResource extends ListResourceBundle implements
MyAdapterConstants{
    static final String IS_PKG_NAME = "/MyAdapter/";
    static final Object[][] _contents = {
        // adapter type display name.
        {ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME, "MyAdapter"}
        // adapter type descriptions.
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Adapter for MyAdapter Server (a Sample System)"}
        // adapter type vendor.
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_VENDORNAME, "Software AG"}
        //Copyright URL Page
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_THIRDPARTYCOPYRIGHTURL,
IS_PKG_NAME + "copyright.html"}
        //Copyright Encoding
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_COPYRIGHTENCODING, "UTF-8"}
        //About URL Page
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_ABOUT, IS_PKG_NAME + "About.html"}
        //Release Notes URL Page
        ,{ADAPTER_NAME + ADKGLOBAL.RESOURCEBUNDLEKEY_RELEASENOTEURL, IS_PKG_NAME +
"ReleaseNotes.html"}
        //SimpleConnection
        ,{SimpleConnectionFactory.class.getName() +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Simple Connection"}
        ,{SimpleConnectionFactory.class.getName() +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "Simple framework for demonstration purposes"}
        ,{SimpleConnectionFactory.SIMPLE_SERVER_HOST_NAME +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Host Name"}
        ,{SimpleConnectionFactory.SIMPLE_SERVER_PORT_NUMBER +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "Port"}

        //UIMockDB Group Resource Domain Values
        ,{UIMockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
        "UI Mock Update Service"}
        ,{UIMockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
        "UI Simulates a database update service"}
        ,{UIMockDbUpdate.UI_UPD_SETTINGS_GRP + ADKGLOBAL.RESOURCEBUNDLEKEY_GROUP,
        UIMockDbUpdate.UI_UPD_SETTINGS_GRP}
        ,{UIMockDbUpdate.UI_TABLE_NAME_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,

```

```

    "UI Table Name"}
    ,{UIMockDbUpdate.UI_TABLE_NAME_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,

    "UI Select Table Name"}
    ,{UIMockDbUpdate.UI_COLUMN_NAMES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "UI Column Names"}
    ,{UIMockDbUpdate.UI_COLUMN_NAMES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "UI Name of column updated by this service"}
    ,{UIMockDbUpdate.UI_COLUMN_TYPES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "UI Column Types"}
    ,{UIMockDbUpdate.UI_COLUMN_TYPES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "UI Default data type for column"}
    ,{UIMockDbUpdate.UI_OVERRIDE_TYPES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "UI Override Data Types"}
    ,{UIMockDbUpdate.UI_OVERRIDE_TYPES_PARM +
ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "UI Type to override column default"}
    ,{UIMockDbUpdate.UI_REPEATING_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,

    "UI Update Multiple Rows?"}
    ,{UIMockDbUpdate.UI_REPEATING_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "UI Select if input will include multiple rows to update"}
//MockDB Group Resource Domain Values
    ,{MockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Mock Update Service"}
    ,{MockDbUpdate.class.getName() + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Simulates a database update service"}
    ,{MockDbUpdate.UPD_SETTINGS_GRP + ADKGLOBAL.RESOURCEBUNDLEKEY_GROUP,
    MockDbUpdate.UPD_SETTINGS_GRP}
    ,{MockDbUpdate.TABLE_NAME_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Table Name"}
    ,{MockDbUpdate.TABLE_NAME_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Select Table Name"}
    ,{MockDbUpdate.COLUMN_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Column Names"}
    ,{MockDbUpdate.COLUMN_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Name of column updated by this service"}
    ,{MockDbUpdate.COLUMN_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Column Types"}
    ,{MockDbUpdate.COLUMN_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Default data type for column"}
    ,{MockDbUpdate.OVERRIDE_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,

    "Override Data Types"}
    ,{MockDbUpdate.OVERRIDE_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,

    "Type to override column default"}
    ,{MockDbUpdate.REPEATING_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Update Multiple Rows?"}
    ,{MockDbUpdate.REPEATING_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Select if input will include multiple rows to update"}
//MockDB Signature Group Resource Domain Values
    ,{MockDbUpdate.SIG_SETTINGS_GRP + ADKGLOBAL.RESOURCEBUNDLEKEY_GROUP,
    MockDbUpdate.SIG_SETTINGS_GRP}
    ,{MockDbUpdate.FIELD_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,

```

```

    "Field Names"}
    ,{MockDbUpdate.FIELD_NAMES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Name of Field"}
    ,{MockDbUpdate.FIELD_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Field Type"}
    ,{MockDbUpdate.FIELD_TYPES_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Type of Field"}
    ,{MockDbUpdate.SIG_IN_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Input Signature"}
    ,{MockDbUpdate.SIG_IN_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Input Signature"}
    ,{MockDbUpdate.SIG_OUT_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DISPLAYNAME,
    "Output Signature"}
    ,{MockDbUpdate.SIG_OUT_PARM + ADKGLOBAL.RESOURCEBUNDLEKEY_DESCRIPTION,
    "Output Signature"}
};

protected Object[][] getContents() {
    // TODO Auto-generated method stub
    return _contents;
}
}

```

Using Resource Bundles with Resource Domain Values

A resource bundle contains all display strings and messages used by the adapter at runtime and at design time. A resource bundle is:

- Specific to a particular locale.
- Enables you to internationalize an adapter quickly, without having to change the code in the adapter.

For more information, see [“Creating Resource Bundles Class With Example ” on page 38](#).

Adapters can explicitly use a resource bundle to localize other data known at development time. For example, you can localize data such as resource domain values. For more information about resource domain values, see [“Populating Resource Domains with Values” on page 88](#). In those cases, the strategy for key composition is left to your discretion.

Important:

When a localized value needs to be understood in the other parts of the code, a translation is performed. For example, a list of known record status values as shown: `(if this.getStatus() != "active")`

For example, suppose an adapter service includes a metadata parameter called `status`, which can be set to a value of `active` or `inactive`. These values must be localized so that `active` or `inactive` appear in the language of the current user. If the runtime locale is different from the locale in design-time client, then the adapter needs to know the language that is used when the value of `status` was set. You must use the following:

```
if (this.getStatus() != inLanguageOfDesignTimeClient("active"))
```




instead of the following:

```
if (this.getStatus() != "active")
```

The localization becomes more complicated if the adapter service node is managed (edited) in more than one locale. If a resource domain lookup depends on a value that is localized, and that value was set by a client in a different locale than the current client, then use the following:

```
if (this.getStatus() != inLanguageUsedWhenValueWasSet("active"))
```

To support this functionality, the ADK inserts a special *designTimeLocale* metadata property, in each adapter service node and notification node.

- Value of *designTimeLocale* is set when the node is created.
- Adapter users update the value in the node when they click the  icon on the Designer toolbar which reloads values from the adapter. The  icon appears when users view an adapter service or notification node.
- An adapter that uses the *designTimeLocale* property can then present resource domain values in the appropriate client locale when the node is created, and understand the value at runtime regardless of what locale Integration Server may use.
- For example, if a Japanese client wants to edit a node that was configured by an English client, then
 - The initial presentation in the Adapter Service Editor or Adapter Notification Editor reflects the English values currently stored in the node.
 - When the Japanese client clicks the  icon, Integration Server changes the *designTimeLocale* to Japanese, and performs all resource domain lookups using the new locale.
 - From the adapter user's perspective, all localized values change from English to Japanese.
 - This example assumes that the adapter includes both an English and a Japanese resource bundle.

To support localized resource domain values, perform the following:

- Include the *ADKGLOBAL.DESIGN_LOCALE_PROPERTY* property name in the resource domain dependency list of any parameter that uses a resource domain with localized resource domain values.
- Include *ADKGLOBAL.DESIGN_LOCALE_PROPERTY* property name for all the parameters where the lookup depends on a parameter containing localized values.

For example:

```
d.setResourceDomain( "status", "statusLookup", new
    String[]{ADKGLOBAL.DESIGN_LOCALE_PROPERTY} );
d.setResourceDomain( "statusDescription","statusDescriptionLookup", new
    String[]
    { "status",ADKGLOBAL.DESIGN_LOCALE_PROPERTY } );
```

You must present the *designTimeLocale* as a string (derived from `Locale.toString`) so that the value is available as a resource domain value. To convert this string to a `Locale` object, use `AdapterUtil.parseLocaleString` method.

Within the implementation of the *statusLookup*, the adapter uses the *designTimeLocale* value provided to create a list of localized string values:

```
if (resourceDomainName.equals("statusLookup"))
{
    AdapterResourceBundleManager ar =
        MyAdapter.getInstance().getAdapterResourceBundleManager();
    Locale lookupLocale = AdapterUtil.parseLocaleString(values[0][0]);
    for (int i =0; i< statusNames.length;i++)
    {
        try {
            displayNames[i] =
                ar.getStringResource(statusNames[i], lookupLocale);
        }
        catch(Throwable t)
        {
            displayNames[i] = statusNames[i];
        }
    }
    ResourceDomainValues rdv = new
        ResourceDomainValues(resourceDomainName,displayNames);
    rdv.setComplete(true); // allow user edit
    return new ResourceDomainValues[] {rdv};
}
```

At runtime, the adapter uses the `getDesignTimeLocale` method to retrieve the locale, and to interpret the value of localized parameters. For example:

```
Locale lookupLocale =
    AdapterUtil.parseLocaleString(this.getDesignTimeLocale());
AdapterResourceBundleManager ar =
    MyAdapter.getInstance().getAdapterResourceBundleManager();
if (this.getStatus.equals(ar.getStringResource("active",lookupLocale)))
{
    ...
}
```

For more information, see the Javadoc for the following:

- `com.wm.adk.ADKGLOBALS.DESIGN_TIME_LOCALE_PROPERTY`
- `com.wm.adk.cci.interaction.WmAdapterService.getDesignTimeLocale`
- `com.wm.adk.notification.WmNotification.getDesignTimeLocale`
- `com.wm.adk.util.AdapterUtil.parseLocaleString`

B Integration Server Transaction Support

■ Overview	330
■ Simple Transactions	331
■ More Complex Transactions	331
■ Implicit Transaction Usage Cases	332
■ Explicit Transaction Usage Cases	333
■ Built-In Services For Explicit Transactions	337
■ Transaction Error Situations	340
■ Specifying Transaction Support in Connections	341

Overview

This section describes how Integration Server supports transactions. Integration Server considers a transaction to be one or more interactions with one or more resources that are treated as a single logical unit of work. 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 types of transactions:

- **Local Transaction.** Transaction to a resource's local transaction mechanism.
- **XAResource Transaction.** Transaction to a resource's XAResource transaction mechanism.

Integration Server can manage both types of transactions, without requiring the adapter users' input.

- Integration Server uses a container-managed (implicit) transaction management approach based on the JCA standard.
- Integration Server performs additional connection management. This is because the adapter services use connections to create transactions.

For more information about implicit transactions, see [“Implicit Transaction Usage Cases” on page 332](#). However, there are cases where adapter users need to explicitly control the transactional units of work. For more information about explicit transactions, see [“Explicit Transaction Usage Cases” on page 333](#).

Integration Server relies on a built-in transaction manager to support transactions. The transaction manager is responsible for the following:

- Beginning and ending transactions.
- Maintaining a transaction context.
- Enlisting newly connected resources into existing transactions.
- Ensuring that local and XAResource transactions are not combined in invalid ways.
- Manages operations performed by a transacted JMS trigger, or a built-in JMS service that uses a transacted JMS connection alias.

Important:

You cannot create steps and trace a flow that contains a transacted adapter service.

Note:

If you interact with a resource that does not support transactions, Integration Server does not create a transaction for it.

For more information about specifying the type(s) of transactions to support in your adapter, see [“Specifying Transaction Support in Connections” on page 341](#).

Simple Transactions

The simplest Integration Server transaction scenario is a Flow Service (or a Java Service) that invokes one adapter service that interacts with one resource. For example, the transaction might perform a database insert.

Integration Server executes the transaction without requiring adapter users to perform any transaction management, as follows:

1. Integration Server invokes the request as follows:
 - a. Adapter service obtains a connection from the connection pool.
 - b. Adapter service creates a transaction.
 - c. Adapter service enlists the connection in the current transaction.
 - d. Adapter service performs the database insert. If the insert fails, a `ServiceException` is thrown.
2. Integration Server informs the transaction manager that the service request is completed as follows:
 - If the service request succeeds, the transaction manager commits the current transaction. If the commit fails, the transaction manager throws an exception that results in the failure of the service request, and a `ServiceException` is returned to the adapter user.
 - If the service request fails, the transaction manager rolls back the current transaction.

Note:

The commit or rollback occurs after the service is complete, but before any response is sent to the client that invoked the service. This is because if the transaction commit fails, the service itself must fail. So, the transaction notification essentially becomes the last step of the service as opposed to occurring after the service is already complete.

More Complex Transactions

Suppose that a Flow Service or a Java Service invokes multiple adapter services. These adapter services might interact with a single resource (such as two services that perform two inserts into a single database) or with multiple resources (such as services that synchronize a database and an ERP system).

You can have one or more connections to a single resource. If multiple connections are used, then Integration Server enlists each connection in the transaction.

When the service request is complete, Integration Server notifies the transaction manager, which closes all enlisted connections and transactions.

- If the service request is successful, then Integration Server commits all transactions automatically using a two-phase commit if multiple resources are used.
- If the service request returns an error, then Integration Server rolls back the transaction and returns an error; that causes the service to return an error.

As mentioned previously, there are cases where adapter users must explicitly control the transactional units of work. For more information, see [“Explicit Transaction Usage Cases” on page 333](#).

Note:

If a transaction accesses multiple resources, and more than one of the resources supports local transactions only, then the integrity of the transaction cannot be guaranteed. For example, if the first resource successfully commits, and the second resource fails to commit, the first resource interaction cannot be rolled back; it has already been committed. Integration Server detects this case when connecting to more than one resource that do not support two-phase commits and throws a runtime exception which results in the failure of the service execution.

Implicit Transaction Usage Cases

Integration Server handles implicit transactions. For a flow to be managed implicitly, it can contain one of the following:

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

If a flow contains multiple local transactions, the adapter user must explicitly control the transactional units of work. For more information, see [“Explicit Transaction Usage Cases” on page 333](#).

Following are examples of implicit transactions.

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.
- Integration Server 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.

Two Local Transactions

The following flow is *invalid* 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, this flow contains an 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. Integration Server performs the following:
 - Establishes a second connection to *Database2*.
 - Enlists the new connection in the XAResource transaction.
 - Performs the insert to *tableA*.
- When the request is complete
 - Integration Server closes both connections.
 - Transaction Manager performs a local commit for the non-XAResource and then a two-phase commit for the XAResource enlisted in the transaction.

Explicit Transaction Usage Cases

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

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

Integration Server provides the following built-in services to support multiple local transactions and multiple XAResource transactions:

- `pub.art.transaction.startTransaction`
- `pub.art.transaction.commitTransaction`
- `pub.art.transaction.rollbackTransaction`
- `pub.art.transaction.setTransactionTimeout`

For more information, see [“Built-In Services For Explicit Transactions” on page 337](#).

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; 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.

- 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:

- When *interactWithResourceA* is invoked, Integration Server starts an *XAResource transaction 1* and enlists *ResourceA*.
- When *interactWithXAResourceB* is invoked, Integration Server enlists *XAResourceB* in *transaction 1*.
- Transaction 2* is executed as follows:
 - When *startTransaction(2)* is invoked, Integration Server starts a new, nested transaction.
 - When *interactWithLocalResourceA* is invoked, *LocalResourceA* is enlisted in *transaction 2*.
 - When *interactWithXAResourceC* is invoked, *XAResourceC* is enlisted in *transaction 2*.
 - When *interactWithLocalResourceA* is invoked, *LocalResourceA* is enlisted in *transaction 2*.
 - 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.
- When the flow ends, Integration Server closes the connection for *transaction 1* and commits its work to *XAResourceA* and *XAResourceB*.

Built-In Services For Explicit Transactions

Use the built-in services described in this section to manage explicit transactions for your Adapter Development Kit services.

Explicit transactions are transactions that is manually controlled within flow services using built-in services. Implicit transactions are automatically handled by the Integration Server's transaction manager. 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.

Any flow service steps found between a `pub.art.transaction:startTransaction` service and either a `pub.art.transaction:commitTransaction` service or a `pub.art.transaction:rollbackTransaction` service are part of an explicit transaction rather than the implicit transaction.

Within both implicit and explicit transactions, you cannot have multiple connections with a transaction type of `LOCAL_TRANSACTION` because you will not be able to rollback the first `LOCAL_TRANSACTION` after it is committed. Use the built-in services to define explicit transactions to prevent from inadvertently committing transactions if you need to rollback the transaction.

The table below briefly describes the public services in Integration Servers' WmART package. The sections that follow describe each service in detail.

Service	Function
<code>pub.art.transaction:startTransaction</code>	Starts an explicit transaction.
<code>pub.art.transaction:commitTransaction</code>	Commits an explicit transaction.
<code>pub.art.transaction:rollbackTransaction</code>	Rolls back an explicit transaction.
<code>pub.art.transaction:setTransactionTimeout</code>	Enables you to manually set a transaction timeout interval for implicit and explicit transactions.

pub.art.transaction:startTransaction

Starts an explicit transaction. The service must be used in conjunction with either a `pub.art.transaction:commitTransaction` service or `pub.art.transaction:rollbackTransaction` service. If a corresponding `pub.art.transaction:commitTransaction` service or `pub.art.transaction:rollbackTransaction` service is not provided, then the flow service receives a runtime error.

Input Parameters

startTransactionInput **Document.** Document that contains the variable *transactionName*.

transactionName **String.** Used to associate a name with an explicit transaction. The *transactionName* must correspond to the *transactionName* in any `pub.art.transaction:rollbackTransaction` or `pub.art.transaction:commitTransaction` services associated with the explicit transaction.

Output Parameters

startTransactionOutput **Document.** Document that contains the variable *transactionName*.

transactionName **String.** Used to associate a name with an explicit transaction. The *transactionName* must correspond to the *transactionName* in any `pub.art.transaction:rollbackTransaction` or `pub.art.transaction:commitTransaction` services associated with the explicit transaction.

pub.art.transaction:commitTransaction

Commits an explicit transaction. The service must be used in conjunction with the `pub.art.transaction:startTransaction` service. If a corresponding `pub.art.transaction:startTransaction` service is not provided, then the flow service receives a runtime error.

Input Parameters

commitTransactionInput **Document.** Document that contains the variable *transactionName*.

transactionName **String.** Used to associate a name with an explicit transaction. The *transactionName* must correspond to the *transactionName* in any `pub.art.transaction:startTransaction` or `pub.art.transaction:rollbackTransaction` services associated with the explicit transaction.

pub.art.transaction:rollbackTransaction

Rolls back an explicit transaction. The service must be used in conjunction with a `pub.art.transaction:startTransaction` service. If a corresponding `pub.art.transaction:startTransaction` service is not provided, then the flow service receives a runtime error.

Input Parameters

rollbackTransactionInput **Document.** Document that contains the variable *transactionName*.

transactionName **String.** Used to associate a name with an explicit transaction. The *transactionName* must correspond to the *transactionName* in any `WmART.pub.art.transaction:startTransaction` or `WmART.pub.art.transaction:commitTransaction` services associated with the explicit transaction.

pub.art.transaction:setTransactionTimeout

Enables you to manually set a transaction timeout interval for implicit and explicit transactions. The service overrides the Integration Server's transaction timeout interval. For more information

about changing the Integration Server default transaction timeout, see [“Changing the Integration Servers' Transaction Timeout Interval” on page 340](#).

- You must call this service within a flow before the start of any implicit or explicit transactions. Implicit transactions start when you call an adapter service in a flow. Explicit transactions start when you call the `pub.art.transaction:startTransaction` service.
- If the execution of a transaction takes longer than the transaction timeout interval, all current executions associated with the flow are cancelled and rolled back if necessary.
- The service only overrides the transaction timeout interval for the flow service in which you call it.

Input Parameters

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

Changing the Integration Servers' Transaction Timeout Interval

Configures the maximum number of seconds that a transaction can remain open and still be considered valid. This transaction timeout parameter does not halt the execution of a flow. Default value is `NO_TIMEOUT`.

For example, if a current transaction has a timeout value of 60 seconds and a flow takes 120 seconds to complete, the transaction manager rolls back all registered operations regardless of the execution status.

You can configure the following property on the **Extended Settings** screen (**Settings > Extended**) in Integration Server Administrator:

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

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

Restart Integration Server after configuring the property.

Transaction Error Situations

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

- A transaction includes a resource that only supports local transactions.

If a transaction accesses multiple resources, and more than one of the resources support local transactions only, then the integrity of the transaction cannot be guaranteed. For example, if the first resource successfully commits, and the second resource fails to commit, the first resource interaction cannot be rolled back; it has already been committed. To prevent this, Integration Server detects this case when connecting to more than one resource that does not

support two-phase commits, and throws a runtime exception resulting in the failure of the service execution.

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

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

- A parent transaction is closed before its nested transaction.
- After a service request has invoked all its services, but before returning results to the caller, the service may commit its work. This commit could fail if the resource is unavailable or rejects the commit and causes the entire request to fail, and the transaction is rolled back.

Specifying Transaction Support in Connections

Return the appropriate transaction support level in your `WmManagedConnectionFactory.queryTransactionSupportLevel` implementation to support transactions in your adapter.

- For local transaction support, override the `WmManagedConnection.getLocalTransaction` method to return a `javax.resource.spi.LocalTransaction` object that is capable of interfacing with the transactional capabilities of your resource.
- For XA transaction support, override the `WmManagedConnection.getXAResource` method to return a `javax.transaction.xa.XAResource` object that is capable of interfacing with the XA transactional capabilities of your resource.

Important:

Do not call the super method when you override `getLocalTransaction` or `getXAResource` methods.

C Using the Services for Managing Namespace

Nodes

■ Overview	344
■ Connection Services	392
■ Adapter Service Services	402
■ Listener Services	410
■ Listener Notification Services	418
■ Polling Notification Services	430

Overview

The ADK provides a set of auxiliary Java services that you can use to replicate namespace nodes programmatically and to change the nodes' metadata appropriately when deploying an adapter to a different Integration Server. It provides services for connections, adapter services, listeners, listener notifications, and polling notifications.

Connection Services

These services are located in the `WmART.wm.art.dev.connection` package.

Service	Description
<code>wm.art.dev.connection:createConnectionNode</code>	Creates a new connection node in the specified package and folder.
<code>wm.art.dev.connection:deleteConnectionNode</code>	Removes the specified connection node.
<code>wm.art.dev.connection:fetchConnectionManagerMetadata</code>	Returns the connection manager metadata properties that are predefined for all connections.
<code>wm.art.dev.connection:fetchConnectionMetadata</code>	Queries the connection factory and returns the metadata for all properties supported by connections of the specified type.
<code>wm.art.dev.connection:updateConnectionNode</code>	Alters the values of an existing connection.

`wm.art.dev.connection:createConnectionNode`

This service creates a new connection node in the specified package and folder, and initializes the connection in a disabled state.

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.connection:fetchConnectionManagerMetadata` service to identify the supported connection manager properties and configure the `connectionManagerSettings` input parameter.
 - All connection manager properties have default values.
 - The default value is set in the `defaultValue` metadata attribute.
 - You can use the default values or override them with values that conform to the underlying data types of the properties.
 - The property values are not automatically set to their default values; you must explicitly set all the required properties.
 - If you omit a property, then no attempt is made to locate and assign default values to the property.
 - The connection manager properties may be optional or a required property.

- A required property is identified when the *isRequired* metadata attribute is set to `true`.
 - The absence of the *isRequired* attribute implies that the property is not required.
 - If you fail to set a required property, the `wm.art.dev.connection:createConnectionNode` service throws an exception.
 - You may have properties that might be required, based on the current value of some other property.
- The *poolable* connection manager property is always required.
 - The remaining connection manager properties depend on the value of *poolable* property.
 - If *poolable* is set to `true`, then the remaining connection manager properties must be assigned values as well.
 - If *poolable* is set to `false`, you may omit the remaining connection manager properties.
2. Use `wm.art.dev.connection:fetchConnectionMetada` service to identify the connection-specific properties and configure the *connectionSettings* input parameter.
- Connection-specific properties may or may not have default values, depending on the specific adapter's implementation.
 - Depending on the underlying data type of the property, it might not be possible to assign a default value to a connection specific property.
 - You may use the default values or override them with values that conform to the underlying data types of the properties.
 - The property values are not automatically set to their default values; you must explicitly set all the required properties.
 - If you omit a property, then no attempt is made to locate and assign default values to the property.
 - Connection-specific properties may be optional or a required property.
 - A required property is identified when the *isRequired* metadata attribute is set to `true`.
 - The absence of the *isRequired* attribute implies that the property is not required.
 - If you fail to set a required property, the `wm.art.dev.connection:createConnectionNode` service throws an exception.
 - You may have properties that might be required, based on the current value of some other property.
 - The number of properties to be configured is adapter-dependent. At a minimum, set those properties whose *isRequired* attribute is set to `true`.
3. Enable the connection using the `pub.art.connection:enableConnection` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

The resource domains registered by the connection factories are set in the connection's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Input Parameters

Name	Description
<i>connectionAlias</i>	String . Required. Name of the connection in the format: <code>folder:node</code> .
<i>packageName</i>	String . Required. Package where the connection is installed.
<i>adapterTypeName</i>	String . Required. Name of the adapter. Same as the value returned by calling <code>WmAdapter.getAdapterName</code> method.
<i>connectionFactoryType</i>	String . Required. Fully qualified path of the connection factory implementation class.
<i>connectionManagerSettings</i>	IData . Required. Structure for passing connection's manager property values. The connection's manager property are predefined for all connections
<i>poolable</i>	<p>Boolean. Required. Determines whether to pool the connection.</p> <p>Note: If <i>poolable</i> is <code>false</code>, then the following property values are not used:</p> <ul style="list-style-type: none"> ■ <i>minimumPoolSize</i> ■ <i>maximumPoolSize</i> ■ <i>poolIncrementSize</i> ■ <i>blockingTimeout</i> ■ <i>expireTimeout</i>
<i>minimumPoolSize</i>	Integer . Minimum number of connections retained in the pool.
<i>maximumPoolSize</i>	Integer . Maximum number of connections retained in the pool.
<i>poolIncrementSize</i>	Integer . Number of connections to add to the pool when additional connections are needed without exceeding the <i>maximumPoolSize</i> value.
<i>blockingTimeout</i>	Integer . Milliseconds to wait for a connection.

Name	Description
	<i>expireTimeout</i> Integer. Milliseconds of inactivity that may elapse prior to destroying the connection.
<i>connectionSettings</i>	IData Required. Structure for passing a connection's property values. The actual connection properties and their underlying data types vary from adapter to adapter. You must set a connection property's value in accordance with its data type. To determine the value, call the service <code>wm.art.dev.connection:fetchConnectionMetadata</code> and note the <i>parameterType</i> attribute for that property.
	<i>systemName</i> String. Internal name of the property.

For more information about the predefined connection manager properties, see [wm.art.dev.connection:fetchConnectionManagerMetadata](#)

For more information about the adapter-specific connection properties, see [wm.art.dev.connection:fetchConnectionMetadata](#)

Output Parameters

None.

Example

You must construct *connectionManagerSettings* and *connectionSettings* to create a connection node. The value of property's *systemName* is the internal name of the property. When constructing the *connectionSettings* input parameters, use this internal name as the key for setting a property's value. For example, if a connection defines a property named *hostPort*, then its *systemName* returned by `wm.art.dev.connection:fetchConnectionMetadata` service is *hostPort*. If the caller is a Java application, it might then use this information to set this property's value as follows:

```
IData pipeline = IDataFactory.create();
IDataCursor pipeCursor = pipeline.getCursor();
.
.
.
IData connSettings = IDataFactory.create();
IDataCursor connCursor = connSettings.getCursor();
connCursor.insertAfter("hostPort", new Integer(1234));
.
.
.
pipeCursor.insertAfter("connectionSettings", connSettings);
.
.
.
```

In this example, the *hostPort* property takes a `java.lang.Integer` value.

wm.art.dev.connection:deleteConnectionNode

This service deletes the specified connection node.

- You must disable the connection node before you delete it, using the `pub.art.connection:disableConnection` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

- The delete action is immediate and non-reversible, and returns no output data.
- You may assume that the service completed successfully if it does not throw a checked exception.

Input Parameters

Name	Description
<code>connectionAlias</code>	String . Required. Name of the connection in the format: <code>folder:node</code> .

Output Parameters

None.

wm.art.dev.connection:fetchConnectionManagerMetadata

This service fetches the connection manager properties that are predefined for all the connections, such as `poolable`, `minimumPoolSize`, `maximumPoolSize`, and others.

This service receives no inputs and returns an array of `connectionManagerProperties` structure. The `connectionManagerProperties` structure contains all metadata associated with each of the connection manager properties such as `systemName`, `parameterType`, `defaultValue`, and `isRequired` attributes, which you can use to configure a connection node.

Input Parameters

None.

Output Parameters

Name	Description
<code>connectionManagerProperties[n]</code>	IData . Required. An <i>n</i> -dimensioned array of connection manager properties.
<code>systemName</code>	String . Required. Internal property name. Values are:

Name	Description
	<ul style="list-style-type: none"> ■ poolable ■ minimumPoolSize ■ maximumPoolSize ■ poolIncrementSize ■ blockingTimeout ■ expireTimeout
<i>displayName</i>	String. Required. External property name displayed.
<i>description</i>	String. Required. Description of the property.
<i>parameterType</i>	<p>String. Required. Data type of the property.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean. ■ Arrays are not supported.
<i>groupURL</i>	String. Not applicable to connection manager properties.
<i>groupName</i>	String. Not applicable to connection manager properties.
<i>tupleName</i>	String. Not applicable to connection manager properties.
<i>treeName</i>	String. Not applicable to connection manager properties.
<i>treeDelimiter</i>	String. Not applicable to connection manager properties.
<i>resourceDomain</i>	String. Not applicable to connection manager properties.
<i>defaultValue</i>	String. Default value of the property.
<i>isRequired</i>	Boolean. Specifies whether the property is required.

wm.art.dev.connection:fetchConnectionMetadata

This service fetches the adapter-specific properties.

This service receives an adapter type name and a fully qualified connection factory class path, queries the connection factory and returns the metadata for all properties supported by the connections of that type.

Input Parameters

Name	Description
<i>adapterTypeName</i>	String . Required. Name of adapter. Same as the value returned by <code>WmAdapter.getAdapterName</code> method.
<i>connectionFactoryType</i>	String . Required. Fully qualified path of the connection factory implementation class.

Output Parameters

The service returns a *connectionProperties* array containing all metadata associated with each of the connection properties. You can use the following attributes to configure a connection node: *systemName*, *parameterType*, *defaultValue*, and *isRequired*.

Name	Description
<i>displayName</i>	String . Required. Adapter specific property name. Same as the value returned by <code>WmDescriptor.getDisplayName</code> method.
<i>description</i>	String . Required. Adapter specific property description. Same as the value returned by <code>WmDescriptor.getDescription</code> method.
<i>templateURL</i>	String . Required. URL of online help page for the connection.
<i>connectionProperties[n]</i>	IData[] . Required. An <i>n</i> -dimensioned array of properties.
<i>systemName</i>	String . Required. Adapter specific internal property name.
<i>displayName</i>	String . Required. External property name displayed.
<i>description</i>	String . Required. Description of the property.
<i>parameterType</i>	String . Required. Data type of the property. <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>,

Name	Description
	<p>java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean.</p> <p>■ Arrays are not supported.</p>
<i>groupURL</i>	String. URL of the group's help page.
<i>groupName</i>	String. Name of the group to which the property belongs.
<i>tupleName</i>	String. Name of the tuple to which the property belongs.
<i>treeName</i>	String. Name of the tree to which the property belongs.
<i>treeDelimiter</i>	String. Delimiter character used in the tree.
<i>resourceDomain</i>	String. Resource domain name to which the property belongs.
<i>defaultValue</i>	String. Default property value.
<i>isRequired</i>	Boolean. Specifies whether the property is required.
<i>isHidden</i>	Boolean. Specifies whether the property is displayable.
<i>isReadOnly</i>	Boolean. Specifies whether the property can be modified.
<i>isFill</i>	Boolean. Specifies whether the editors must pre-fill the property.
<i>isPassword</i>	Boolean. Specifies whether the property is a password.
<i>isMultiline</i>	Boolean. Specifies whether the property traverses multiple lines.
<i>isKey</i>	Boolean. Specifies whether the property is a key field.
<i>minSeqLength</i>	String. Lower bound of the sequence.
<i>minStringLength</i>	String. Minimum string length.
<i>maxStringLength</i>	String. Maximum string length.

Name	Description
<i>useParam</i>	String. Specifies whether the property is available for use.

Example

For example, a Java client might invoke this service as follows:

```
IData pipeline = IDataFactory.create();
IDataCursor pipeCursor = pipeline.getCursor();
pipeCursor.insertAfter("adapterTypeName",
"FooAdapter");
pipeCursor.insertAfter("connectionFactoryType",
"com.wm.adapters.FooConnFactory");
ExtendedConnectionUtils.fetchConnectionMetadata(pipeline);
.
.
.
```

wm.art.dev.connection:updateConnectionNode

This service updates the values of an existing connection.

- You must disable the connection node before you update it, using the `pub.art.connection:disableConnection` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.connection:fetchConnectionManagerMetadata` service to identify the supported connection manager properties and configure the *connectionManagerSettings* input parameter.
2. Use `wm.art.dev.connection:fetchConnectionMetadata` service to identify the connection-specific properties and configure the *connectionSettings* input parameter.
3. Provide values for the properties you want to change.

This service attempts to overlay these new values on the connection's current property values. The resulting set of merged property values are used to reconfigure the connection.

If you are not changing any connection manager or connection-specific properties, it is not necessary to pass in that container parameter. For example, if you are not changing any connection manager properties, you must not build and pass in the *connectionManagerSettings* parameter.

4. If you are providing explicit property values in *connectionManagerSettings* and *connectionSettings* parameter, then the values must conform to the underlying data types of those properties.

For an example of setting a connection property, see the Java code example in [wm.art.dev.connection:createConnectionNode](#).

Input Parameters

Name	Description
<i>connectionAlias</i>	String. Required. Name of the connection in the format: <code>folder:node</code> .
<i>connectionManagerSettings</i>	IData. Required. Structure for passing connection's manager property values. The connection's manager property are predefined for all connections.
<i>poolable</i>	<p>Boolean. Required. Determines whether to pool the connection.</p> <p>Note: If <i>poolable</i> is <code>false</code>, then the following property values are not used:</p> <ul style="list-style-type: none"> ■ <i>minimumPoolSize</i> ■ <i>maximumPoolSize</i> ■ <i>poolIncrementSize</i> ■ <i>blockingTimeout</i> ■ <i>expireTimeout</i>
<i>minimumPoolSize</i>	Integer. Minimum number of connections retained in the pool.
<i>maximumPoolSize</i>	Integer. Maximum number of connections retained in the pool.
<i>poolIncrementSize</i>	Integer. Number of connections to add to the pool when additional connections are needed and must not exceed <i>maximumPoolSize</i> .
<i>blockingTimeout</i>	Integer. Milliseconds to wait for a connection.
<i>expireTimeout</i>	Integer. Milliseconds of inactivity that may elapse prior to destroying the connection.
<i>connectionSettings</i>	IData. Required. Structure for passing a connection's property values. The actual connection properties and their underlying data types vary from adapter to adapter. You must set a connection property's value in accordance with its data type. To determine the value, call the service <code>wm.art.dev.connection:fetchConnectionMetadata</code> and note the <i>parameterType</i> attribute for that property.
<i>systemName</i>	String. Internal name of the property.

For more information about the predefined connection manager properties, see [wm.art.dev.connection:fetchConnectionManagerMetadata](#)

For more information about the adapter-specific connection properties, see [wm.art.dev.connection:fetchConnectionMetadata](#)

Output Parameters

None.

Adapter Service Services

These services are located in the WmART.wm.art.dev.service package.

Service	Description
wm.art.dev.service:createAdapterServiceNode	Creates a new adapter service node in the specified package and folder from the specified service template and connection alias.
wm.art.dev.service:deleteAdapterServiceNode	Removes a specified adapter service node.
wm.art.dev.service:fetchAdapterServiceTemplateMetadata	Returns all metadata for a specified adapter service template.
wm.art.dev.service:updateAdapterServiceNode	Alters the values of an existing adapter service.

wm.art.dev.service:createAdapterServiceNode

This service creates a new adapter service node in the specified package and folder from the specified service template and connection alias.

You must perform the following steps to populate the input pipeline:

1. Use [wm.art.dev.service:fetchAdapterServiceTemplateMetadata](#) service to identify the supported service template properties and configure the *adapterServiceSettings* input parameter.
 - The service's *inputFieldNames*, *inputFieldTypes*, *outputFieldNames*, and *outputFieldTypes* parameters in the *adapterServiceSettings* structure define the properties that comprise the adapter service's input and output signatures.
 - The data types of properties in the *adapterServiceSettings* structure are arrays of `java.lang.String` type.
 - A one-to-one correspondence exists between the elements in the **FieldNames* and **FieldTypes* arrays. For example, if the property names *abc*, *xyz*, and *foo* are inserted into the *outputFieldNames* parameter, then the service expects that exactly three data types will be inserted into *outputFieldTypes*, and that those data types correspond to the same element in *outputFieldNames*.
 - Adapter service properties may or may not have default values, depending on the specific adapter's implementation.

- Depending on the underlying data type of the property, it might not be possible to assign a default value to a property.
- You may use the default values or override them with values that conform to the underlying data types of the properties.
- The property values are not automatically set to their default values; you must explicitly set all the required properties.
- If you omit a property, then no attempt is made to locate and assign default values to the property.
- Adapter service properties may be optional or a required property.
 - A required property is identified when the *isRequired* metadata attribute is set to `true`.
 - The absence of the *isRequired* attribute implies that the property is not required.
 - If you fail to set a required property, the `wm.art.dev.service:createAdapterServiceNode` service throws an exception.
 - You may have properties that might be required, based on the current value of some other property.
 - The number of properties to be configured is adapter-dependent. At a minimum, set those properties whose *isRequired* attribute is set to `true`.

The resource domains registered by the adapter service template are set in the adapter service's properties according to the interdependencies between the resource domains. This includes input and output signatures since they are supported through resource domains. This service provides the properties *inputFieldNames*, *inputFieldTypes*, *outputFieldNames*, and *outputFieldTypes* for this purpose. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Input Parameters

Name	Description		
<i>serviceName</i>	String . Required. Namespace name of the new adapter service in the format: <code>folder:node</code> .		
<i>packageName</i>	String . Required. Package in which to install the adapter service.		
<i>connectionAlias</i>	String . Required. Namespace name of the connection in the format: <code>folder:node</code> .		
<i>serviceTemplate</i>	String . Required. Fully qualified pathname of the adapter service template class.		
<i>adapterServiceSettings</i>	IData . Required. Structure for passing the adapter's property values.		
	<table border="0"> <tr> <td style="padding-right: 20px;"><i>systemName</i></td> <td>String. Required. Adapter service specific internal property name.</td> </tr> </table>	<i>systemName</i>	String . Required. Adapter service specific internal property name.
<i>systemName</i>	String . Required. Adapter service specific internal property name.		

Name	Description
<i>inputFieldNames</i>	String[] . Names of the fields used in the adapter's input signature.
<i>inputFieldTypes</i>	String[] . Data types of the fields used in the adapter's input signature. Note: <ul style="list-style-type: none"> The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean. Arrays are supported.
<i>outputFieldNames</i>	String[] . Names of the fields used in the adapter's output signature.
<i>outputFieldTypes</i>	String[] . Data types of the fields used in the adapter's output signature. Note: <ul style="list-style-type: none"> The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean. Arrays are supported.

Output Parameters

None.

Example

You must construct *adapterServiceSettings* to create an adapter service node. The value of a property's *systemName* is the internal name of the property. When constructing the input parameter *adapterServiceSettings*, you must use this internal name as the key for setting a property's value. For example, if a service template defines a property named *sqlCommand*, then its *systemName* as returned by *fetchAdapterServiceTemplateMetadata* service is *sqlCommand*. If the caller is a Java application, it might then use this information to set this property's value as follows:

```
IData pipeline = IDataFactory.create();
IDataCursor pipeCursor = pipeline.getCursor();
```

```

.
.
.
IData svcSettings = IDataFactory.create();
IDataCursor svcCursor = svcSettings.getCursor();
svcCursor.insertAfter("sqlCommand", "SELECT * FROM fooTable");
.
.
.
pipeCursor.insertAfter("adapterServiceSettings", svcSettings);
.
.
.

```

In this example, the *sqlCommand* property takes a `java.lang.String` value.

wm.art.dev.service:deleteAdapterServiceNode

This service deletes the specified adapter service node.

- The delete action is immediate and non-reversible, and returns no output data.
- You may assume that the service completed successfully if it does not throw a checked exception.

Input Parameters

Name	Description
<i>serviceName</i>	String . Required. Name of the adapter service in the format: <code>folder:node</code> .

Output Parameters

None.

wm.art.dev.service:fetchAdapterServiceTemplateMetadata

This service fetches all metadata for an adapter service template for a specified connection alias and adapter service template class path.

This service receives a connection and a fully qualified adapter service template class path, and returns the metadata for all properties supported by the adapter service template.

This service returns an array of *templateProperties* structure each containing attributes such as *systemName*, *parameterType*, *defaultValue*, and *isRequired*.

Input Parameters

Name	Description
<i>connectionAlias</i>	String . Required. Name of the connection in the format: <code>folder:node</code> .
<i>serviceTemplate</i>	String . Required. Fully qualified pathname of the adapter service template class.

Output Parameters

Name	Description
<i>description</i>	String . Required. Adapter service template description. Same as the value returned by <code>WmDescriptor.getDisplayName</code> method.
<i>displayName</i>	String . Required. Adapter service template name displayed. Same as the value returned by <code>WmDescriptor.getDescription</code> method.
<i>templateURL</i>	String . Required. URL of the online help page for the adapter service.
<i>indexMaps[i]</i>	IData[] . Required. An <i>i</i> -dimensioned array of field maps.
<i>mapName</i>	String . Field map name.
<i>isVariable</i>	Boolean . Specifies whether the field map is of variable length.
<i>disableAppendAll</i>	Boolean . Disables all the buttons used for appending the rows for a field map in the Adapter Service Editor.
<i>templateProperties[n]</i>	IData . Required. An <i>n</i> -dimensioned array of properties.
<i>systemName</i>	String Required. Internal property name.
<i>displayName</i>	String Required. External property name.
<i>description</i>	String Required. Description of the property.
<i>parameterType</i>	String Required. Data type of property. <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. ■ Arrays are not supported.

Name	Description
<i>groupURL</i>	String. URL of the group's help page.
<i>groupName</i>	String. Name of the group to which the property belongs.
<i>tupleName</i>	String. Name of the tuple to which the property belongs.
<i>treeName</i>	String. Name of the tree to which the property belongs.
<i>treeDelimiter</i>	String. Delimiter character used in the tree.
<i>resourceDomain</i>	String. Resource domain name and dependencies.
<i>treeView</i>	String. Property name for which values are displayed in the tree structure for selection.
<i>defaultValue</i>	String. Default value of the property.
<i>isRequired</i>	Boolean. Specifies whether the property is required.
<i>isHidden</i>	Boolean. Specifies whether the property is displayable.
<i>isReadOnly</i>	Boolean. Specifies whether the property can be modified.
<i>isFill</i>	Boolean. Specifies whether the editors must pre-fill the property.
<i>isPassword</i>	Boolean. Specifies whether the property is a password.
<i>isMultiline</i>	Boolean. Specifies whether the property traverses multiple lines.
<i>isKey</i>	Boolean. Specifies whether the property is a key field.
<i>minSeqLength</i>	String. Lower bound of the sequence.
<i>maxSeqLength</i>	String. Upper bound of the sequence.
<i>minStringLength</i>	String. Minimum string length.
<i>maxStringLength</i>	String. Maximum string length.
<i>useParam</i>	String. Specifies whether the property is available for use.

wm.art.dev.service:updateAdapterServiceNode

This service updates an existing adapter service node.

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.service:fetchAdapterServiceTemplateMetadata` service to identify the supported service template properties and configure the `adapterServiceSettings` input parameter.
 - The value of the `systemName` metadata attribute is the internal name of a property. Use this internal name as the key for setting a property's value when constructing the `adapterServiceSettings` input parameter.
2. Set only those properties that you want to change.

This service attempts to overlay these new values on the adapter service's current property values. The resulting set of merged property values are used to reconfigure the adapter service.
3. Update the properties depending on the data type.
 - If a property's data type is non-primitive (derived from `java.lang.Object`), you may clear a property's current value by setting its value to `null`.
 - If a property's data type is Java primitive, then the property value cannot be cleared; however the property can be updated.
4. If you are providing explicit property values in the `adapterServiceSettings` parameter, then the values must conform to the underlying data types of those properties.
5. Change the connection resource that the adapter service uses by providing a new `connectionAlias` input parameter.
 - If you omit the `connectionAlias` parameter, the adapter service will continue to use its current connection resource.
 - If you are changing only the connection resource, it is not necessary to provide the `adapterServiceSettings` input parameter.

The resource domains registered by the adapter service template are set in the adapter service's properties according to the interdependencies between the resource domains. This includes input and output signatures since they are supported through resource domains. This service provides the properties `inputFieldNames`, `inputFieldTypes`, `outputFieldNames`, and `outputFieldTypes` for this purpose. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

For an example of setting an adapter service property, see the Java code example in [wm.art.dev.service:createAdapterServiceNode](#).

Input Parameters

Name	Description
<i>serviceName</i>	String . Required. Name of the existing adapter service in the format: <code>folder:node</code> .
<i>connectionAlias</i>	String . Required. Name of the connection in the format: <code>folder:node</code> .
<i>adapterServiceSettings</i>	IData . Required. Structure for passing the adapter's property values.
<i>systemName</i>	String . Required. Adapter service specific internal property name.
<i>inputFieldNames</i>	String[] . Names of the fields used in the adapter's input signature.
<i>inputFieldTypes</i>	<p>String[]. Data types of the fields used in the adapter's input signature.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. ■ Arrays are supported.
<i>outputFieldNames</i>	String[] . Names of the fields used in the adapter's output signature.
<i>outputFieldTypes</i>	<p>String[]. Data types of the fields used in the adapter's output signature.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. ■ Arrays are supported.

Output Parameters

None.

Listener Services

These services are located in the `WmART.wm.art.dev.listener` package.

Service	Description
wm.art.dev.listener:analyzeListenerNodes	Logs the data for listeners.
wm.art.dev.listener:createListenerNode	Creates a new instance of a listener in the specified package and folder from the specified listener template and connection alias.
wm.art.dev.listener:deleteListenerNode	Removes a specified instance of a listener.
wm.art.dev.listener:fetchListenerTemplateMetadata	Returns all metadata supported by that listener template.
wm.art.dev.listener:updateListenerNode	Alters the values of an existing listener.
wm.art.dev.listener:updateRegisteredNotifications	Checks the registration of listener notifications.

wm.art.dev.listener:analyzeListenerNodes

This service logs the data for listeners in the server log file. The data includes the names of associated listener notifications, the class name of listener notifications, their status (active or disabled), and whether the associated listener notification is linked with the same listener or not.

Note:

A listener can be used by multiple listener notifications, but a listener notification will have only one listener node.

Input Parameters

None.

Output Parameters

None.

wm.art.dev.listener:createListenerNode

This service creates a new listener node in the specified package and folder from the specified listener template and connection alias, and initializes the new listener in a disabled state.

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.service:fetchListenerTemplateMetadata` service to identify the supported listener template properties and configure the `listenerSettings` input parameter.

- Listener properties may or may not have default values, depending on the specific adapter's implementation.
 - Depending on the underlying data type of the property, it might not be possible to assign a default value to a property.
 - You may use the default values or override them with values that conform to the underlying data types of the properties.
 - The property values are not automatically set to their default values; you must explicitly set all the required properties.
 - If you omit a property, then no attempt is made to locate and assign default values to the property.
 - Listener properties may be optional or a required property.
 - A required property is identified when the *isRequired* metadata attribute is set to `true`.
 - The absence of the *isRequired* attribute implies that the property is not required.
 - If you fail to set a required property, the `wm.art.dev.listener:createListenerNode` service throws an exception.
 - You may have properties that might be required, based on the current value of some other property.
 - The number of properties to be configured is adapter-dependent. At a minimum, set those properties whose *isRequired* attribute is set to `true`.
2. Activate the listener using the `pub.art.listener:enableListener` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

The resource domains registered by the listener service template are set in the listener's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Input Parameters

Name	Description
<i>listenerName</i>	String . Required. Name of the listener in the format: <code>folder:node</code> .
<i>packageName</i>	String . Required. Package where the listener is installed.
<i>connectionAlias</i>	String . Required. Name of the connection in the format: <code>folder:node</code> .
<i>listenerTemplate</i>	String . Required. Fully qualified pathname of the listener template class.
<i>listenerSettings</i>	IData . Required. Structure for passing the listener's property values.

Name	Description
<i>systemName</i>	<p>String. Required. Internal name of the property.</p> <ul style="list-style-type: none"> The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean.
<i>retryLimit</i>	<p>String. Required. Number of times that the system must attempt to start the listener if the initial attempt fails.</p>
<i>retryBackoffTimeout</i>	<p>String. Required. Number of seconds the system must wait between each attempt to start the listener. This field is irrelevant if the value of Retry Limit is 0.</p>

Output Parameters

None.

Example

You must construct *listenerSettings* to create a listener node. The value of a property's *systemName* is the internal name of the property. When constructing the input parameter *listenerSettings*, you must use this internal name as the key for setting a property's value. For example, if a listener template defines a property named *portNumber*, then its *systemName* as returned by `wm.art.dev.listener.fetchListenerTemplateMetadata` service is *portNumber*. If the caller is a Java application, it might then use this information to set this property's value as follows:

```

IData pipeline = IDataFactory.create();
IDataCursor pipeCursor = pipeline.getCursor();
.
.
.
IData lstnrSettings = IDataFactory.create();
IDataCursor lstnrCursor = lstnrSettings.getCursor();
lstnrCursor.insertAfter("portNumber", new Integer((int)8888));
.
.
.
pipeCursor.insertAfter("listenerSettings", lstnrSettings);
.
.
.

```

In this example, the *portNumber* property takes a `java.lang.Integer` value.

wm.art.dev.listener:deleteListenerNode

This service deletes a specified listener node.

- You must disable the listener node before you delete it, using the `pub.art.listener:disableListener` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

- The delete action is immediate and non-reversible, and returns no output data.
- You may assume that the service completed successfully if it does not throw a checked exception.

Input Parameters

Name	Description
<i>listenerName</i>	String . Required. Name of the listener in the format: <code>folder:node</code> .

Output Parameters

None.

wm.art.dev.listener:fetchListenerTemplateMetadata

This service fetches all metadata supported by the listener template for a specified connection alias and listener template class path.

This service takes a valid connection alias and listener template class path, and returns an array of *templateProperties* structure each containing attributes such as *systemName*, *parameterType*, *defaultValue*, and *isRequired*.

Input Parameters

Name	Description
<i>connectionAlias</i>	String . Required. Name of an existing connection in the format: <code>folder:node</code> .
<i>listenerTemplate</i>	String . Required. Fully qualified pathname of listener template class.

Output Parameters

Name	Description
<i>description</i>	String. Required. Listener template description. Same as the value returned by <code>WmDescriptor.getDisplayName</code> method.
<i>displayName</i>	String. Required. Listener template name displayed. Same as the value returned by <code>WmDescriptor.getDescription</code> method.
<i>listAllConnection</i>	String. Default template metadata
<i>templateURL</i>	String. Required. URL of the online help page for the listener.
<i>requiresConnection</i>	Boolean. Specifies whether the listener requires a connection.
<i>indexMaps[i]</i>	IData[]. Required. An <i>i</i> -dimensioned array of field maps.
<i>mapName</i>	String. Field map name.
<i>isVariable</i>	Boolean. Specifies whether the field map is variable length.
<i>disableAppendAll</i>	Boolean. Disables all the buttons used for appending the rows for a field map in the Adapter Service Editor.
<i>templateProperties[n]</i>	IData[]. Required. An <i>n</i> -dimensioned array of properties.
<i>systemName</i>	String. Required. Internal property name.
<i>displayName</i>	String. Required. External displayable property name.
<i>description</i>	String. Required. Description of the property.
<i>parameterType</i>	<p>String. Required. Data type of property.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. ■ Arrays are not supported.
<i>groupURL</i>	String. URL of the group's help page.
<i>groupName</i>	String. Name of the group to which the property belongs.

Name	Description
<i>tupleName</i>	String. Name of the tuple to which the property belongs.
<i>treeName</i>	String. Name of the tree to which the property belongs.
<i>treeDelimiter</i>	String. Delimiter character used in the tree.
<i>resourceDomain</i>	String. Resource domain name to which the property belongs.
<i>treeView</i>	String. Property name for which values are displayed in the tree structure for selection.
<i>defaultValue</i>	String. Default value of the property.
<i>isRequired</i>	Boolean. Specifies whether the property is required.
<i>isHidden</i>	Boolean. Specifies whether the property is displayable.
<i>isReadOnly</i>	Boolean. Specifies whether the property can be modified.
<i>isFill</i>	Boolean. Specifies whether the editors should pre-fill the property.
<i>isPassword</i>	Boolean. Specifies whether the property is a password.
<i>isMultiline</i>	Boolean. Specifies whether the property traverses multiple lines.
<i>isKey</i>	Boolean. Specifies whether the property is a key field.
<i>minSeqLength</i>	String. Lower bound of the sequence.
<i>maxSeqLength</i>	String. Upper bound of the sequence.
<i>minStringLength</i>	String. Minimum string length.
<i>maxStringLength</i>	String. Maximum string length.
<i>useParam</i>	String. Specifies whether the property is available for use.

wm.art.dev.listener:updateListenerNode

This service updates an existing listener node.

- You must disable the listener before updating its properties, using the `pub.art.listener:disableListener` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.listener:fetchListenerTemplateMetadata` service to identify the supported service template properties and configure the `listenerSettings` input parameter.
 - The value of the `systemName` metadata attribute is the internal name of a property. Use this internal name as the key for setting a property's value when constructing the input parameter `listenerSettings`.
2. Set only those properties that you want to change.

Note:

This service attempts to overlay these new values on the listener's current property values. The resulting set of merged property values are used to reconfigure the listener.

3. Update the property depending on the data type.
 - If a property's data type is non-primitive (derived from `java.lang.Object`), you may clear a property's current value by setting its value to `null`.
 - If a property's data type is Java primitive, then the property value cannot be cleared; however the property can be updated.
4. If you are providing explicit property values in `listenerSettings` parameter, then the values must conform to the underlying data types of those properties.
5. Change the connection resource that the listener uses by providing a new `connectionAlias` input parameter.
 - If you omit the `connectionAlias` parameter, the listener continues to use its current connection resource.
 - If you are changing only the connection resource, it is not necessary to provide the `listenerSettings` input parameter.

The resource domains registered by the listener templates are set in the listener's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

For an example of setting an listener property, see the Java code example in [wm.art.dev.listener:createListenerNode](#).

Input Parameters

Name	Description
<i>listenerName</i>	String . Required. Name of an existing listener in the format: <code>folder:node</code> .
<i>connectionAlias</i>	String . Required. Name of the connection in the format: <code>folder:node</code> .
<i>listenerSettings</i>	IData . Required. Structure for passing the listener's property values.
<i>systemName</i>	<p>String. Required. Internal property name.</p> <ul style="list-style-type: none"> The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. Arrays are not supported.
<i>retryLimit</i>	String . Required. Number of times that the system must attempt to start the listener if the initial attempt fails.
<i>retryBackoffTimeout</i>	String . Required. Number of seconds the system must wait between each attempt to start the listener. This field is irrelevant if the value of Retry Limit is <code>0</code> .

Output Parameters

None.

wm.art.dev.listener:updateRegisteredNotifications

This service checks listener notifications, finds the linked listener, and then checks whether the listener notification is also registered for the same listener or not. If not, then the service registers the listener notification with the linked listener.

Note:

A listener can be used by multiple listener notifications, but a listener notification has only one listener node.

Input Parameters

Name	Description
<i>listenerNodeName</i>	String . Specifies the name of the WmART-based listener for which the notifications must be updated.

Name	Description
	<ul style="list-style-type: none"> ■ Specify the listener node name to update a specific listener. ■ Specify * to update all the WmART-based listeners .

Output Parameters

None.

Listener Notification Services

These services are located in the WmART.wm.art.dev.notification package.

Service	Description
wm.art.dev.notification:analyzeListenerNotifications	Logs the data for listener notifications.
wm.art.dev.notification:createListenerNotificationNode	Creates a new instance of a synchronous or asynchronous listener notification in the specified package and folder from the specified notification template and connection alias.
wm.art.dev.notification:deleteListenerNotificationNode	Removes a specified instance of listener notification.
wm.art.dev.notification:fetchListenerNotificationTemplateMetadata	Returns all metadata for a specified listener notification template.
wm.art.dev.notification:updateListenerNotificationNode	Alters the values of an existing synchronous or asynchronous listener notification.

wm.art.dev.notification:analyzeListenerNotifications

This service logs the data for listener notifications. The data includes the name of a listener to which a listener notification is linked, and the class name of the listener. The service also checks whether the listener notification is registered with the listener or not.

Note:

A listener can be used by multiple listener notifications, but a listener notification will have only one listener node.

Input Parameters

None.

Output Parameters

None.

wm.art.dev.notification:createListenerNotificationNode

This service creates a new synchronous or asynchronous listener notification node in the specified package and folder from the specified notification template and connection alias, and initializes the listener notification node in a disabled state.

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.notification:fetchListenerNotificationTemplateMetadata` service to identify the supported listener notification template properties and configure the `notificationSettings` input parameter.
 - Listener notification properties may or may not have default values, depending on the specific adapter's implementation.
 - Depending on the underlying data type of the property, it might not be possible to assign a default value to a property.
 - You may use the default values or override them with values that conform to the underlying data types of the properties.
 - The property values are not automatically set to their default values; you must explicitly set all the required properties.
 - If you omit a property, then no attempt is made to locate and assign default values to the property.
 - Listener notification properties may be optional or a required property.
 - A required property is identified when the `isRequired` metadata attribute is set to `true`.
 - The absence of the `isRequired` attribute implies that the property is not required.
 - If you fail to set a required property, the `wm.art.dev.notification:createListenerNotificationNode` service throws an exception.
 - You may have properties that might be required, based on the current value of some other property.
 - The number of properties to be configured is adapter-dependent. At a minimum, set those properties whose `isRequired` attribute is set to `true`.
2. The particular combination of input parameters you specify determines whether you create a synchronous or an asynchronous listener notification as described in the section for input parameters. This service throws an exception if you specify invalid or ambiguous combinations of input parameters.
3. When an asynchronous listener notification is triggered, Integration Server generates a runtime publishable output document.

- The names and types of the data fields in the publishable output document is predefined.
 - Specify the names and types of the data fields in the publishable output document in the service's *publishableRecordDef* input parameter.
 - The *publishableRecordDef* input parameter consists of *fieldNames* and *fieldTypes* properties.
 - The data types of properties in the *publishableRecordDef* structure are arrays of `java.lang.String` type.
 - The *fieldNames* and *fieldTypes* properties are an array of `String`.
 - A one-to-one correspondence exists between the elements in these *fieldNames* and *fieldTypes* arrays.
 - The values assigned to the *fieldNames* and *fieldTypes* properties must correspond to fields that the listener notification class outputs in its `runNotification` method.
 - The service execution may fail if an empty *publishableRecordDef* is specified.
4. Activate the listener notification node using the `pub.art.notification:enableListenerNotification` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

Note:

1. This service does not validate that the fields in the publishable document are actually generated by the notification.
2. This service creates a publishable document node in a *notificationNamePublishDocument* folder where *notificationName* is the value you specify for the *notificationName* input parameter. The name of publishable document is not configurable.

The resource domains registered by the listener notification templates are set in the listener notification's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Note:

1. Synchronous listener notification classes must extend the `WmSyncListenerNotification` class.
2. Asynchronous listener notifications must extend the `WmAsyncListenerNotification` class.

Input Parameters

Name	Description
<i>notificationName</i>	String. Required. Name of the listener notification in the format: <code>folder:node</code> .
<i>packageName</i>	String. Required. Package where the listener notification is installed.
<i>listenerNode</i>	String. Required. Name of the listener in the format: <code>folder:node</code> .

Name	Description
<i>notificationTemplate</i>	String. Required. Fully qualified pathname of the listener notification template class.
<i>notificationSettings</i>	IData. Required. Structure for passing the listener notification's property values.
<i>systemName</i>	<p>String. Required. Internal name of the property.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean. ■ Arrays of all the above data types are also supported.
Properties applicable to synchronous listener notifications only	
<i>serviceName</i>	String. Required. Name of the service to invoke in the format: folder:node. This parameter must not be null.
<i>executionMode</i>	IData. Structure to identify whether a service defined in <i>serviceName</i> is invoked or a document is published.
<i>mode</i>	<p>String. Possible values are:</p> <ul style="list-style-type: none"> ■ <code>publishAndWait</code>. Publish the document and wait for the response. ■ <code>invokeService</code>. Invoke the service defined in <i>serviceName</i> parameter.
<i>local</i>	Boolean. Enables publishing the document locally.
<i>waitTime</i>	String. Time in millisecond to wait for the response in an synchronous listener notification.
<i>jmsSettings</i>	IData. Structure specifying the JMS setting.
<i>isJMSConfigured</i>	Boolean. Specifies whether the document is published to a JMS provider.
<i>ConnectionAliasName</i>	String. Required. Connection name of the JMS provider. This field is relevant if the value of isJMSConfigured is true.

Name	Description
	<p><i>DestinationName</i> String. Required. Name of the destination where the document is published. This field is relevant if the value of isJMSConfigured is <code>true</code>.</p>
	<p><i>DestinationType</i> String. Required. Type of destination where the document is published. Possible values are:</p> <ul style="list-style-type: none"> ■ Queue ■ Topic <p>This field is relevant if the value of isJMSConfigured is <code>true</code>.</p>
<i>requestRecordDef</i>	<p>IData. Required. Structure specifying the definition of the request document sent to <i>serviceName</i>. This parameter must not be <code>null</code>, but may be empty. This property contains the following fields:</p>
	<p><i>fieldNames</i> String[]. Names of the fields used in the listener notification's request document.</p>
	<p><i>fieldTypes</i> String[]. Data types of the fields used in the listener notification's request document.</p>
<i>replyRecordDef</i>	<p>IData. Required. Structure specifying the definition of the reply document received from <i>serviceName</i>. This parameter must not be <code>null</code>, but may be empty. This property contains the following fields:</p>
	<p><i>fieldNames</i> String[]. Names of the fields used in the listener notification's reply document.</p>
	<p><i>fieldTypes</i> String[]. Data types of the fields used in the listener notification's reply document.</p>
Properties applicable to asynchronous listener notifications only	
<i>publishableRecordDef</i>	<p>IData. Required. Structure specifying the definition of the runtime publishable output document. This property contains the following fields:</p>
	<p><i>fieldNames</i> String[]. Names of the fields used in the listener notification's output document.</p>
	<p><i>fieldTypes</i> String[]. Data types of the fields used in the listener notification's output document.</p>

Output Parameters

None.

Example

You must construct *notificationSettings* to create a listener notification node. The value of a property's *systemName* is the internal name of the property. When constructing the *notificationSettings* input parameter, you must use this internal name as the key for setting a property's value. For example, if a listener notification template defines a property named *foo*, then its *systemName* as returned by *fetchListenerNotificationTemplateMetadata* service is *foo*. If the caller is a Java application, it might then use this information to set this property's value as follows:

```
IData pipeline = IDataFactory.create();
IDataCursor pipeCursor = pipeline.getCursor();
.
.
.
IData ntfySettings = IDataFactory.create();
IDataCursor ntfyCursor = ntfySettings.getCursor();
ntfyCursor.insertAfter("foo", "bar");
.
.
.
pipeCursor.insertAfter("notificationSettings",
ntfySettings);
.
.
.
```

In this example, the *foo* property takes a `java.lang.String` value.

wm.art.dev.notification:deleteListenerNotificationNode

This service deletes the specified listener notification node.

- You must disable the listener notification node before you delete it, using the `pub.art.notification:disableListenerNotification` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

- The delete action is immediate and non-reversible, and returns no output data.
- Set the *cascadeDelete* flag to propagate the deletion across the Broker.
- You may assume that the service completed successfully if it does not throw a checked exception.

Input Parameters

Name	Description
<i>notificationName</i>	String . Required. Name of the listener notification node in the format: <code>folder:node</code> .
<i>cascadeDelete</i>	Boolean . Possible values are:

Name	Description
	<ul style="list-style-type: none"> ■ true. Propagates the deletion across the Broker. ■ false. Default.

Output Parameters

None.

wm.art.dev.notification:fetchListenerNotificationTemplateMetadata

This service fetches all metadata supported by the listener notification template for a specified connection alias and listener notification template class path.

This service takes a valid listener node and listener notification template class path, and returns an array of *templateProperties* structure, each containing attributes such as *systemName*, *parameterType*, *defaultValue*, and *isRequired*.

Input Parameters

Name	Description
<i>listenerNode</i>	String . Required. Name of an existing listener in the format: <code>folder:node</code> .
<i>notificationTemplate</i>	String . Required. Fully qualified pathname of listener notification template class.

Output Parameters

Name	Description				
<i>description</i>	String . Required. Listener template description. Same as the value returned by <code>WmDescriptor.getDisplayName</code> method.				
<i>displayName</i>	String . Required. Listener template name displayed. Same as the value returned by <code>WmDescriptor.getDescription</code> method.				
<i>templateURL</i>	String . Required. URL of the online help page for the listener.				
<i>indexMaps[i]</i>	IData[] . Required. An <i>i</i> -dimensioned array of field maps.				
	<table border="1"> <tbody> <tr> <td><i>mapName</i></td> <td>String. Field map name.</td> </tr> <tr> <td><i>isVariable</i></td> <td>Boolean. Specifies whether the field map is variable length.</td> </tr> </tbody> </table>	<i>mapName</i>	String . Field map name.	<i>isVariable</i>	Boolean . Specifies whether the field map is variable length.
<i>mapName</i>	String . Field map name.				
<i>isVariable</i>	Boolean . Specifies whether the field map is variable length.				

Name	Description
<i>disableAppendAll</i>	Boolean. Disables all the buttons used for appending the rows for a field map in the Adapter Service Editor.
<i>templateProperties[n]</i>	IData. Required. An <i>n</i> -dimensioned array of properties.
<i>systemName</i>	String Required. Internal property name.
<i>displayName</i>	String Required. External property name displayed.
<i>description</i>	String Required. Property description.
<i>parameterType</i>	<p data-bbox="837 665 1338 695">String Required. Data type of property.</p> <ul data-bbox="837 726 1474 1018" style="list-style-type: none"> <li data-bbox="837 726 1474 961">■ The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean. <li data-bbox="837 993 1219 1018">■ Arrays are not supported.
<i>groupURL</i>	String. URL of the group's help page.
<i>groupName</i>	String. Name of the group to which the property belongs.
<i>tupleName</i>	String. Name of the tuple to which the property belongs.
<i>treeName</i>	String. Name of the tree to which the property belongs.
<i>treeDelimiter</i>	String. Delimiter character used in the tree.
<i>resourceDomain</i>	String. Resource domain name to which the property belongs.
<i>treeView</i>	String. Property name for which values are displayed in the tree structure for selection.
<i>defaultValue</i>	String. Default value of the property.
<i>isRequired</i>	Boolean. Specifies whether the property is required.
<i>isHidden</i>	Boolean. Specifies whether the property is displayable.

Name	Description
<i>isReadOnly</i>	Boolean. Specifies whether the property can be modified.
<i>isFill</i>	Boolean. Specifies whether the editors should pre-fill the property.
<i>isPassword</i>	Boolean. Specifies whether the property is a password.
<i>isMultiline</i>	Boolean. Specifies whether the property traverses multiple lines.
<i>isKey</i>	Boolean. Specifies whether the property is a key field.
<i>minSeqLength</i>	String. Lower bound of the sequence.
<i>maxSeqLength</i>	String. Upper bound of the sequence.
<i>minStringLength</i>	String. Minimum string length.
<i>maxStringLength</i>	String. Maximum string length.
<i>useParam</i>	String. Specifies whether the property is available for use.

wm.art.dev.notification:updateListenerNotificationNode

This service updates the existing synchronous or asynchronous listener notification node.

- You must disable the listener notification node before updating its properties, using the `pub.art.notification:disableListenerNotification` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

You must perform the following steps to populate the input pipeline:

1. Use `fetchListenerNotificationTemplateMetadata` service to identify the supported service template properties and configure the *listenerNode* and *notificationSettings* input parameter.
 - The value of the *systemName* metadata attribute is the internal name of a property. Use this internal name as the key for setting a property's value when constructing the input parameter *notificationSettings*.
2. Set only those properties that you want to change.

This service attempts to overlay these new values on the listener notification's current property values. The resulting set of merged property values are used to reconfigure the listener.

3. If you are providing explicit property values in *notificationSettings* parameter, then the values must conform to the underlying data types of those properties.
4. Change the listener resource that the listener notification uses by providing a new *listenerNode* input parameter.
 - If you omit the *listenerNode* parameter, the listener notification continues to use its current listener resource.
 - If you are changing only the listener resource, it is not necessary to provide the *notificationSettings* or any other input parameter.
5. A synchronous listener notification contains three additional input parameters:
 - a. *serviceName*. Specifies the node name of a separate service that is invoked by the notification at runtime. If you omit this parameter, the notifications still retains the current value of the parameter.
 - b. *requestRecordDef* and *replyRecordDef*. Document record definitions that the notification uses to format messages when communicating with this service. You can modify either or both of these definitions.
 - The *requestRecordDef* and *replyRecordDef* input parameters consists of *fieldNames* and *fieldTypes* properties.
 - The *fieldNames* and *fieldTypes* properties are an array of `java.lang.String`.
 - A one-to-one correspondence exists between the elements in these *fieldNames* and *fieldTypes* arrays.

Note:

This service cannot verify this assertion. If you configure either record definition incorrectly, the consequence may not manifest until runtime.

6. An asynchronous listener notification use the following input parameters:
 - a. *publishableRecordDef*. Redefine the structure of an asynchronous notification's output document.
 - The *publishableRecordDef* input parameter consists of *fieldNames* and *fieldTypes* properties.
 - The *fieldNames* and *fieldTypes* properties are an array of `java.lang.String`.
 - A one-to-one correspondence exists between the elements in these *fieldNames* and *fieldTypes* arrays.
 - The values assigned to the *fieldNames* and *fieldTypes* properties must correspond to fields that the listener notification class outputs in its `runNotification` method.

Note:

This service cannot verify this assertion. If you configure either record definition incorrectly, the consequence may not manifest until runtime.

7. The *requestRecordDef*, *replyRecordDef*, and *publishableRecordDef* input parameters can be replaced in their entirety. The service does not attempt to merge the values in these parameters with the notification's current values. For instance, if you want to redefine a synchronous notification's reply record definition, you must define the entire record in the *replyRecordDef* input parameter.
8. The two *listenerNode* and *notificationSettings* input parameters apply to both synchronous and asynchronous listener notifications.

Note:

1. You cannot convert a synchronous notification to an asynchronous notification.
2. Similarly you cannot convert an asynchronous notification to a synchronous notification.

The resource domains registered by the listener notification templates are set in the listener notification's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Input Parameters

Name	Description
<i>notificationName</i>	String . Required. Name of the existing listener notification in the format: <code>folder:node</code> .
<i>listenerNode</i>	String . Required. Name of the associated listener in the format: <code>folder:node</code> .
<i>notificationSettings</i>	IData . Required. Structure for passing the listener notification's property values.
<i>systemName</i>	<p>String. Required. Internal property name.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. ■ Arrays of all the above data types are also supported.

Properties applicable to synchronous listener notifications only

<i>serviceName</i>	String . Required. Name of the service to invoke in the format: <code>folder:node</code> . This parameter must not be <code>null</code> .
<i>executionMode</i>	IData . Structure to identify whether a service defined in <i>serviceName</i> is invoked or a document is published.

Name	Description
<i>mode</i>	<p>String. Possible values are:</p> <ul style="list-style-type: none"> ■ <code>publishAndWait</code>. Publish the document and wait for the response. ■ <code>invokeService</code>. Invoke the service defined in <code>serviceName</code> parameter.
<i>local</i>	<p>Boolean. Enables publishing the document locally.</p>
<i>waitTime</i>	<p>String. Time in millisecond to wait for the response in an synchronous listener notification.</p>
<i>jmsSettings</i>	<p>IData. Structure specifying the JMS settings.</p>
<i>isJMSConfigured</i>	<p>Boolean. Specifies whether the document is published to a JMS provider.</p>
<i>ConnectionAliasName</i>	<p>String. Required. Connection name of the JMS provider. This field is relevant if the value of isJMSConfigured is true.</p>
<i>DestinationName</i>	<p>String. Required. Name of the destination where the document is published. This field is relevant if the value of isJMSConfigured is true.</p>
<i>DestinationType</i>	<p>String. Required. Type of destination where the document is published. Possible values are:</p> <ul style="list-style-type: none"> ■ <code>Queue</code> ■ <code>Topic</code> <p>This field is relevant if the value of isJMSConfigured is true.</p>
<i>requestRecordDef</i>	<p>IData. Required. Specifies the definition of the request document sent to <code>serviceName</code>. This parameter must not be <code>null</code>, but may be empty. This property contains the following fields:</p>
<i>fieldNames</i>	<p>String[]. Names of the fields used in the listener notification's request document.</p>
<i>fieldTypes</i>	<p>String[]. Data types of the fields used in the listener notification's request document.</p>
<i>replyRecordDef</i>	<p>IData. Required. Specifies the definition of the reply document received from <code>serviceName</code>. This parameter must not be <code>null</code>, but may be empty.</p>

Name	Description
<i>fieldNames</i>	String[] . Names of the fields used in the listener notification's reply document.
<i>fieldTypes</i>	String[] . Data types of the fields used in the listener notification's reply document.
Properties applicable to asynchronous listener notifications only	
<i>publishableRecordDef</i>	IData . Required. Specifies the definition of the runtime publishable output document. This property contains the following fields:
<i>fieldNames</i>	String[] . Names of the fields used in the listener notification's output document.
<i>fieldTypes</i>	String[] . Data types of the fields used in the listener notification's output document.

Output Parameters

None.

Polling Notification Services

These services are located in the WmART.wm.art.dev.notification package.

Service	Description
wm.art.dev.notification:createPollingNotificationNode	Creates a new instance of a polling notification in the specified package and folder from the specified notification template and connection alias.
wm.art.dev.notification:deletePollingNotificationNode	Removes an instance of a specified polling notification.
wm.art.dev.notification:fetchPollingNotificationTemplateMetadata	Returns all metadata for a specified polling notification template, and returns any scheduling properties that are set for the notification.
wm.art.dev.notification:updatePollingNotificationNode	Alters the characteristics of an existing polling notification.

wm.art.dev.notification:createPollingNotificationNode

This service creates a new instance of a polling notification in the specified package and folder from the specified notification template and connection alias, and initializes the new polling notification in a disabled state.

You must perform the following steps to populate the input pipeline:

1. Use `fetchPollingNotificationTemplateMetadata` service to identify the supported polling notification template properties and configure the input parameter `notificationSettings` structure.
 - Polling notification properties may or may not have default values, depending on the specific adapter's implementation.
 - Depending on the underlying data type of the property, it might not be possible to assign a default value to a property.
 - You may use the default values or override them with values that conform to the underlying data types of the properties.
 - The property values are not automatically set to their default values; you must explicitly set all the required properties.
 - If you omit a property, then no attempt is made to locate and assign default values to the property.
 - Polling notification properties may be optional or a required property.
 - A required property is identified when the `isRequired` metadata attribute is set to `true`.
 - The absence of the `isRequired` attribute implies that the property is not required.
 - If you fail to set a required property, the `wm.art.dev.notification:createPollingNotificationNode` service throws an exception.
 - You may have properties that might be required, based on the current value of some other property.
 - The number of properties to be configured is adapter-dependent. At a minimum, set those properties whose `isRequired` attribute is set to `true`.
2. This service creates a publishable document node in a `notificationNamePublishDocument` folder where `notificationName` is the value you specify for the `notificationName` input parameter. The name of publishable document is not configurable.
3. When a polling notification is triggered, Integration Server generates a runtime publishable output document.
 - The names and types of the data fields in the publishable output document is predefined.
 - Specify the names and types of the data fields in the publishable output document in the service's `publishableRecordDef` input parameter in `notificationSettings` parameter.
 - The `fieldNames` and `fieldTypes` properties of `publishableRecordDef` are an array of `String`. A one-to-one correspondence exists between the elements in these two arrays.

- The values assigned to the *fieldNames* and *fieldTypes* properties of *publishableRecordDef* must correspond to fields that the notification class outputs in its *runNotification* method.
 - The service execution may fail if an empty *publishableRecordDef* is specified.
5. A newly created polling notification is not assigned a delivery schedule. You must configure the polling notification's delivery schedule before enabling it. Provide a valid *scheduleSettings* input parameter in the call to *createPollingNotificationNode* or *updatePollingNotificationNode* service.
 6. You can activate the polling notification using the *pub.art.notification:enablePollingNotification* service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

Note:

This service does not validate that the fields in the publishable document are actually generated by the notification.

The resource domains registered by the polling notification templates are set in the polling notification's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Input Parameters

Name	Description
<i>notificationName</i>	String . Required. Name of the polling notification in the format: <i>folder:node</i> .
<i>packageName</i>	String . Required. Package the polling notification is installed.
<i>connectionAlias</i>	String . Required. Name of the connection in the format: <i>folder:node</i> .
<i>notificationTemplate</i>	String . Required. Fully qualified pathname of the polling notification template class.
<i>notificationSettings</i>	IData . Required. Structure for passing the polling notification's property values.
<i>systemName</i>	<p>String. Required. Internal name of the property.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <i>char</i>, <i>short</i>, <i>int</i>, <i>long</i>, <i>float</i>, <i>double</i>, <i>boolean</i>, <i>java.lang.String</i>, <i>java.lang.Character</i>, <i>java.lang.Short</i>, <i>java.lang.Integer</i>, <i>java.lang.Long</i>, <i>java.lang.Float</i>, <i>java.lang.Double</i>, <i>java.lang.Boolean</i>. ■ Arrays of all the above data types are also supported.

Name	Description
<i>scheduleSettings</i>	IData . Required. Structure for specifying the scheduling properties.
<i>notificationInterval</i>	Integer . Frequency to poll in seconds.
<i>notificationOverlap</i>	Boolean . Specifies whether notifications can overlap.
<i>notificationImmediate</i>	Boolean . Specifies whether to invoke the notification immediately.
<i>clusterProperties[k]</i>	IData[] . A <i>k</i> -dimensioned array of cluster properties.
<i>systemName</i>	String . Required. Internal property name
<i>parameterType</i>	String . Required. Data type of the property
<i>publishableRecordDef</i>	IData . Required. Structure for specifying the definition of the runtime publishable output document.
<i>fieldNames</i>	String[] . Names of the fields used in the polling notification's output document.
<i>fieldTypes</i>	String[] . Data types of the fields used in the polling notification's output document.
<i>jmsSettings</i>	IData . Structure specifying the JMS setting.
<i>isJMSConfigured</i>	Boolean . Specifies whether the document is published to a JMS provider.
<i>ConnectionAliasName</i>	String . Required. Connection name of the JMS provider. This field is relevant if the value of isJMSConfigured is true.
<i>DestinationName</i>	String . Required. Name of the destination where the document is published. This field is relevant if the value of isJMSConfigured is true.
<i>DestinationType</i>	<p>String. Required. Type of destination where the document is published. Possible values are:</p> <ul style="list-style-type: none"> ■ Queue ■ Topic <p>This field is relevant if the value of isJMSConfigured is true.</p>

Output Parameters

None.

Example

You must construct *notificationSettings* to create a polling notification node. The value of a property's *systemName* is the internal name of the property. When constructing the *notificationSettings* input parameter, you must use this internal name as the key for setting a property's value. For example, if a polling notification template defines a property named *sqlCommand*, then its *systemName* as returned by *fetchPollingNotificationTemplateMetadata* service is *sqlCommand*. If the caller is a Java application, it might then use this information to set this property's value as follows:

```
IData pipeline = IDataFactory.create();
IDataCursor pipeCursor = pipeline.getCursor();
.
.
.
IData ntfySettings = IDataFactory.create();
IDataCursor ntfyCursor = ntfySettings.getCursor();
ntfyCursor.insertAfter("sqlCommand", "SELECT * FROM fooTable");
.
.
.
pipeCursor.insertAfter("notificationSettings", ntfySettings);
.
.
.
```

In this example, the *sqlCommand* property takes a `java.lang.String` value.

wm.art.dev.notification:deletePollingNotificationNode

This service deletes the specified polling notification node. You must disable the notification before deleting it, using the *disablePollingNotificationNode* service. For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

- You must disable the polling notification node before you delete it, using the `pub.art.notification:disablePollingNotificationNode` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

- The delete action is immediate and non-reversible, and returns no output data.
- Set the *cascadeDelete* flag to propagate the deletion across Broker.
- You may assume that the service completed successfully if it does not throw a checked exception.

This action is immediate and non-reversible, and returns no output data. You may assume that the service completed successfully if it does not throw a checked exception.

Input Parameters

Name	Description
<i>notificationName</i>	String . Required. Name of the polling notification in the format: <code>folder:node</code> .
<i>cascadeDelete</i>	Boolean . Possible values are: <ul style="list-style-type: none"> ■ <code>true</code>. Propagates the deletion across the Broker. ■ <code>false</code>. Default.

Output Parameters

None.

wm.art.dev.notification:fetchPollingNotificationTemplateMetadata

This service fetches all metadata supported by the polling notification template, and the scheduling properties for a specified connection alias and the polling notification template class path.

This service takes a valid connection alias and polling notification template class path, and returns an array of *templateProperties* and an array of *scheduleProperties* structure. Each polling notification template properties (*templateProperties* structure) includes *systemName*, *parameterType*, *defaultValue*, and *isRequired* attributes, which you can use to configure a new polling notification instance.

Input Parameters

Name	Description
<i>connectionAlias</i>	String . Required. Namespace name of an existing connection in the format: <code>folder:node</code> .
<i>notificationTemplate</i>	String . Required. Fully qualified pathname of the polling notification template class.

Output Parameters

Name	Description
<i>description</i>	String . Required. Polling notification template description. Same as the value returned by <code>WmDescriptor.getDisplayName</code> method.
<i>displayName</i>	String . Required. Polling notification template name displayed. Same as the value returned by <code>WmDescriptor.getDescription</code> method.
<i>templateURL</i>	String . Required. URL of the online help page for the polling notification.

Name	Description
<i>indexMaps[i]</i>	IData[] . Required. An <i>i</i> -dimensioned array of field maps.
<i>mapName</i>	String . Field map name.
<i>isVariable</i>	Boolean . Specifies whether the field map is variable length.
<i>disableAppendAll</i>	Boolean . Disables all the buttons used for appending the rows for a field map in the Adapter Service Editor.
<i>templateProperties[n]</i>	IData . Required. An <i>n</i> -dimensioned array of properties.
<i>systemName</i>	String Required. Internal property name.
<i>displayName</i>	String Required. External property name displayed.
<i>description</i>	String Required. Description of the property.
<i>parameterType</i>	<p>String Required. Data type of property.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean. ■ Arrays are not supported.
<i>groupURL</i>	String . URL of the group's help page.
<i>groupName</i>	String . Name of the group to which the property belongs.
<i>tupleName</i>	String . Name of the tuple to which the property belongs.
<i>treeName</i>	String . Name of the tree to which the property belongs.
<i>treeDelimiter</i>	String . Delimiter character used in the tree.
<i>resourceDomain</i>	String . Resource domain name to which the property belongs.
<i>treeView</i>	String . Property name for which values are displayed in the tree structure for selection.
<i>defaultValue</i>	String . Default value of the property.

Name	Description
<i>isRequired</i>	Boolean. Specifies whether the property is required.
<i>isHidden</i>	Boolean. Specifies whether the property is displayable.
<i>isReadOnly</i>	Boolean. Specifies whether the property can be modified.
<i>isFill</i>	Boolean. Specifies whether the editors should pre-fill the property.
<i>isPassword</i>	Boolean. Specifies whether the property is a password.
<i>isMultiline</i>	Boolean. Specifies whether the property traverses multiple lines.
<i>isKey</i>	Boolean. Specifies whether the property is a key field.
<i>minSeqLength</i>	String. Lower bound of the sequence.
<i>maxSeqLength</i>	String. Upper bound of the sequence.
<i>minStringLength</i>	String. Minimum string length.
<i>maxStringLength</i>	String. Maximum string length.
<i>useParam</i>	String. Specifies whether the property is available for use.
<i>scheduleProperties[j]</i>	IData[]. A <i>j</i> -dimensioned array of scheduling properties.
<i>systemName</i>	String. Required. Internal property name
<i>parameterType</i>	String. Required. Data type of the property
<i>clusterProperties[k]</i>	IData[]. A <i>k</i> -dimensioned array of cluster properties.
<i>systemName</i>	String. Required. Internal property name
<i>parameterType</i>	String. Required. Data type of the property

wm.art.dev.notification:updatePollingNotificationNode

This service updates the values of an existing polling notification. You must perform the following:

- You must disable the polling notification before updating its properties, using the `pub.art.notification:disablePollingNotification` service.
- You can modify the following items:

- Underlying connection resource used by the notification
- Metadata property values
- Schedule
- Signature of its publishable output document

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.notification.fetchPollingNotificationTemplateMetadata` service to identify the supported service template properties and configure the `connectionAlias`, `notificationSettings`, `scheduleSettings`, and `publishableRecordDef` input parameter.
2. Configure the `connectionAlias`, `notificationSettings`, `scheduleSettings` input parameters.
3. Set only those properties that you want to change in `connectionAlias`, `notificationSettings`, `scheduleSettings` input parameters.

This service attempts to overlay these new values on the polling notification's current property values. The resulting set of merged property values are used to reconfigure the polling notification. For instance, change the connection resource that the polling notification uses by providing a new `connectionAlias` input parameter. If you omit this parameter, the polling notification continues to use the current connection resource. If you are changing only the connection resource, it is not necessary to provide the `notificationSettings`, `scheduleSettings`, or `publishableRecordDef` input parameters.

4. If you are providing explicit property values in `notificationSettings` parameter, then the values must conform to the underlying data types of those properties.
5. Configuring the `publishableRecordDef`.

- The `publishableRecordDef` input parameter must be replaced in its entirety. The service does not attempt to merge the values into the current publishable record definition. .
- The `fieldNames` and `fieldTypes` properties of `publishableRecordDef` are an array of String. A one-to-one correspondence exists between the elements in these two arrays.
- The values assigned to the `fieldNames` and `fieldTypes` properties of `publishableRecordDef` must correspond to fields that the notification class and the invoked service use to communicate with each other.

Note:

This service cannot verify this assertion. If you configure either record definition incorrectly, the consequence may not manifest until runtime.

6. The value of the `systemName` metadata attribute is the internal name of a property. Use this internal name as the key for setting a property's value when constructing the input parameter `notificationSettings`.
7. Update the property depending on the data type.
 - If a property's data type is non-primitive (derived from `java.lang.Object`), you may clear a property's current value by setting its value to `null`.

- If a property's data type is Java primitive, then the property value cannot be cleared; however the property can be updated.

The resource domains registered by the polling notification templates are set in the polling notification's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Input Parameters

Name	Description
<i>notificationName</i>	String . Required. Namespace name of the existing polling notification in the format: <code>folder:node</code> .
<i>connectionAlias</i>	String . Required. Namespace name of the connection in the format: <code>folder:node</code> .
<i>notificationSettings</i>	IData . Required. Structure for passing the polling notification's property values.
<i>systemName</i>	<p>String. Required. Internal property name.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. ■ Arrays of all the above data types are also supported.
<i>scheduleSettings</i>	IData . Required. Specifies the scheduling properties.
<i>notificationInterval</i>	Integer . Frequency to poll in seconds.
<i>notificationOverlap</i>	Boolean . Specifies whether notifications can overlap.
<i>notificationImmediate</i>	Boolean . Specifies whether to invoke the notification immediately.
<i>clusterProperties[k]</i>	IData[] . A <i>k</i> -dimensioned array of cluster properties.
<i>systemName</i>	String . Required. Internal property name
<i>parameterType</i>	String . Required. Data type of the property

Name	Description
<i>publishableRecordDef</i>	IData . Required. Specifies the definition of the runtime publishable output document.
<i>fieldNames</i>	String[] . Names of the fields used in the polling notification's output document.
<i>fieldTypes</i>	String[] . Data types of the fields used in the polling notification's output document. <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean. ■ Arrays of all the above data types are also supported.

Output Parameters

None.

Connection Services

These services are located in the WmART.wm.art.dev.connection package.

Service	Description
wm.art.dev.connection:createConnectionNode	Creates a new connection node in the specified package and folder.
wm.art.dev.connection:deleteConnectionNode	Removes the specified connection node.
wm.art.dev.connection:fetchConnectionManagerMetadata	Returns the connection manager metadata properties that are predefined for all connections.
wm.art.dev.connection:fetchConnectionMetadata	Queries the connection factory and returns the metadata for all properties supported by connections of the specified type.
wm.art.dev.connection:updateConnectionNode	Alters the values of an existing connection.

wm.art.dev.connection:createConnectionNode

This service creates a new connection node in the specified package and folder, and initializes the connection in a disabled state.

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.connection:fetchConnectionManagerMetadata` service to identify the supported connection manager properties and configure the `connectionManagerSettings` input parameter.
 - All connection manager properties have default values.
 - The default value is set in the `defaultValue` metadata attribute.
 - You can use the default values or override them with values that conform to the underlying data types of the properties.
 - The property values are not automatically set to their default values; you must explicitly set all the required properties.
 - If you omit a property, then no attempt is made to locate and assign default values to the property.
 - The connection manager properties may be optional or a required property.
 - A required property is identified when the `isRequired` metadata attribute is set to `true`.
 - The absence of the `isRequired` attribute implies that the property is not required.
 - If you fail to set a required property, the `wm.art.dev.connection:createConnectionNode` service throws an exception.
 - You may have properties that might be required, based on the current value of some other property.
 - The `poolable` connection manager property is always required.
 - The remaining connection manager properties depend on the value of `poolable` property.
 - If `poolable` is set to `true`, then the remaining connection manager properties must be assigned values as well.
 - If `poolable` is set to `false`, you may omit the remaining connection manager properties.
2. Use `wm.art.dev.connection:fetchConnectionMetadata` service to identify the connection-specific properties and configure the `connectionSettings` input parameter.
 - Connection-specific properties may or may not have default values, depending on the specific adapter's implementation.
 - Depending on the underlying data type of the property, it might not be possible to assign a default value to a connection specific property.

- You may use the default values or override them with values that conform to the underlying data types of the properties.
 - The property values are not automatically set to their default values; you must explicitly set all the required properties.
 - If you omit a property, then no attempt is made to locate and assign default values to the property.
 - Connection-specific properties may be optional or a required property.
 - A required property is identified when the *isRequired* metadata attribute is set to `true`.
 - The absence of the *isRequired* attribute implies that the property is not required.
 - If you fail to set a required property, the `wm.art.dev.connection:createConnectionNode` service throws an exception.
 - You may have properties that might be required, based on the current value of some other property.
 - The number of properties to be configured is adapter-dependent. At a minimum, set those properties whose *isRequired* attribute is set to `true`.
3. Enable the connection using the `pub.art.connection:enableConnection` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

The resource domains registered by the connection factories are set in the connection's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Input Parameters

Name	Description
<i>connectionAlias</i>	String . Required. Name of the connection in the format: <code>folder:node</code> .
<i>packageName</i>	String . Required. Package where the connection is installed.
<i>adapterTypeName</i>	String . Required. Name of the adapter. Same as the value returned by calling <code>WmAdapter.getAdapterName</code> method.
<i>connectionFactoryType</i>	String . Required. Fully qualified path of the connection factory implementation class.
<i>connectionManagerSettings</i>	IData . Required. Structure for passing connection's manager property values. The connection's manager property are predefined for all connections

Name	Description
<i>poolable</i>	<p>Boolean. Required. Determines whether to pool the connection.</p> <p>Note: If <i>poolable</i> is <code>false</code>, then the following property values are not used:</p> <ul style="list-style-type: none"> ■ <i>minimumPoolSize</i> ■ <i>maximumPoolSize</i> ■ <i>poolIncrementSize</i> ■ <i>blockingTimeout</i> ■ <i>expireTimeout</i>
<i>minimumPoolSize</i>	Integer. Minimum number of connections retained in the pool.
<i>maximumPoolSize</i>	Integer. Maximum number of connections retained in the pool.
<i>poolIncrementSize</i>	Integer. Number of connections to add to the pool when additional connections are needed without exceeding the <i>maximumPoolSize</i> value.
<i>blockingTimeout</i>	Integer. Milliseconds to wait for a connection.
<i>expireTimeout</i>	Integer. Milliseconds of inactivity that may elapse prior to destroying the connection.
<i>connectionSettings</i>	IData Required. Structure for passing a connection's property values. The actual connection properties and their underlying data types vary from adapter to adapter. You must set a connection property's value in accordance with its data type. To determine the value, call the service <code>wm.art.dev.connection:fetchConnectionMetadata</code> and note the <i>parameterType</i> attribute for that property.
<i>systemName</i>	String. Internal name of the property.

For more information about the predefined connection manager properties, see [wm.art.dev.connection:fetchConnectionManagerMetadata](#)

For more information about the adapter-specific connection properties, see [wm.art.dev.connection:fetchConnectionMetadata](#)

Output Parameters

None.

Example

You must construct *connectionManagerSettings* and *connectionSettings* to create a connection node. The value of property's *systemName* is the internal name of the property. When constructing the *connectionSettings* input parameters, use this internal name as the key for setting a property's value. For example, if a connection defines a property named *hostPort*, then its *systemName* returned by `wm.art.dev.connection:fetchConnectionMetadata` service is *hostPort*. If the caller is a Java application, it might then use this information to set this property's value as follows:

```
IData pipeline = IDataFactory.create();
IDataCursor pipeCursor = pipeline.getCursor();
.
.
.
IData connSettings = IDataFactory.create();
IDataCursor connCursor = connSettings.getCursor();
connCursor.insertAfter("hostPort", new Integer(1234));
.
.
.
pipeCursor.insertAfter("connectionSettings", connSettings);
.
.
.
```

In this example, the *hostPort* property takes a `java.lang.Integer` value.

wm.art.dev.connection:deleteConnectionNode

This service deletes the specified connection node.

- You must disable the connection node before you delete it, using the `pub.art.connection:disableConnection` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

- The delete action is immediate and non-reversible, and returns no output data.
- You may assume that the service completed successfully if it does not throw a checked exception.

Input Parameters

Name	Description
<i>connectionAlias</i>	String . Required. Name of the connection in the format: <code>folder:node</code> .

Output Parameters

None.

wm.art.dev.connection:fetchConnectionManagerMetadata

This service fetches the connection manager properties that are predefined for all the connections, such as *poolable*, *minimumPoolSize*, *maximumPoolSize*, and others.

This service receives no inputs and returns an array of *connectionManagerProperties* structure. The *connectionManagerProperties* structure contains all metadata associated with each of the connection manager properties such as *systemName*, *parameterType*, *defaultValue*, and *isRequired* attributes, which you can use to configure a connection node.

Input Parameters

None.

Output Parameters

Name	Description
<i>connectionManagerProperties[n]</i>	IData . Required. An <i>n</i> -dimensioned array of connection manager properties.
<i>systemName</i>	<p>String. Required. Internal property name. Values are:</p> <ul style="list-style-type: none"> ■ <i>poolable</i> ■ <i>minimumPoolSize</i> ■ <i>maximumPoolSize</i> ■ <i>poolIncrementSize</i> ■ <i>blockingTimeout</i> ■ <i>expireTimeout</i>
<i>displayName</i>	String . Required. External property name displayed.
<i>description</i>	String . Required. Description of the property.
<i>parameterType</i>	<p>String. Required. Data type of the property.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <i>char</i>, <i>short</i>, <i>int</i>, <i>long</i>, <i>float</i>, <i>double</i>, <i>boolean</i>, <i>java.lang.String</i>, <i>java.lang.Character</i>, <i>java.lang.Short</i>, <i>java.lang.Integer</i>, <i>java.lang.Long</i>, <i>java.lang.Float</i>, <i>java.lang.Double</i>, <i>java.lang.Boolean</i>.

Name	Description
	<ul style="list-style-type: none"> ■ Arrays are not supported.
<i>groupURL</i>	String. Not applicable to connection manager properties.
<i>groupName</i>	String. Not applicable to connection manager properties.
<i>tupleName</i>	String. Not applicable to connection manager properties.
<i>treeName</i>	String. Not applicable to connection manager properties.
<i>treeDelimiter</i>	String. Not applicable to connection manager properties.
<i>resourceDomain</i>	String. Not applicable to connection manager properties.
<i>defaultValue</i>	String. Default value of the property.
<i>isRequired</i>	Boolean. Specifies whether the property is required.

wm.art.dev.connection:fetchConnectionMetadata

This service fetches the adapter-specific properties.

This service receives an adapter type name and a fully qualified connection factory class path, queries the connection factory and returns the metadata for all properties supported by the connections of that type.

Input Parameters

Name	Description
<i>adapterTypeName</i>	String. Required. Name of adapter. Same as the value returned by <code>WmAdapter.getAdapterName</code> method.
<i>connectionFactoryType</i>	String. Required. Fully qualified path of the connection factory implementation class.

Output Parameters

The service returns a *connectionProperties* array containing all metadata associated with each of the connection properties. You can use the following attributes to configure a connection node: *systemName*, *parameterType*, *defaultValue*, and *isRequired*.

Name	Description
<i>displayName</i>	String . Required. Adapter specific property name. Same as the value returned by <code>WmDescriptor.getDisplayName</code> method.
<i>description</i>	String . Required. Adapter specific property description. Same as the value returned by <code>WmDescriptor.getDescription</code> method.
<i>templateURL</i>	String . Required. URL of online help page for the connection.
<i>connectionProperties[n]</i>	IData[] . Required. An <i>n</i> -dimensioned array of properties.
<i>systemName</i>	String . Required. Adapter specific internal property name.
<i>displayName</i>	String . Required. External property name displayed.
<i>description</i>	String . Required. Description of the property.
<i>parameterType</i>	<p>String. Required. Data type of the property.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. ■ Arrays are not supported.
<i>groupURL</i>	String . URL of the group's help page.
<i>groupName</i>	String . Name of the group to which the property belongs.
<i>tupleName</i>	String . Name of the tuple to which the property belongs.
<i>treeName</i>	String . Name of the tree to which the property belongs.
<i>treeDelimiter</i>	String . Delimiter character used in the tree.
<i>resourceDomain</i>	String . Resource domain name to which the property belongs.
<i>defaultValue</i>	String . Default property value.
<i>isRequired</i>	Boolean . Specifies whether the property is required.

Name	Description
<i>isHidden</i>	Boolean. Specifies whether the property is displayable.
<i>isReadOnly</i>	Boolean. Specifies whether the property can be modified.
<i>isFill</i>	Boolean. Specifies whether the editors must pre-fill the property.
<i>isPassword</i>	Boolean. Specifies whether the property is a password.
<i>isMultiline</i>	Boolean. Specifies whether the property traverses multiple lines.
<i>isKey</i>	Boolean. Specifies whether the property is a key field.
<i>minSeqLength</i>	String. Lower bound of the sequence.
<i>minStringLength</i>	String. Minimum string length.
<i>maxStringLength</i>	String. Maximum string length.
<i>useParam</i>	String. Specifies whether the property is available for use.

Example

For example, a Java client might invoke this service as follows:

```
IData pipeline = IDataFactory.create();
IDataCursor pipeCursor = pipeline.getCursor();
pipeCursor.insertAfter("adapterTypeName",
    "FooAdapter");
pipeCursor.insertAfter("connectionFactoryType",
    "com.wm.adapters.FooConnFactory");
ExtendedConnectionUtils.fetchConnectionMetadata(pipeline);
.
.
.
```

wm.art.dev.connection:updateConnectionNode

This service updates the values of an existing connection.

- You must disable the connection node before you update it, using the `pub.art.connection:disableConnection` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.connection:fetchConnectionManagerMetadata` service to identify the supported connection manager properties and configure the `connectionManagerSettings` input parameter.
2. Use `wm.art.dev.connection:fetchConnectionMetadata` service to identify the connection-specific properties and configure the `connectionSettings` input parameter.
3. Provide values for the properties you want to change.

This service attempts to overlay these new values on the connection's current property values. The resulting set of merged property values are used to reconfigure the connection.

If you are not changing any connection manager or connection-specific properties, it is not necessary to pass in that container parameter. For example, if you are not changing any connection manager properties, you must not build and pass in the `connectionManagerSettings` parameter.

4. If you are providing explicit property values in `connectionManagerSettings` and `connectionSettings` parameter, then the values must conform to the underlying data types of those properties.

For an example of setting a connection property, see the Java code example in [wm.art.dev.connection:createConnectionNode](#).

Input Parameters

Name	Description
<code>connectionAlias</code>	String . Required. Name of the connection in the format: <code>folder:node</code> .
<code>connectionManagerSettings</code>	IData . Required. Structure for passing connection's manager property values. The connection's manager property are predefined for all connections.
<code>poolable</code>	<p>Boolean. Required. Determines whether to pool the connection.</p> <p>Note: If <code>poolable</code> is <code>false</code>, then the following property values are not used:</p> <ul style="list-style-type: none"> ■ <code>minimumPoolSize</code> ■ <code>maximumPoolSize</code> ■ <code>poolIncrementSize</code> ■ <code>blockingTimeout</code> ■ <code>expireTimeout</code>
<code>minimumPoolSize</code>	Integer . Minimum number of connections retained in the pool.
<code>maximumPoolSize</code>	Integer . Maximum number of connections retained in the pool.

Name	Description
	<i>poolIncrementSize</i> Integer. Number of connections to add to the pool when additional connections are needed and must not exceed <i>maximumPoolSize</i> .
	<i>blockingTimeout</i> Integer. Milliseconds to wait for a connection.
	<i>expireTimeout</i> Integer. Milliseconds of inactivity that may elapse prior to destroying the connection.
<i>connectionSettings</i>	IData. Required. Structure for passing a connection's property values. The actual connection properties and their underlying data types vary from adapter to adapter. You must set a connection property's value in accordance with its data type. To determine the value, call the service <code>wm.art.dev.connection:fetchConnectionMetadata</code> and note the <i>parameterType</i> attribute for that property.
	<i>systemName</i> String. Internal name of the property.

For more information about the predefined connection manager properties, see [wm.art.dev.connection:fetchConnectionManagerMetadata](#)

For more information about the adapter-specific connection properties, see [wm.art.dev.connection:fetchConnectionMetadata](#)

Output Parameters

None.

Adapter Service Services

These services are located in the `WmART.wm.art.dev.service` package.

Service	Description
wm.art.dev.service:createAdapterServiceNode	Creates a new adapter service node in the specified package and folder from the specified service template and connection alias.
wm.art.dev.service:deleteAdapterServiceNode	Removes a specified adapter service node.
wm.art.dev.service:fetchAdapterServiceTemplateMetadata	Returns all metadata for a specified adapter service template.
wm.art.dev.service:updateAdapterServiceNode	Alters the values of an existing adapter service.

wm.art.dev.service:createAdapterServiceNode

This service creates a new adapter service node in the specified package and folder from the specified service template and connection alias.

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.service:fetchAdapterServiceTemplateMetadata` service to identify the supported service template properties and configure the `adapterServiceSettings` input parameter.
 - The service's `inputFieldNames`, `inputFieldTypes`, `outputFieldNames`, and `outputFieldTypes` parameters in the `adapterServiceSettings` structure define the properties that comprise the adapter service's input and output signatures.
 - The data types of properties in the `adapterServiceSettings` structure are arrays of `java.lang.String` type.
 - A one-to-one correspondence exists between the elements in the `*FieldNames` and `*FieldTypes` arrays. For example, if the property names `abc`, `xyz`, and `foo` are inserted into the `outputFieldNames` parameter, then the service expects that exactly three data types will be inserted into `outputFieldTypes`, and that those data types correspond to the same element in `outputFieldNames`.
 - Adapter service properties may or may not have default values, depending on the specific adapter's implementation.
 - Depending on the underlying data type of the property, it might not be possible to assign a default value to a property.
 - You may use the default values or override them with values that conform to the underlying data types of the properties.
 - The property values are not automatically set to their default values; you must explicitly set all the required properties.
 - If you omit a property, then no attempt is made to locate and assign default values to the property.
 - Adapter service properties may be optional or a required property.
 - A required property is identified when the `isRequired` metadata attribute is set to `true`.
 - The absence of the `isRequired` attribute implies that the property is not required.
 - If you fail to set a required property, the `wm.art.dev.service:createAdapterServiceNode` service throws an exception.
 - You may have properties that might be required, based on the current value of some other property.
 - The number of properties to be configured is adapter-dependent. At a minimum, set those properties whose `isRequired` attribute is set to `true`.

The resource domains registered by the adapter service template are set in the adapter service's properties according to the interdependencies between the resource domains. This includes input and output signatures since they are supported through resource domains. This service provides the properties *inputFieldNames*, *inputFieldTypes*, *outputFieldNames*, and *outputFieldTypes* for this purpose. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Input Parameters

Name	Description
<i>serviceName</i>	String . Required. Namespace name of the new adapter service in the format: <code>folder:node</code> .
<i>packageName</i>	String . Required. Package in which to install the adapter service.
<i>connectionAlias</i>	String . Required. Namespace name of the connection in the format: <code>folder:node</code> .
<i>serviceTemplate</i>	String . Required. Fully qualified pathname of the adapter service template class.
<i>adapterServiceSettings</i>	IData . Required. Structure for passing the adapter's property values.
<i>systemName</i>	String . Required. Adapter service specific internal property name.
<i>inputFieldNames</i>	String[] . Names of the fields used in the adapter's input signature.
<i>inputFieldTypes</i>	String[] . Data types of the fields used in the adapter's input signature.
	<p>Note:</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. ■ Arrays are supported.
<i>outputFieldNames</i>	String[] . Names of the fields used in the adapter's output signature.
<i>outputFieldTypes</i>	String[] . Data types of the fields used in the adapter's output signature.
	<p>Note:</p>

Name	Description
	<ul style="list-style-type: none"> ■ The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean. ■ Arrays are supported.

Output Parameters

None.

Example

You must construct *adapterServiceSettings* to create an adapter service node. The value of a property's *systemName* is the internal name of the property. When constructing the input parameter *adapterServiceSettings*, you must use this internal name as the key for setting a property's value. For example, if a service template defines a property named *sqlCommand*, then its *systemName* as returned by *fetchAdapterServiceTemplateMetadata* service is *sqlCommand*. If the caller is a Java application, it might then use this information to set this property's value as follows:

```

IData pipeline = IDataFactory.create();
IDataCursor pipeCursor = pipeline.getCursor();
.
.
.
IData svcSettings = IDataFactory.create();
IDataCursor svcCursor = svcSettings.getCursor();
svcCursor.insertAfter("sqlCommand", "SELECT * FROM fooTable");
.
.
.
pipeCursor.insertAfter("adapterServiceSettings", svcSettings);
.
.
.

```

In this example, the *sqlCommand* property takes a `java.lang.String` value.

wm.art.dev.service:deleteAdapterServiceNode

This service deletes the specified adapter service node.

- The delete action is immediate and non-reversible, and returns no output data.
- You may assume that the service completed successfully if it does not throw a checked exception.

Input Parameters

Name	Description
<i>serviceName</i>	String . Required. Name of the adapter service in the format: <code>folder:node</code> .

Output Parameters

None.

wm.art.dev.service:fetchAdapterServiceTemplateMetadata

This service fetches all metadata for an adapter service template for a specified connection alias and adapter service template class path.

This service receives a connection and a fully qualified adapter service template class path, and returns the metadata for all properties supported by the adapter service template.

This service returns an array of *templateProperties* structure each containing attributes such as *systemName*, *parameterType*, *defaultValue*, and *isRequired*.

Input Parameters

Name	Description
<i>connectionAlias</i>	String . Required. Name of the connection in the format: <code>folder:node</code> .
<i>serviceTemplate</i>	String . Required. Fully qualified pathname of the adapter service template class.

Output Parameters

Name	Description
<i>description</i>	String . Required. Adapter service template description. Same as the value returned by <code>WmDescriptor.getDisplayName</code> method.
<i>displayName</i>	String . Required. Adapter service template name displayed. Same as the value returned by <code>WmDescriptor.getDescription</code> method.
<i>templateURL</i>	String . Required. URL of the online help page for the adapter service.
<i>indexMaps[i]</i>	IData[] . Required. An <i>i</i> -dimensioned array of field maps.
<i>mapName</i>	String . Field map name.
<i>isVariable</i>	Boolean . Specifies whether the field map is of variable length.

Name	Description
<i>disableAppendAll</i>	Boolean. Disables all the buttons used for appending the rows for a field map in the Adapter Service Editor.
<i>templateProperties[n]</i>	IData. Required. An <i>n</i> -dimensioned array of properties.
<i>systemName</i>	String Required. Internal property name.
<i>displayName</i>	String Required. External property name.
<i>description</i>	String Required. Description of the property.
<i>parameterType</i>	<p data-bbox="841 642 1347 672">String Required. Data type of property.</p> <ul data-bbox="841 684 1474 982" style="list-style-type: none"> <li data-bbox="841 684 1474 928">■ The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean. <li data-bbox="841 953 1224 982">■ Arrays are not supported.
<i>groupURL</i>	String. URL of the group's help page.
<i>groupName</i>	String. Name of the group to which the property belongs.
<i>tupleName</i>	String. Name of the tuple to which the property belongs.
<i>treeName</i>	String. Name of the tree to which the property belongs.
<i>treeDelimiter</i>	String. Delimiter character used in the tree.
<i>resourceDomain</i>	String. Resource domain name and dependencies.
<i>treeView</i>	String. Property name for which values are displayed in the tree structure for selection.
<i>defaultValue</i>	String. Default value of the property.
<i>isRequired</i>	Boolean. Specifies whether the property is required.
<i>isHidden</i>	Boolean. Specifies whether the property is displayable.
<i>isReadOnly</i>	Boolean. Specifies whether the property can be modified.

Name	Description
<i>isFill</i>	Boolean. Specifies whether the editors must pre-fill the property.
<i>isPassword</i>	Boolean. Specifies whether the property is a password.
<i>isMultiline</i>	Boolean. Specifies whether the property traverses multiple lines.
<i>isKey</i>	Boolean. Specifies whether the property is a key field.
<i>minSeqLength</i>	String. Lower bound of the sequence.
<i>maxSeqLength</i>	String. Upper bound of the sequence.
<i>minStringLength</i>	String. Minimum string length.
<i>maxStringLength</i>	String. Maximum string length.
<i>useParam</i>	String. Specifies whether the property is available for use.

wm.art.dev.service:updateAdapterServiceNode

This service updates an existing adapter service node.

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.service:fetchAdapterServiceTemplateMetadata` service to identify the supported service template properties and configure the `adapterServiceSettings` input parameter.

- The value of the `systemName` metadata attribute is the internal name of a property. Use this internal name as the key for setting a property's value when constructing the `adapterServiceSettings` input parameter.

2. Set only those properties that you want to change.

This service attempts to overlay these new values on the adapter service's current property values. The resulting set of merged property values are used to reconfigure the adapter service.

3. Update the properties depending on the data type.

- If a property's data type is non-primitive (derived from `java.lang.Object`), you may clear a property's current value by setting its value to `null`.
- If a property's data type is Java primitive, then the property value cannot be cleared; however the property can be updated.

4. If you are providing explicit property values in the `adapterServiceSettings` parameter, then the values must conform to the underlying data types of those properties.

5. Change the connection resource that the adapter service uses by providing a new *connectionAlias* input parameter.
 - If you omit the *connectionAlias* parameter, the adapter service will continue to use its current connection resource.
 - If you are changing only the connection resource, it is not necessary to provide the *adapterServiceSettings* input parameter.

The resource domains registered by the adapter service template are set in the adapter service's properties according to the interdependencies between the resource domains. This includes input and output signatures since they are supported through resource domains. This service provides the properties *inputFieldNames*, *inputFieldTypes*, *outputFieldNames*, and *outputFieldTypes* for this purpose. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

For an example of setting an adapter service property, see the Java code example in [wm.art.dev.service:createAdapterServiceNode](#).

Input Parameters

Name	Description
<i>serviceName</i>	String . Required. Name of the existing adapter service in the format: <code>folder:node</code> .
<i>connectionAlias</i>	String . Required. Name of the connection in the format: <code>folder:node</code> .
<i>adapterServiceSettings</i>	IData . Required. Structure for passing the adapter's property values.
<i>systemName</i>	String . Required. Adapter service specific internal property name.
<i>inputFieldNames</i>	String[] . Names of the fields used in the adapter's input signature.
<i>inputFieldTypes</i>	String[] . Data types of the fields used in the adapter's input signature. <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. ■ Arrays are supported.
<i>outputFieldNames</i>	String[] . Names of the fields used in the adapter's output signature.

Name	Description
<i>outputFieldTypes</i>	<p>String[]. Data types of the fields used in the adapter's output signature.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean. ■ Arrays are supported.

Output Parameters

None.

Listener Services

These services are located in the WmART.wm.art.dev.listener package.

Service	Description
wm.art.dev.listener:analyzeListenerNodes	Logs the data for listeners.
wm.art.dev.listener:createListenerNode	Creates a new instance of a listener in the specified package and folder from the specified listener template and connection alias.
wm.art.dev.listener:deleteListenerNode	Removes a specified instance of a listener.
wm.art.dev.listener:fetchListenerTemplateMetadata	Returns all metadata supported by that listener template.
wm.art.dev.listener:updateListenerNode	Alters the values of an existing listener.
wm.art.dev.listener:updateRegisteredNotifications	Checks the registration of listener notifications.

wm.art.dev.listener:analyzeListenerNodes

This service logs the data for listeners in the server log file. The data includes the names of associated listener notifications, the class name of listener notifications, their status (active or disabled), and whether the associated listener notification is linked with the same listener or not.

Note:

A listener can be used by multiple listener notifications, but a listener notification will have only one listener node.

Input Parameters

None.

Output Parameters

None.

wm.art.dev.listener:createListenerNode

This service creates a new listener node in the specified package and folder from the specified listener template and connection alias, and initializes the new listener in a disabled state.

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.service.fetchListenerTemplateMetadata` service to identify the supported listener template properties and configure the `listenerSettings` input parameter.
 - Listener properties may or may not have default values, depending on the specific adapter's implementation.
 - Depending on the underlying data type of the property, it might not be possible to assign a default value to a property.
 - You may use the default values or override them with values that conform to the underlying data types of the properties.
 - The property values are not automatically set to their default values; you must explicitly set all the required properties.
 - If you omit a property, then no attempt is made to locate and assign default values to the property.
 - Listener properties may be optional or a required property.
 - A required property is identified when the `isRequired` metadata attribute is set to `true`.
 - The absence of the `isRequired` attribute implies that the property is not required.
 - If you fail to set a required property, the `wm.art.dev.listener:createListenerNode` service throws an exception.
 - You may have properties that might be required, based on the current value of some other property.
 - The number of properties to be configured is adapter-dependent. At a minimum, set those properties whose `isRequired` attribute is set to `true`.
2. Activate the listener using the `pub.art.listener.enableListener` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

The resource domains registered by the listener service template are set in the listener's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Input Parameters

Name	Description						
<i>listenerName</i>	String . Required. Name of the listener in the format: <code>folder:node</code> .						
<i>packageName</i>	String . Required. Package where the listener is installed.						
<i>connectionAlias</i>	String . Required. Name of the connection in the format: <code>folder:node</code> .						
<i>listenerTemplate</i>	String . Required. Fully qualified pathname of the listener template class.						
<i>listenerSettings</i>	IData . Required. Structure for passing the listener's property values. <table><tbody><tr><td><i>systemName</i></td><td>String. Required. Internal name of the property.<ul style="list-style-type: none">■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>.</td></tr><tr><td><i>retryLimit</i></td><td>String. Required. Number of times that the system must attempt to start the listener if the initial attempt fails.</td></tr><tr><td><i>retryBackoffTimeout</i></td><td>String. Required. Number of seconds the system must wait between each attempt to start the listener. This field is irrelevant if the value of Retry Limit is 0.</td></tr></tbody></table>	<i>systemName</i>	String . Required. Internal name of the property. <ul style="list-style-type: none">■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>.	<i>retryLimit</i>	String . Required. Number of times that the system must attempt to start the listener if the initial attempt fails.	<i>retryBackoffTimeout</i>	String . Required. Number of seconds the system must wait between each attempt to start the listener. This field is irrelevant if the value of Retry Limit is 0.
<i>systemName</i>	String . Required. Internal name of the property. <ul style="list-style-type: none">■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>.						
<i>retryLimit</i>	String . Required. Number of times that the system must attempt to start the listener if the initial attempt fails.						
<i>retryBackoffTimeout</i>	String . Required. Number of seconds the system must wait between each attempt to start the listener. This field is irrelevant if the value of Retry Limit is 0.						

Output Parameters

None.

Example

You must construct *listenerSettings* to create a listener node. The value of a property's *systemName* is the internal name of the property. When constructing the input parameter *listenerSettings*, you must use this internal name as the key for setting a property's value. For example, if a listener template defines a property named *portNumber*, then its *systemName* as returned by `wm.art.dev.listener:fetchListenerTemplateMetadata` service is *portNumber*. If the caller is a Java application, it might then use this information to set this property's value as follows:

```

IData pipeline = IDataFactory.create();
IDataCursor pipeCursor = pipeline.getCursor();
.
.
.
IData lstnrSettings = IDataFactory.create();
IDataCursor lstnrCursor = lstnrSettings.getCursor();
lstnrCursor.insertAfter("portNumber", new Integer((int)8888));
.
.
.
pipeCursor.insertAfter("listenerSettings", lstnrSettings);
.
.
.

```

In this example, the *portNumber* property takes a `java.lang.Integer` value.

wm.art.dev.listener:deleteListenerNode

This service deletes a specified listener node.

- You must disable the listener node before you delete it, using the `pub.art.listener:disableListener` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

- The delete action is immediate and non-reversible, and returns no output data.
- You may assume that the service completed successfully if it does not throw a checked exception.

Input Parameters

Name	Description
<i>listenerName</i>	String . Required. Name of the listener in the format: <code>folder:node</code> .

Output Parameters

None.

wm.art.dev.listener:fetchListenerTemplateMetadata

This service fetches all metadata supported by the listener template for a specified connection alias and listener template class path.

This service takes a valid connection alias and listener template class path, and returns an array of *templateProperties* structure each containing attributes such as *systemName*, *parameterType*, *defaultValue*, and *isRequired*.

Input Parameters

Name	Description
<i>connectionAlias</i>	String . Required. Name of an existing connection in the format: folder:node.
<i>listenerTemplate</i>	String . Required. Fully qualified pathname of listener template class.

Output Parameters

Name	Description
<i>description</i>	String . Required. Listener template description. Same as the value returned by <code>WmDescriptor.getDisplayName</code> method.
<i>displayName</i>	String . Required. Listener template name displayed. Same as the value returned by <code>WmDescriptor.getDescription</code> method.
<i>listAllConnection</i>	String . Default template metadata
<i>templateURL</i>	String . Required. URL of the online help page for the listener.
<i>requiresConnection</i>	Boolean . Specifies whether the listener requires a connection.
<i>indexMaps[i]</i>	IData[] . Required. An <i>i</i> -dimensioned array of field maps.
<i>mapName</i>	String . Field map name.
<i>isVariable</i>	Boolean . Specifies whether the field map is variable length.
<i>disableAppendAll</i>	Boolean . Disables all the buttons used for appending the rows for a field map in the Adapter Service Editor.
<i>templateProperties[n]</i>	IData[] . Required. An <i>n</i> -dimensioned array of properties.
<i>systemName</i>	String . Required. Internal property name.
<i>displayName</i>	String . Required. External displayable property name.
<i>description</i>	String . Required. Description of the property.
<i>parameterType</i>	String . Required. Data type of property. <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>,

Name	Description
	<p>java.lang.Float, java.lang.Double, java.lang.Boolean.</p> <ul style="list-style-type: none"> ■ Arrays are not supported.
<i>groupURL</i>	String. URL of the group's help page.
<i>groupName</i>	String. Name of the group to which the property belongs.
<i>tupleName</i>	String. Name of the tuple to which the property belongs.
<i>treeName</i>	String. Name of the tree to which the property belongs.
<i>treeDelimiter</i>	String. Delimiter character used in the tree.
<i>resourceDomain</i>	String. Resource domain name to which the property belongs.
<i>treeView</i>	String. Property name for which values are displayed in the tree structure for selection.
<i>defaultValue</i>	String. Default value of the property.
<i>isRequired</i>	Boolean. Specifies whether the property is required.
<i>isHidden</i>	Boolean. Specifies whether the property is displayable.
<i>isReadOnly</i>	Boolean. Specifies whether the property can be modified.
<i>isFill</i>	Boolean. Specifies whether the editors should pre-fill the property.
<i>isPassword</i>	Boolean. Specifies whether the property is a password.
<i>isMultiline</i>	Boolean. Specifies whether the property traverses multiple lines.
<i>isKey</i>	Boolean. Specifies whether the property is a key field.
<i>minSeqLength</i>	String. Lower bound of the sequence.
<i>maxSeqLength</i>	String. Upper bound of the sequence.
<i>minStringLength</i>	String. Minimum string length.

Name	Description
<i>maxLength</i>	String. Maximum string length.
<i>useParam</i>	String. Specifies whether the property is available for use.

wm.art.dev.listener:updateListenerNode

This service updates an existing listener node.

- You must disable the listener before updating its properties, using the `pub.art.listener:disableListener` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.listener:fetchListenerTemplateMetadata` service to identify the supported service template properties and configure the *listenerSettings* input parameter.

- The value of the *systemName* metadata attribute is the internal name of a property. Use this internal name as the key for setting a property's value when constructing the input parameter *listenerSettings*.

2. Set only those properties that you want to change.

Note:

This service attempts to overlay these new values on the listener's current property values. The resulting set of merged property values are used to reconfigure the listener.

3. Update the property depending on the data type.
 - If a property's data type is non-primitive (derived from `java.lang.Object`), you may clear a property's current value by setting its value to `null`.
 - If a property's data type is Java primitive, then the property value cannot be cleared; however the property can be updated.
4. If you are providing explicit property values in *listenerSettings* parameter, then the values must conform to the underlying data types of those properties.
5. Change the connection resource that the listener uses by providing a new *connectionAlias* input parameter.
 - If you omit the *connectionAlias* parameter, the listener continues to use its current connection resource.
 - If you are changing only the connection resource, it is not necessary to provide the *listenerSettings* input parameter.

The resource domains registered by the listener templates are set in the listener's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

For an example of setting an listener property, see the Java code example in [wm.art.dev.listener:createListenerNode](#).

Input Parameters

Name	Description
<i>listenerName</i>	String . Required. Name of an existing listener in the format: <code>folder:node</code> .
<i>connectionAlias</i>	String . Required. Name of the connection in the format: <code>folder:node</code> .
<i>listenerSettings</i>	IData . Required. Structure for passing the listener's property values.
<i>systemName</i>	<p>String. Required. Internal property name.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. ■ Arrays are not supported.
<i>retryLimit</i>	String . Required. Number of times that the system must attempt to start the listener if the initial attempt fails.
<i>retryBackoffTimeout</i>	String . Required. Number of seconds the system must wait between each attempt to start the listener. This field is irrelevant if the value of Retry Limit is \emptyset .

Output Parameters

None.

wm.art.dev.listener:updateRegisteredNotifications

This service checks listener notifications, finds the linked listener, and then checks whether the listener notification is also registered for the same listener or not. If not, then the service registers the listener notification with the linked listener.

Note:

A listener can be used by multiple listener notifications, but a listener notification has only one listener node.

Input Parameters

Name	Description
<i>listenerNodeName</i>	<p>String. Specifies the name of the WmART-based listener for which the notifications must be updated.</p> <ul style="list-style-type: none"> ■ Specify the listener node name to update a specific listener. ■ Specify * to update all the WmART-based listeners .

Output Parameters

None.

Listener Notification Services

These services are located in the WmART.wm.art.dev.notification package.

Service	Description
wm.art.dev.notification:analyzeListenerNotifications	Logs the data for listener notifications.
wm.art.dev.notification:createListenerNotificationNode	Creates a new instance of a synchronous or asynchronous listener notification in the specified package and folder from the specified notification template and connection alias.
wm.art.dev.notification:deleteListenerNotificationNode	Removes a specified instance of listener notification.
wm.art.dev.notification:fetchListenerNotificationTemplateMetadata	Returns all metadata for a specified listener notification template.
wm.art.dev.notification:updateListenerNotificationNode	Alters the values of an existing synchronous or asynchronous listener notification.

wm.art.dev.notification:analyzeListenerNotifications

This service logs the data for listener notifications. The data includes the name of a listener to which a listener notification is linked, and the class name of the listener. The service also checks whether the listener notification is registered with the listener or not.

Note:

A listener can be used by multiple listener notifications, but a listener notification will have only one listener node.

Input Parameters

None.

Output Parameters

None.

wm.art.dev.notification:createListenerNotificationNode

This service creates a new synchronous or asynchronous listener notification node in the specified package and folder from the specified notification template and connection alias, and initializes the listener notification node in a disabled state.

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.notification:fetchListenerNotificationTemplateMetadata` service to identify the supported listener notification template properties and configure the `notificationSettings` input parameter.
 - Listener notification properties may or may not have default values, depending on the specific adapter's implementation.
 - Depending on the underlying data type of the property, it might not be possible to assign a default value to a property.
 - You may use the default values or override them with values that conform to the underlying data types of the properties.
 - The property values are not automatically set to their default values; you must explicitly set all the required properties.
 - If you omit a property, then no attempt is made to locate and assign default values to the property.
 - Listener notification properties may be optional or a required property.
 - A required property is identified when the `isRequired` metadata attribute is set to `true`.
 - The absence of the `isRequired` attribute implies that the property is not required.
 - If you fail to set a required property, the `wm.art.dev.notification:createListenerNotificationNode` service throws an exception.
 - You may have properties that might be required, based on the current value of some other property.
 - The number of properties to be configured is adapter-dependent. At a minimum, set those properties whose `isRequired` attribute is set to `true`.
2. The particular combination of input parameters you specify determines whether you create a synchronous or an asynchronous listener notification as described in the section for input

parameters. This service throws an exception if you specify invalid or ambiguous combinations of input parameters.

3. When an asynchronous listener notification is triggered, Integration Server generates a runtime publishable output document.
 - The names and types of the data fields in the publishable output document is predefined.
 - Specify the names and types of the data fields in the publishable output document in the service's *publishableRecordDef* input parameter.
 - The *publishableRecordDef* input parameter consists of *fieldNames* and *fieldTypes* properties.
 - The data types of properties in the *publishableRecordDef* structure are arrays of `java.lang.String` type.
 - The *fieldNames* and *fieldTypes* properties are an array of `String`.
 - A one-to-one correspondence exists between the elements in these *fieldNames* and *fieldTypes* arrays.
 - The values assigned to the *fieldNames* and *fieldTypes* properties must correspond to fields that the listener notification class outputs in its `runNotification` method.
 - The service execution may fail if an empty *publishableRecordDef* is specified.
4. Activate the listener notification node using the `pub.art.notification:enableListenerNotification` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

Note:

1. This service does not validate that the fields in the publishable document are actually generated by the notification.
2. This service creates a publishable document node in a *notificationNamePublishDocument* folder where *notificationName* is the value you specify for the *notificationName* input parameter. The name of publishable document is not configurable.

The resource domains registered by the listener notification templates are set in the listener notification's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Note:

1. Synchronous listener notification classes must extend the `WmSyncListenerNotification` class.
2. Asynchronous listener notifications must extend the `WmAsyncListenerNotification` class.

Input Parameters

Name	Description
<i>notificationName</i>	String . Required. Name of the listener notification in the format: <code>folder:node</code> .

Name	Description
<i>packageName</i>	String. Required. Package where the listener notification is installed.
<i>listenerNode</i>	String. Required. Name of the listener in the format: <code>folder:node</code> .
<i>notificationTemplate</i>	String. Required. Fully qualified pathname of the listener notification template class.
<i>notificationSettings</i>	IData. Required. Structure for passing the listener notification's property values.
<i>systemName</i>	<p>String. Required. Internal name of the property.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. ■ Arrays of all the above data types are also supported.
Properties applicable to synchronous listener notifications only	
<i>serviceName</i>	String. Required. Name of the service to invoke in the format: <code>folder:node</code> . This parameter must not be <code>null</code> .
<i>executionMode</i>	IData. Structure to identify whether a service defined in <i>serviceName</i> is invoked or a document is published.
<i>mode</i>	<p>String. Possible values are:</p> <ul style="list-style-type: none"> ■ <code>publishAndWait</code>. Publish the document and wait for the response. ■ <code>invokeService</code>. Invoke the service defined in <i>serviceName</i> parameter.
<i>local</i>	Boolean. Enables publishing the document locally.
<i>waitTime</i>	String. Time in millisecond to wait for the response in an synchronous listener notification.
<i>jmsSettings</i>	IData. Structure specifying the JMS setting.
<i>isJMSConfigured</i>	Boolean. Specifies whether the document is published to a JMS provider.

Name	Description
	<p><i>ConnectionAliasName</i> String. Required. Connection name of the JMS provider. This field is relevant if the value of isJMSConfigured is true.</p>
	<p><i>DestinationName</i> String. Required. Name of the destination where the document is published. This field is relevant if the value of isJMSConfigured is true.</p>
	<p><i>DestinationType</i> String. Required. Type of destination where the document is published. Possible values are:</p> <ul style="list-style-type: none"> ■ Queue ■ Topic <p>This field is relevant if the value of isJMSConfigured is true.</p>
<i>requestRecordDef</i>	<p>IData. Required. Structure specifying the definition of the request document sent to <i>serviceName</i>. This parameter must not be <code>null</code>, but may be empty. This property contains the following fields:</p>
	<p><i>fieldNames</i> String[]. Names of the fields used in the listener notification's request document.</p>
	<p><i>fieldTypes</i> String[]. Data types of the fields used in the listener notification's request document.</p>
<i>replyRecordDef</i>	<p>IData. Required. Structure specifying the definition of the reply document received from <i>serviceName</i>. This parameter must not be <code>null</code>, but may be empty. This property contains the following fields:</p>
	<p><i>fieldNames</i> String[]. Names of the fields used in the listener notification's reply document.</p>
	<p><i>fieldTypes</i> String[]. Data types of the fields used in the listener notification's reply document.</p>
Properties applicable to asynchronous listener notifications only	
<i>publishableRecordDef</i>	<p>IData. Required. Structure specifying the definition of the runtime publishable output document. This property contains the following fields:</p>
	<p><i>fieldNames</i> String[]. Names of the fields used in the listener notification's output document.</p>
	<p><i>fieldTypes</i> String[]. Data types of the fields used in the listener notification's output document.</p>

Output Parameters

None.

Example

You must construct *notificationSettings* to create a listener notification node. The value of a property's *systemName* is the internal name of the property. When constructing the *notificationSettings* input parameter, you must use this internal name as the key for setting a property's value. For example, if a listener notification template defines a property named *foo*, then its *systemName* as returned by *fetchListenerNotificationTemplateMetadata* service is *foo*. If the caller is a Java application, it might then use this information to set this property's value as follows:

```

IData pipeline = IDataFactory.create();
IDataCursor pipeCursor = pipeline.getCursor();
.
.
.
IData ntfySettings = IDataFactory.create();
IDataCursor ntfyCursor = ntfySettings.getCursor();
ntfyCursor.insertAfter("foo", "bar");
.
.
.
pipeCursor.insertAfter("notificationSettings",
ntfySettings);
.
.
.

```

In this example, the *foo* property takes a `java.lang.String` value.

wm.art.dev.notification:deleteListenerNotificationNode

This service deletes the specified listener notification node.

- You must disable the listener notification node before you delete it, using the `pub.art.notification:disableListenerNotification` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

- The delete action is immediate and non-reversible, and returns no output data.
- Set the `cascadeDelete` flag to propagate the deletion across the Broker.
- You may assume that the service completed successfully if it does not throw a checked exception.

Input Parameters

Name	Description
<i>notificationName</i>	String . Required. Name of the listener notification node in the format: <code>folder:node</code> .
<i>cascadeDelete</i>	Boolean . Possible values are: <ul style="list-style-type: none">■ <code>true</code>. Propagates the deletion across the Broker.■ <code>false</code>. Default.

Output Parameters

None.

wm.art.dev.notification:fetchListenerNotificationTemplateMetadata

This service fetches all metadata supported by the listener notification template for a specified connection alias and listener notification template class path.

This service takes a valid listener node and listener notification template class path, and returns an array of *templateProperties* structure, each containing attributes such as *systemName*, *parameterType*, *defaultValue*, and *isRequired*.

Input Parameters

Name	Description
<i>listenerNode</i>	String . Required. Name of an existing listener in the format: <code>folder:node</code> .
<i>notificationTemplate</i>	String . Required. Fully qualified pathname of listener notification template class.

Output Parameters

Name	Description
<i>description</i>	String . Required. Listener template description. Same as the value returned by <code>WmDescriptor.getDisplayName</code> method.
<i>displayName</i>	String . Required. Listener template name displayed. Same as the value returned by <code>WmDescriptor.getDescription</code> method.
<i>templateURL</i>	String . Required. URL of the online help page for the listener.

Name	Description
<i>indexMaps[i]</i>	IData[] . Required. An <i>i</i> -dimensioned array of field maps.
<i>mapName</i>	String . Field map name.
<i>isVariable</i>	Boolean . Specifies whether the field map is variable length.
<i>disableAppendAll</i>	Boolean . Disables all the buttons used for appending the rows for a field map in the Adapter Service Editor.
<i>templateProperties[n]</i>	IData . Required. An <i>n</i> -dimensioned array of properties.
<i>systemName</i>	String Required. Internal property name.
<i>displayName</i>	String Required. External property name displayed.
<i>description</i>	String Required. Property description.
<i>parameterType</i>	<p>String Required. Data type of property.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean. ■ Arrays are not supported.
<i>groupURL</i>	String . URL of the group's help page.
<i>groupName</i>	String . Name of the group to which the property belongs.
<i>tupleName</i>	String . Name of the tuple to which the property belongs.
<i>treeName</i>	String . Name of the tree to which the property belongs.
<i>treeDelimiter</i>	String . Delimiter character used in the tree.
<i>resourceDomain</i>	String . Resource domain name to which the property belongs.
<i>treeView</i>	String . Property name for which values are displayed in the tree structure for selection.
<i>defaultValue</i>	String . Default value of the property.

Name	Description
<i>isRequired</i>	Boolean. Specifies whether the property is required.
<i>isHidden</i>	Boolean. Specifies whether the property is displayable.
<i>isReadOnly</i>	Boolean. Specifies whether the property can be modified.
<i>isFill</i>	Boolean. Specifies whether the editors should pre-fill the property.
<i>isPassword</i>	Boolean. Specifies whether the property is a password.
<i>isMultiline</i>	Boolean. Specifies whether the property traverses multiple lines.
<i>isKey</i>	Boolean. Specifies whether the property is a key field.
<i>minSeqLength</i>	String. Lower bound of the sequence.
<i>maxSeqLength</i>	String. Upper bound of the sequence.
<i>minStringLength</i>	String. Minimum string length.
<i>maxStringLength</i>	String. Maximum string length.
<i>useParam</i>	String. Specifies whether the property is available for use.

wm.art.dev.notification:updateListenerNotificationNode

This service updates the existing synchronous or asynchronous listener notification node.

- You must disable the listener notification node before updating its properties, using the `pub.art.notification:disableListenerNotification` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

You must perform the following steps to populate the input pipeline:

1. Use `fetchListenerNotificationTemplateMetadata` service to identify the supported service template properties and configure the `listenerNode` and `notificationSettings` input parameter.
 - The value of the `systemName` metadata attribute is the internal name of a property. Use this internal name as the key for setting a property's value when constructing the input parameter `notificationSettings`.

2. Set only those properties that you want to change.

This service attempts to overlay these new values on the listener notification's current property values. The resulting set of merged property values are used to reconfigure the listener.

3. If you are providing explicit property values in *notificationSettings* parameter, then the values must conform to the underlying data types of those properties.
4. Change the listener resource that the listener notification uses by providing a new *listenerNode* input parameter.
 - If you omit the *listenerNode* parameter, the listener notification continues to use its current listener resource.
 - If you are changing only the listener resource, it is not necessary to provide the *notificationSettings* or any other input parameter.
5. A synchronous listener notification contains three additional input parameters:
 - a. *serviceName*. Specifies the node name of a separate service that is invoked by the notification at runtime. If you omit this parameter, the notifications still retains the current value of the parameter.
 - b. *requestRecordDef* and *replyRecordDef*. Document record definitions that the notification uses to format messages when communicating with this service. You can modify either or both of these definitions.
 - The *requestRecordDef* and *replyRecordDef* input parameters consists of *fieldNames* and *fieldTypes* properties.
 - The *fieldNames* and *fieldTypes* properties are an array of `java.lang.String`.
 - A one-to-one correspondence exists between the elements in these *fieldNames* and *fieldTypes* arrays.

Note:

This service cannot verify this assertion. If you configure either record definition incorrectly, the consequence may not manifest until runtime.

6. An asynchronous listener notification use the following input parameters:
 - a. *publishableRecordDef*. Redefine the structure of an asynchronous notification's output document.
 - The *publishableRecordDef* input parameter consists of *fieldNames* and *fieldTypes* properties.
 - The *fieldNames* and *fieldTypes* properties are an array of `java.lang.String`.
 - A one-to-one correspondence exists between the elements in these *fieldNames* and *fieldTypes* arrays.
 - The values assigned to the *fieldNames* and *fieldTypes* properties must correspond to fields that the listener notification class outputs in its `runNotification` method.

Note:

This service cannot verify this assertion. If you configure either record definition incorrectly, the consequence may not manifest until runtime.

7. The *requestRecordDef*, *replyRecordDef*, and *publishableRecordDef* input parameters can be replaced in their entirety. The service does not attempt to merge the values in these parameters with the notification's current values. For instance, if you want to redefine a synchronous notification's reply record definition, you must define the entire record in the *replyRecordDef* input parameter.
8. The two *listenerNode* and *notificationSettings* input parameters apply to both synchronous and asynchronous listener notifications.

Note:

1. You cannot convert a synchronous notification to an asynchronous notification.
2. Similarly you cannot convert an asynchronous notification to a synchronous notification.

The resource domains registered by the listener notification templates are set in the listener notification's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Input Parameters

Name	Description
<i>notificationName</i>	String . Required. Name of the existing listener notification in the format: <code>folder:node</code> .
<i>listenerNode</i>	String . Required. Name of the associated listener in the format: <code>folder:node</code> .
<i>notificationSettings</i>	IData . Required. Structure for passing the listener notification's property values.
<i>systemName</i>	<p>String. Required. Internal property name.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. ■ Arrays of all the above data types are also supported.

Properties applicable to synchronous listener notifications only

<i>serviceName</i>	String . Required. Name of the service to invoke in the format: <code>folder:node</code> . This parameter must not be <code>null</code> .
--------------------	--

Name	Description
<i>executionMode</i>	IData. Structure to identify whether a service defined in <i>serviceName</i> is invoked or a document is published.
<i>mode</i>	String. Possible values are: <ul style="list-style-type: none"> ■ <code>publishAndWait</code>. Publish the document and wait for the response. ■ <code>invokeService</code>. Invoke the service defined in <i>serviceName</i> parameter.
<i>local</i>	Boolean. Enables publishing the document locally.
<i>waitTime</i>	String. Time in millisecond to wait for the response in an synchronous listener notification.
<i>jmsSettings</i>	IData. Structure specifying the JMS settings.
<i>isJMSConfigured</i>	Boolean. Specifies whether the document is published to a JMS provider.
<i>ConnectionAliasName</i>	String. Required. Connection name of the JMS provider. This field is relevant if the value of isJMSConfigured is true.
<i>DestinationName</i>	String. Required. Name of the destination where the document is published. This field is relevant if the value of isJMSConfigured is true.
<i>DestinationType</i>	String. Required. Type of destination where the document is published. Possible values are: <ul style="list-style-type: none"> ■ <code>Queue</code> ■ <code>Topic</code> This field is relevant if the value of isJMSConfigured is true.
<i>requestRecordDef</i>	IData. Required. Specifies the definition of the request document sent to <i>serviceName</i> . This parameter must not be <code>null</code> , but may be empty. This property contains the following fields:
<i>fieldNames</i>	String[]. Names of the fields used in the listener notification's request document.
<i>fieldTypes</i>	String[]. Data types of the fields used in the listener notification's request document.

Name	Description
<i>replyRecordDef</i>	IData . Required. Specifies the definition of the reply document received from <i>serviceName</i> . This parameter must not be null, but may be empty.
<i>fieldNames</i>	String[] . Names of the fields used in the listener notification's reply document.
<i>fieldTypes</i>	String[] . Data types of the fields used in the listener notification's reply document.
Properties applicable to asynchronous listener notifications only	
<i>publishableRecordDef</i>	IData . Required. Specifies the definition of the runtime publishable output document. This property contains the following fields:
<i>fieldNames</i>	String[] . Names of the fields used in the listener notification's output document.
<i>fieldTypes</i>	String[] . Data types of the fields used in the listener notification's output document.

Output Parameters

None.

Polling Notification Services

These services are located in the WmART.wm.art.dev.notification package.

Service	Description
wm.art.dev.notification:createPollingNotificationNode	Creates a new instance of a polling notification in the specified package and folder from the specified notification template and connection alias.
wm.art.dev.notification:deletePollingNotificationNode	Removes an instance of a specified polling notification.
wm.art.dev.notification:fetchPollingNotificationTemplateMetadata	Returns all metadata for a specified polling notification template, and returns any scheduling properties that are set for the notification.
wm.art.dev.notification:updatePollingNotificationNode	Alters the characteristics of an existing polling notification.

wm.art.dev.notification:createPollingNotificationNode

This service creates a new instance of a polling notification in the specified package and folder from the specified notification template and connection alias, and initializes the new polling notification in a disabled state.

You must perform the following steps to populate the input pipeline:

1. Use `fetchPollingNotificationTemplateMetadata` service to identify the supported polling notification template properties and configure the input parameter `notificationSettings` structure.
 - Polling notification properties may or may not have default values, depending on the specific adapter's implementation.
 - Depending on the underlying data type of the property, it might not be possible to assign a default value to a property.
 - You may use the default values or override them with values that conform to the underlying data types of the properties.
 - The property values are not automatically set to their default values; you must explicitly set all the required properties.
 - If you omit a property, then no attempt is made to locate and assign default values to the property.
 - Polling notification properties may be optional or a required property.
 - A required property is identified when the `isRequired` metadata attribute is set to `true`.
 - The absence of the `isRequired` attribute implies that the property is not required.
 - If you fail to set a required property, the `wm.art.dev.notification:createPollingNotificationNode` service throws an exception.
 - You may have properties that might be required, based on the current value of some other property.
 - The number of properties to be configured is adapter-dependent. At a minimum, set those properties whose `isRequired` attribute is set to `true`.
2. This service creates a publishable document node in a `notificationNamePublishDocument` folder where `notificationName` is the value you specify for the `notificationName` input parameter. The name of publishable document is not configurable.
3. When a polling notification is triggered, Integration Server generates a runtime publishable output document.
 - The names and types of the data fields in the publishable output document is predefined.
 - Specify the names and types of the data fields in the publishable output document in the service's `publishableRecordDef` input parameter in `notificationSettings` parameter.

- The *fieldNames* and *fieldTypes* properties of *publishableRecordDef* are an array of *String*. A one-to-one correspondence exists between the elements in these two arrays.
 - The values assigned to the *fieldNames* and *fieldTypes* properties of *publishableRecordDef* must correspond to fields that the notification class outputs in its *runNotification* method.
 - The service execution may fail if an empty *publishableRecordDef* is specified.
5. A newly created polling notification is not assigned a delivery schedule. You must configure the polling notification's delivery schedule before enabling it. Provide a valid *scheduleSettings* input parameter in the call to *createPollingNotificationNode* or *updatePollingNotificationNode* service.
 6. You can activate the polling notification using the *pub.art.notification:enablePollingNotification* service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

Note:

This service does not validate that the fields in the publishable document are actually generated by the notification.

The resource domains registered by the polling notification templates are set in the polling notification's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Input Parameters

Name	Description
<i>notificationName</i>	String . Required. Name of the polling notification in the format: <i>folder:node</i> .
<i>packageName</i>	String . Required. Package the polling notification is installed.
<i>connectionAlias</i>	String . Required. Name of the connection in the format: <i>folder:node</i> .
<i>notificationTemplate</i>	String . Required. Fully qualified pathname of the polling notification template class.
<i>notificationSettings</i>	IData . Required. Structure for passing the polling notification's property values.
<i>systemName</i>	<p>String. Required. Internal name of the property.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <i>char</i>, <i>short</i>, <i>int</i>, <i>long</i>, <i>float</i>, <i>double</i>, <i>boolean</i>, <i>java.lang.String</i>, <i>java.lang.Character</i>, <i>java.lang.Short</i>, <i>java.lang.Integer</i>, <i>java.lang.Long</i>, <i>java.lang.Float</i>, <i>java.lang.Double</i>, <i>java.lang.Boolean</i>.

Name	Description
	<ul style="list-style-type: none"> ■ Arrays of all the above data types are also supported.
<i>scheduleSettings</i>	IData . Required. Structure for specifying the scheduling properties.
<i>notificationInterval</i>	Integer . Frequency to poll in seconds.
<i>notificationOverlap</i>	Boolean . Specifies whether notifications can overlap.
<i>notificationImmediate</i>	Boolean . Specifies whether to invoke the notification immediately.
<i>clusterProperties[k]</i>	IData[] . A <i>k</i> -dimensioned array of cluster properties.
<i>systemName</i>	String . Required. Internal property name
<i>parameterType</i>	String . Required. Data type of the property
<i>publishableRecordDef</i>	IData . Required. Structure for specifying the definition of the runtime publishable output document.
<i>fieldNames</i>	String[] . Names of the fields used in the polling notification's output document.
<i>fieldTypes</i>	String[] . Data types of the fields used in the polling notification's output document.
<i>jmsSettings</i>	IData . Structure specifying the JMS setting.
<i>isJMSConfigured</i>	Boolean . Specifies whether the document is published to a JMS provider.
<i>ConnectionAliasName</i>	String . Required. Connection name of the JMS provider. This field is relevant if the value of isJMSConfigured is true.
<i>DestinationName</i>	String . Required. Name of the destination where the document is published. This field is relevant if the value of isJMSConfigured is true.
<i>DestinationType</i>	<p>String. Required. Type of destination where the document is published. Possible values are:</p> <ul style="list-style-type: none"> ■ Queue ■ Topic <p>This field is relevant if the value of isJMSConfigured is true.</p>

Output Parameters

None.

Example

You must construct *notificationSettings* to create a polling notification node. The value of a property's *systemName* is the internal name of the property. When constructing the *notificationSettings* input parameter, you must use this internal name as the key for setting a property's value. For example, if a polling notification template defines a property named *sqlCommand*, then its *systemName* as returned by *fetchPollingNotificationTemplateMetadata* service is *sqlCommand*. If the caller is a Java application, it might then use this information to set this property's value as follows:

```
IData pipeline = IDataFactory.create();
IDataCursor pipeCursor = pipeline.getCursor();
.
.
.
IData ntfySettings = IDataFactory.create();
IDataCursor ntfyCursor = ntfySettings.getCursor();
ntfyCursor.insertAfter("sqlCommand", "SELECT * FROM fooTable");
.
.
.
pipeCursor.insertAfter("notificationSettings", ntfySettings);
.
.
.
```

In this example, the *sqlCommand* property takes a `java.lang.String` value.

wm.art.dev.notification:deletePollingNotificationNode

This service deletes the specified polling notification node. You must disable the notification before deleting it, using the *disablePollingNotificationNode* service. For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

- You must disable the polling notification node before you delete it, using the `pub.art.notification:disablePollingNotificationNode` service.

For more information, see the *webMethods Integration Server Built-In Services Reference* for your release.

- The delete action is immediate and non-reversible, and returns no output data.
- Set the *cascadeDelete* flag to propagate the deletion across Broker.
- You may assume that the service completed successfully if it does not throw a checked exception.

This action is immediate and non-reversible, and returns no output data. You may assume that the service completed successfully if it does not throw a checked exception.

Input Parameters

Name	Description
<i>notificationName</i>	String. Required. Name of the polling notification in the format: <code>folder:node</code> .
<i>cascadeDelete</i>	Boolean. Possible values are: <ul style="list-style-type: none"> ■ <code>true</code>. Propagates the deletion across the Broker. ■ <code>false</code>. Default.

Output Parameters

None.

wm.art.dev.notification:fetchPollingNotificationTemplateMetadata

This service fetches all metadata supported by the polling notification template, and the scheduling properties for a specified connection alias and the polling notification template class path.

This service takes a valid connection alias and polling notification template class path, and returns an array of *templateProperties* and an array of *scheduleProperties* structure. Each polling notification template properties (*templateProperties* structure) includes *systemName*, *parameterType*, *defaultValue*, and *isRequired* attributes, which you can use to configure a new polling notification instance.

Input Parameters

Name	Description
<i>connectionAlias</i>	String. Required. Namespace name of an existing connection in the format: <code>folder:node</code> .
<i>notificationTemplate</i>	String. Required. Fully qualified pathname of the polling notification template class.

Output Parameters

Name	Description
<i>description</i>	String. Required. Polling notification template description. Same as the value returned by <code>WmDescriptor.getDisplayName</code> method.
<i>displayName</i>	String. Required. Polling notification template name displayed. Same as the value returned by <code>WmDescriptor.getDescription</code> method.
<i>templateURL</i>	String. Required. URL of the online help page for the polling notification.

Name	Description
<i>indexMaps[i]</i>	IData[] . Required. An <i>i</i> -dimensioned array of field maps.
<i>mapName</i>	String . Field map name.
<i>isVariable</i>	Boolean . Specifies whether the field map is variable length.
<i>disableAppendAll</i>	Boolean . Disables all the buttons used for appending the rows for a field map in the Adapter Service Editor.
<i>templateProperties[n]</i>	IData . Required. An <i>n</i> -dimensioned array of properties.
<i>systemName</i>	String Required. Internal property name.
<i>displayName</i>	String Required. External property name displayed.
<i>description</i>	String Required. Description of the property.
<i>parameterType</i>	<p>String Required. Data type of property.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean. ■ Arrays are not supported.
<i>groupURL</i>	String . URL of the group's help page.
<i>groupName</i>	String . Name of the group to which the property belongs.
<i>tupleName</i>	String . Name of the tuple to which the property belongs.
<i>treeName</i>	String . Name of the tree to which the property belongs.
<i>treeDelimiter</i>	String . Delimiter character used in the tree.
<i>resourceDomain</i>	String . Resource domain name to which the property belongs.
<i>treeView</i>	String . Property name for which values are displayed in the tree structure for selection.
<i>defaultValue</i>	String . Default value of the property.

Name	Description
<i>isRequired</i>	Boolean. Specifies whether the property is required.
<i>isHidden</i>	Boolean. Specifies whether the property is displayable.
<i>isReadOnly</i>	Boolean. Specifies whether the property can be modified.
<i>isFill</i>	Boolean. Specifies whether the editors should pre-fill the property.
<i>isPassword</i>	Boolean. Specifies whether the property is a password.
<i>isMultiline</i>	Boolean. Specifies whether the property traverses multiple lines.
<i>isKey</i>	Boolean. Specifies whether the property is a key field.
<i>minSeqLength</i>	String. Lower bound of the sequence.
<i>maxSeqLength</i>	String. Upper bound of the sequence.
<i>minStringLength</i>	String. Minimum string length.
<i>maxStringLength</i>	String. Maximum string length.
<i>useParam</i>	String. Specifies whether the property is available for use.
<i>scheduleProperties[j]</i>	IData[]. A <i>j</i> -dimensioned array of scheduling properties.
<i>systemName</i>	String. Required. Internal property name
<i>parameterType</i>	String. Required. Data type of the property
<i>clusterProperties[k]</i>	IData[]. A <i>k</i> -dimensioned array of cluster properties.
<i>systemName</i>	String. Required. Internal property name
<i>parameterType</i>	String. Required. Data type of the property

wm.art.dev.notification:updatePollingNotificationNode

This service updates the values of an existing polling notification. You must perform the following:

- You must disable the polling notification before updating its properties, using the `pub.art.notification:disablePollingNotification` service.

- You can modify the following items:
 - Underlying connection resource used by the notification
 - Metadata property values
 - Schedule
 - Signature of its publishable output document

You must perform the following steps to populate the input pipeline:

1. Use `wm.art.dev.notification:fetchPollingNotificationTemplateMetadata` service to identify the supported service template properties and configure the `connectionAlias`, `notificationSettings`, `scheduleSettings`, and `publishableRecordDef` input parameter.
2. Configure the `connectionAlias`, `notificationSettings`, `scheduleSettings` input parameters.
3. Set only those properties that you want to change in `connectionAlias`, `notificationSettings`, `scheduleSettings` input parameters.

This service attempts to overlay these new values on the polling notification's current property values. The resulting set of merged property values are used to reconfigure the polling notification. For instance, change the connection resource that the polling notification uses by providing a new `connectionAlias` input parameter. If you omit this parameter, the polling notification continues to use the current connection resource. If you are changing only the connection resource, it is not necessary to provide the `notificationSettings`, `scheduleSettings`, or `publishableRecordDef` input parameters.

4. If you are providing explicit property values in `notificationSettings` parameter, then the values must conform to the underlying data types of those properties.
5. Configuring the `publishableRecordDef`.
 - The `publishableRecordDef` input parameter must be replaced in its entirety. The service does not attempt to merge the values into the current publishable record definition. .
 - The `fieldNames` and `fieldTypes` properties of `publishableRecordDef` are an array of String. A one-to-one correspondence exists between the elements in these two arrays.
 - The values assigned to the `fieldNames` and `fieldTypes` properties of `publishableRecordDef` must correspond to fields that the notification class and the invoked service use to communicate with each other.

Note:

This service cannot verify this assertion. If you configure either record definition incorrectly, the consequence may not manifest until runtime.

6. The value of the `systemName` metadata attribute is the internal name of a property. Use this internal name as the key for setting a property's value when constructing the input parameter `notificationSettings`.
7. Update the property depending on the data type.

- If a property's data type is non-primitive (derived from `java.lang.Object`), you may clear a property's current value by setting its value to `null`.
- If a property's data type is Java primitive, then the property value cannot be cleared; however the property can be updated.

The resource domains registered by the polling notification templates are set in the polling notification's properties according to the interdependencies between the resource domains. Knowledge of these interdependencies is adapter-specific, and beyond the scope of this service. This service does not interpret resource domains.

Input Parameters

Name	Description
<i>notificationName</i>	String . Required. Namespace name of the existing polling notification in the format: <code>folder:node</code> .
<i>connectionAlias</i>	String . Required. Namespace name of the connection in the format: <code>folder:node</code> .
<i>notificationSettings</i>	IData . Required. Structure for passing the polling notification's property values.
<i>systemName</i>	<p>String. Required. Internal property name.</p> <ul style="list-style-type: none"> ■ The following Java data types are supported for connections: <code>char</code>, <code>short</code>, <code>int</code>, <code>long</code>, <code>float</code>, <code>double</code>, <code>boolean</code>, <code>java.lang.String</code>, <code>java.lang.Character</code>, <code>java.lang.Short</code>, <code>java.lang.Integer</code>, <code>java.lang.Long</code>, <code>java.lang.Float</code>, <code>java.lang.Double</code>, <code>java.lang.Boolean</code>. ■ Arrays of all the above data types are also supported.
<i>scheduleSettings</i>	IData . Required. Specifies the scheduling properties.
<i>notificationInterval</i>	Integer . Frequency to poll in seconds.
<i>notificationOverlap</i>	Boolean . Specifies whether notifications can overlap.
<i>notificationImmediate</i>	Boolean . Specifies whether to invoke the notification immediately.
<i>clusterProperties[k]</i>	IData[] . A <i>k</i> -dimensioned array of cluster properties.
<i>systemName</i>	String . Required. Internal property name

Name	Description
<i>parameterType</i>	String . Required. Data type of the property
<i>publishableRecordDef</i>	IData . Required. Specifies the definition of the runtime publishable output document.
<i>fieldNames</i>	String[] . Names of the fields used in the polling notification's output document.
<i>fieldTypes</i>	String[] . Data types of the fields used in the polling notification's output document. <ul style="list-style-type: none">■ The following Java data types are supported for connections: char, short, int, long, float, double, boolean, java.lang.String, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.lang.Boolean.■ Arrays of all the above data types are also supported.

Output Parameters

None.

D Using the Sample Adapter

■ Overview	442
■ The Sample Server	442
■ Banking Services, Queries and Alerts	444
■ Prerequisites for Code Compilation	445
■ Phase 1: Creating an Adapter Definition	446
■ Phase 2: Adding a Connection	449
■ Phase 3: Adding Adapter Services	454
■ Phase 4: Adding Polling Notifications	461
■ Phase 5: Adding Listener Notifications	470

Overview

The ADK provides an example adapter package named `WmSampleAdapter`. You can use this adapter as a model for developing your own adapters. This adapter enables you to exchange data with a simulated backend resource provided with the ADK, named *Sample Server*. You will configure this adapter to perform a banking application. All of the underlying Sample Adapter class files are located in the `Integration Server_directory\instances\<instance_name>\packages\WmSampleAdapter` directory. This appendix describes how the Sample Adapter was developed, and how you can configure and run it. The adapter is provided in five phases. Each phase is a standalone adapter, delivered in its own source files; you can build, configure, and run each phase separately. Each phase includes new functionality such that the first phase consists of just the basic framework of the adapter, and the final phase is the fully functional adapter. These phases are as follows:

1. **Phase 1:** Creating the adapter definition.

An adapter definition is recognized as an adapter by Integration Server, but lacks functionality. In the next four phases of development, you add new functionality to create an adapter connection, an adapter service template, a polling notification template, and a listener notification template.

2. **Phase 2:** Adding a connection that connects clients to the *Sample Server*.

3. **Phase 3:** Adding adapter services that perform operations such as depositing, clearing, and bouncing checks, withdrawing and transferring funds, and other operations. For more information, see [“Banking Services, Queries and Alerts” on page 444](#).

4. **Phase 4:** Adding polling notifications and configuring polling notification nodes that query the *Sample Server* and publish documents when the following events occur:

- The Clear Check service or the Bounce Check service clears (approves) or bounces (disapproves) a deposited check.
- The Withdraw service or the Transfer service causes a negative account balance.

5. **Phase 5:** Adding listener and listener notifications. This phase includes the following:

- Configuring a listener to monitor the *Sample Server* for alerts generated by the *Sample Server*. An alert is a *Sample Server* mechanism that informs the adapter that an event has occurred.
- Configuring the listener notification nodes to publish notification documents when the Sample Server generates the following alerts:
 - The Deposit service successfully deposits a check. You can then invoke the Clear Check service or the Bounce Check service to approve or disapprove the check.
 - The Withdraw service or the Transfer service causes a negative account balance.

This phase represents the adapter with all its functionality.

The Sample Server

The *Sample Adapter* provides the following:

1. **Sample Server.** Backend resource delivered as a Java executable.
 - Simulates a banking system.
 - Communicates with the *Sample Adapter* using the TCP/IP protocol.
 - Uses a document-based messaging scheme to exchange data with the *SampleAdapter*.
 - Executes as a separate process on any networked computer.
2. **SampleClient.jar.** Provides the classes necessary to perform the following:
 - Implement a document-based messaging scheme to exchange data with the *Sample Adapter*. These messages provide access to a set of business services, queries, and alerts that are supported by the *Sample Server*.
 - Provide a metadata repository lookup feature that allows the *Sample Adapter* to retrieve a list of the available services, queries, and alerts, and to obtain details of the data fields used in each one.

Sample Server Client APIs

WmSampleConnection communicates with the *Sample Server* using the *Sample Server* client API. Javadocs for the API are available in the *Integration Server_directory*\instances*<instance_name>*\packages\WmSampleAdapter\backendResource\doc directory.

Transaction Control

You can configure the client connection for either no transaction control (auto commit mode) or local transaction control across multiple service invocations. The *SampleAdapter* allows the adapter user to select the transaction type for each WmSampleConnection node.

Customizing Sample Server

You can control the behavior of the *Sample Server* using `SampleServer.properties` property file, located in the *Integration Server_directory*\instances*<instance_name>*\packages\WmSampleAdapter\backendResource\doc directory. You can edit the file to suit your needs, and then restart the server.

You can move the `SampleServer.jar` and the `SampleServer.properties` file to any platform that has a JVM installed so that these files can be accessed through a network connection.

Starting the Sample Server

> To start the Sample Server

1. Navigate to *Integration Server_directory*\instances*<instance_name>*\packages\WmSampleAdapter\backendResource directory.
2. Start *Sample Adapter*.

- Execute the script **startSampleServer.sh**.
- Execute the following:
 - Navigate to *Integration Server_directory* \instances*<instance_name>*\packages\WmSampleAdapter\backendResource\doc directory.
 - Run the command: `java -jar SampleServer.jar`

Banking Services, Queries and Alerts

Banking Services

The *Sample Server* provides the following banking services. You can use the adapter service template to create adapter service nodes that use these services:

Service Name	Description
Deposit	Creates multiple cash and check deposits. When the check number is equal to or less than 0, the deposit is considered to be a cash deposit.
Bounce Check	Bounces (disapproves) a check deposit.
Clear Check	Clears (approves) a check deposit.
Get Balance	Retrieves the current balance of the specified account.
Get History	Retrieves the service history of the specified account.
Transfer	Transfers funds between two accounts. The user ID and Personal Identification Number (PIN) are authenticated against both accounts.
Withdraw	Withdraws funds from the specified account. A negative amount cannot be withdrawn.

Banking Event Queries

Banking event queries retrieve information about specific activities that occur in the *Sample Server* as a result of banking service requests. Each occurrence of a banking event is returned only once. The *Sample Server* uses polling notifications to retrieve these events and to generate notification documents.

The *Sample Server* supports the following banking event queries:

Event Query	Description
CheckDepositStatusChange	Requests a list of checks that have cleared or bounced since the last time the query was made. These events occur as a result of the Clear Check or Bounce Check services.

Event Query	Description
UnderBalance	Requests a list of accounts that have been overdrawn since the last time the query was made. These events occur as a result of Withdraw or Transfer services that result in a negative account balance.
	<p>Note: The Withdraw or Transfer service request is successful if the request had been rejected because the debit exceeded the account's credit limit. In this case, no UnderBalance event is recorded.</p>

Note:

The *Sample Server* client cannot acknowledge the polling message from the receiving side as no protocol support exists. Consequently a document is not published with the same message twice. For example, if a document reports that an account has fallen to -10, no other document is published until the amount changes.

Banking Alerts

Banking alerts are generated when specified events occur on the *Sample Server*. Unlike banking event queries, banking alerts are generated and delivered in real time to a configured address. The *Sample Adapter* implements listeners to monitor these addresses and to retrieve the alert data. Listener notifications are then used to distribute the alert information to Integration Server services. There is no queuing mechanism for alerts, so if no listener is ready to retrieve the alert data, it will be lost.

The *Sample Server* supports the following types of alerts:

Alerts	Description
CheckDepositNotification	Generated for each check that is successfully deposited with the Deposit service.
UnderBalanceNotification	Generated for each Withdraw or Transfer service that results in a negative account balance. This alert is identical to the UnderBalance banking event, but is delivered as an alert.

Prerequisites for Code Compilation

The prerequisites for compiling the *Sample Adapter* package are as follows:

- Java SDK is installed with Integration Server.
- Update your jcode script in the *Integration Server_directory* \instances*<instance_name>*\bin directory as follows:
 - In the very last line of the command that executes the NodeUtil class, you must add the following paths to the classpath in order to compile the code correctly:

- `Software AG_directory \common\lib\glassfish\gf.jakarta.resource.jar`

where *Software AG_directory* is the directory in which you installed Integration Server.

Phase 1: Creating an Adapter Definition

An adapter definition simply defines the adapter in your Integration Server. To create the adapter definition, include the following classes:

Class	Description
<i>WmSampleAdapter</i>	Represents the main class of the adapter. In the <i>Sample Adapter</i> package's main source code directory, the adapter definition extends the <code>com.wm.adk.WmAdapter</code> base class.
<i>WmSampleAdapterConstants</i>	Contains all the string constants used by the adapter including the major code, group names, parameter names, bean property names, and resource domain names.
<i>WmSampleAdapterResourceBundle</i>	Contains all display strings and messages used by the adapter at runtime and at design time.
<i>admin</i>	Registers and delists the adapter when the adapter package starts up and shuts down.

The first three classes are located in the `com.wm.adapter.wmSampleAdapter` Java package. The `admin` class is located in the `wm.wmSampleAdapter` Java package.

This section describes how to:

- Disable the Sample Adapter Package
- Create the `MyWmSampleAdapter` package
- Compile the `MyWmSampleAdapter` package
- Test the `MyWmSampleAdapter` package
- Disable the `MyWmSampleAdapter` package

Disabling Sample Adapter Package

Before you begin, make sure you disable the `WmSampleAdapter` package.

> To disable the sample adapter package

1. Start Integration Server Administrator.
2. Navigate to **Packages > Management** screen.

- Click **Yes** in the **Enabled** column of WmSampleAdapter package.

The **Enabled** column now shows **No** (disabled).

Important:


You must disable the WmSampleAdapter and all MyWmSampleAdapterN packages before you proceed. All of these packages have the same adapter major code and conflict with each other if they are not disabled.

- Refresh Integration Server Administrator.

The **Sample Adapter** is no longer listed on the **Adapters** in the Integration Server Administrator screen.

Creating the MyWmSampleAdapter Package

> To create the WmSampleAdapter1 package

- Start Designer.
- Go to **File > New**.
- Select **Package** from the list of elements.
- Assign the name MyWmSampleAdapter to the package and click **Finish**.
- In the **Package Navigator**, select the MyWmSampleAdapter package.
- Select **File > Properties**.
- In the Package Dependencies section, click  to add a row and specify values for the following fields:

Field	Value
Package	WmART
Version	*.*

Click **OK**.

- Click **OK**.

Compiling the MyWmSampleAdapter Package

➤ To compile the MyWmSampleAdapter package

1. Copy all the source code from the `WmSampleAdapter\code\sourcePhase1` directory to the `MyWmSampleAdapter\code\source` directory.

Important:

You must maintain the proper subdirectory structure when copying the source code files.

2. Copy the `SampleClient.jar` file from the `WmSampleAdapter\code\jars` directory to the `MyWmSampleAdapter\code\jars` directory.

If the `MyWmSampleAdapter\code\jars` directory does not exist, you must create it.

3. Copy all the files in `WmAdapterSample\pub` directory to `MyWmSampleAdapter\pub` directory except the `index.html` file .
4. Execute the following commands from a command prompt:

a. `jcode makeall MyWmSampleAdapter`

b. `jcode fragall MyWmSampleAdapter`

5. Restart Integration Server.
6. Start Integration Server Administrator.
7. Navigate to **Packages > Management** screen.

The `MyWmSampleAdapter` package is now available.

Adding the MyWmSampleAdapter's Startup and ShutDown Services

➤ To add the MyWmSampleAdapter's startup and shutDown services

1. Using Designer, click **File > Refresh**.
2. In the **Package Navigator**, select the `MyWmSampleAdapter` package.
3. Click **File > Properties**.
4. Navigate to **Properties > Startup/Shutdown Services** and assign the following services:

Field	Value
Startup Services	<code>wm.wmSampleAdapter.admin:startUp</code>
Shutdown Services	<code>wm.wmSampleAdapter.admin:shutDown</code>

Click **Apply and Close**.

Testing the MyWmSampleAdapter Package

➤ To test the MyWmSampleAdapter package

1. In Integration Server Administrator, navigate to **Packages > Management** and reload the *MyWmSampleAdapter* package.
2. Refresh Integration Server Administrator.

The **Sample Adapter** is listed on the **Adapters** screen.

Disabling the Phase 1 Implementation

After you compile and test the Phase 1 code, you must disable or delete the *MyWmSampleAdapter* package before you compile and test Phase 2. Not disabling or deleting the package results in a conflict of major codes when you compile and test Phase 2.

Phase 2: Adding a Connection

In this phase, the sample provides a connection template that connects clients to the *Sample Server*. This section describes how to:

- Implement the connection template.
- Revise the adapter definition classes.
- Compile the phase 2 implementation.
- Create the *TestMyWmSampleAdapter* package.
- Configure a connection node.
- Enable the connection node.
- Disable the phase 2 implementation.

Implementing the Connection Template

To define the connection template, the adapter includes the following classes in the `com.wm.adapter.wmSampleAdapter.connection` package:

- *WmSampleConnection* class, which extends the `com.wm.adk.connectionWmManagedConnection` class
- *WmSampleConnectionFactory* class, which extends the `com.wm.adk.connectionWmManagedConnectionFactory` class

The bean properties declared in the *WmSampleConnectionFactory* class are:

Property	Description
<i>sampleServerHostName</i>	IP host name for the computer where the <i>Sample Server</i> is running.
<i>sampleServerPortNumber</i>	TCP/IP port number that <i>Sample Server</i> accepts client connection. Default value is 4444.
<i>timeout</i>	Number of milliseconds that the Integration Server waits to obtain a connection with the <i>Sample Server</i> before it times out and returns an error. Default value is 20000.
<i>transactionType</i>	Flag indicating whether the connection requests local transaction control or an auto commit mode. Possible values are: <ul style="list-style-type: none"> ■ true. ■ false.

The *WmSampleLocalTransaction* class is implemented to support the configurable local transaction control for the *WmSampleConnection* connection.

Revising Adapter Definition Classes From Phase 1

In Phase 2, the existing classes from Phase 1 are modified to include the following revisions. The classes contain comments that detail the following changes:

Class	Revision
<i>WmSampleAdapter</i>	Added a reference to <i>WmSampleConnectionFactory</i> class in the <code>fillAdapterTypeInfo</code> method.
<i>WmSampleAdapterConstants</i>	Added string constants for the connection property names.
<i>WmSampleAdapterResourceBundle</i>	Added entries for the connection property configuration.

Compiling the Phase 2 Implementation

To compile the Phase 2 implementation, perform the following:

- Disable the *WmSampleAdapter* and *MyWmSampleAdapter* packages.

Important:

The `WmSampleAdapter` and `MyWmSampleAdapter` packages have the same adapter major code and conflict with each other if they are not disabled.

- Copy all the source code from the `WmSampleAdapter\code\sourcePhase2` directory to the `MyWmSampleAdapter\code\source` directory.
- Compile using the procedure in “[Compiling the MyWmSampleAdapter Package](#)” on page 448.
- Restart Integration Server.

Creating the TestMyWmSampleAdapter Package

➤ To create the TestMyWmSampleAdapter package

1. Start Designer.
2. Go to **File > New**.
3. Select **Package** from the list of elements.
4. Assign the name `TestMyWmSampleAdapter` to the package and click **Finish**.

Configuring the Connection Node

Perform the following procedure to configure the connection node.

➤ To configure the connection node

1. In Integration Server Administrator, navigate to **Adapters > Sample Adapter**.
The **Sample Adapter** management screen appears.
2. Select **Connections**.
3. Click **Configure New Connection**.
4. In the **Connection Types** screen, click **Sample Server Connection**.
5. In the **Configure Connection Type** screen, provide values for the connection's parameters.
 - a. Complete the **Configure Connection Type > Sample Adapter** section as follows:

Field	Description
Package	Select the namespace node package in which you create the connection. For example, <i>TestMyWmSampleAdapter</i> package.
Folder Name	Name of the folder in which you create the connection. For example, <i>connections</i> folder.
Connection Name	Name of the connection. For example, <i>sampleConnection</i> .

- b. Complete the **Connection Properties** section as follows:

Field	Description
Sample Server Host Name	<i>Sample Server</i> host name. For example, localhost.
Sample Server Port Number	<i>Sample Server</i> port number. Default value is 4444.
User Id	<i>Sample Server</i> user id. Type the <i>suid</i> value. <div style="background-color: #f0f0f0; padding: 5px;"> <p>Note: The user ids and passwords for the <i>Sample Server</i> are set in the <i>SampleServer.jar</i> file located in <i>Integration Server_directory</i> \instances\<i><instance_name></i>\packages\WmSampleAdapter\backendResource directory.</p> </div>
Password	<i>Sample Server</i> password. Type the <i>spin</i> value. <div style="background-color: #f0f0f0; padding: 5px;"> <p>Note: The user ids and passwords for the <i>Sample Server</i> are set in the <i>SampleServer.jar</i> file located in <i>Integration Server_directory</i> \instances\<i><instance_name></i>\packages\WmSampleAdapter\backendResource directory.</p> </div>
Local Transaction Control?	Determines if you want to control the local transaction. Possible values: <ul style="list-style-type: none"> ■ true. ■ false. Default. Select false.
Sample Connector Timeout	Timeout in milliseconds for the connection. Default value is 20000.

- c. Complete the **Connection Management Properties** section as follows:

Field	Description
Enable Connection Pooling	Enables the connection to use connection pooling. Default value is <code>true</code> .
Minimum Pool Size	Specifies the number of connections to create when the connection is enabled. Default value is 1.
Maximum Pool Size	Specifies the maximum number of connections that can exist at one time in the connection pool. Default value is 10.
Pool Increment Size	Specifies the number of connections by which the pool will be incremented if connections are needed, up to the maximum pool size. Default value is 1.
Block Timeout	Specifies the number of milliseconds that the Integration Server waits to obtain a connection with the <i>Sample Server</i> before it times out and returns an error. Default value is 20000.
Expire Timeout	Specifies the number of milliseconds that inactive connections can remain in the pool before they are closed and removed from the pool. Default value is 20000.
Startup Retry Count	Specifies the number of times that the system attempts to initialize the connection pool at startup if the initial attempt fails, before issuing an <code>AdapterConnectionException</code> . Default value is 0. If the value is 0, then the system makes a single attempt.
Startup Backoff Timeout	Specifies the number of seconds to wait between each attempt to initialize the connection pool. Default value is 10.
	<p>Note: This field is irrelevant if the value of Startup Retry Count is set to 0.</p>

- Click **Test Connection**.

The connection is tested based on the settings provided.

Note:

Ensure that the *Sample Server* is up and running before you test the connection.

- Click **Save Connection**.

The connection name is now listed on the adapter's **Connections** screen and in the Designer.

Enabling the Connection Node

- **To enable the connection node**

1. Start the *Sample Server* from a command prompt, as described in “Starting the Sample Server” on page 443.
2. To enable the connection node, on the adapter's Connections screen, click **No** in the **Enabled** column, the value changes to **Yes** (enabled).

The server initializes a connection pool based on the provided settings.

Note:

If a connection node is enabled when the Integration Server shuts down, it is enabled at Integration Server startup.

Disabling the Phase 2 Implementation

After you compile and test the Phase 2 code, you must disable or delete the `MyWmSampleAdapter` package before you compile and test Phase 3. Not disabling or deleting the package results in a conflict of major codes when you compile and test Phase 3.

Phase 3: Adding Adapter Services

In this phase, the sample provides an adapter service template that you can configure to execute the banking services in the *Sample Server*. This section describes how to:

- Implement the adapter service template.
- Revise adapter definition, and adapter connection classes.
- Compile the phase 3 implementation.
- Configure and enabling a connection node.
- Test the connection node.
- Disable the phase 3 implementation

Implementing the Adapter Service Template

Define an adapter service template by extending the `com.wm.adk.cci.interaction.WmAdapterService` base class. The following packages, classes, and methods are added in the *Sample Adapter*; thereby allowing you to configure and execute the services against the *Sample Server*:

- The adapter service templates are created in the `com.wm.adapter.wmSampleAdapter.service` package.
- The `AccountEnquiry`, and `AccountTransaction` are the adapter service template classes added in the `com.wm.adapter.wmSampleAdapter.service` Java package.
- A `DocumentHelp` utility class is added in the `com.wm.adapter.wmSampleAdapter.util` Java package. This utility facilitates the data structure conversion from a *Sample Server* document to the Integration Server's **IData**, and helps to decipher the adapter service and notification signature metadata received from the *Sample Server* repository.

- The bean properties declared in the AccountEnquiry adapter service template class are as follows:

Property	Description
<i>serviceName</i>	Service name. For a list of the services, see “Banking Services, Queries and Alerts” on page 444 .
<i>inputFieldNames</i>	Fully qualified suggested input parameter signature names, including all the record structures and array indicators.
	<p>Note: If you set the Boolean flag to <code>true</code> in the <code>createFieldMap</code> method, the adapter user has the option to overwrite the suggested names. In this implementation, the user cannot change the names.</p>
<i>inputFieldTypes</i>	Input parameter data types, including all the array indicators.
<i>hiddenInputFieldNames</i>	Hidden input parameter signature names, including all the record structures and array indicators.
<i>outputParameterNames</i>	Fully qualified output parameter names, including all the record structures and array indicators.
<i>outputFieldNames</i>	Fully qualified suggested output parameter signature names, including all the record structures and array indicators. You cannot change these names.
<i>outputFieldTypes</i>	Output parameter data types, including all the array indicators.
<i>hiddenOutputFieldTypes</i>	Hidden output parameter data types, including all the array indicators.

- The resource domains declared in the AccountEnquiry class are as follows:

Resource Domain Name	Description
<i>serviceName</i>	Looks up the list of service names. For a list of the services, see “Banking Services, Queries and Alerts” on page 444 .
<i>inputFieldNames</i>	Looks up the fully qualified input parameter names, including all the record structures and array indicators.
<i>inputFieldTypes</i>	Looks up the input parameter data types, including all the array indicators.
<i>outputParameterNames</i>	Looks up the fully qualified output parameter names, including all the record structures and array indicators.
<i>outputFieldTypes</i>	Looks up the output parameter data types, including all the array indicators.

Resource Domain Name	Description
----------------------	-------------

<i>hiddenOutputFieldTypes</i>	Looks up the hidden output parameter data types, including all the array indicators.
-------------------------------	--

- The bean properties declared in the AccountTransaction adapter service template class are as follows:

Property	Description
----------	-------------

<i>serviceName</i>	Service name. For a list of the services, see “Banking Services, Queries and Alerts” on page 444 .
--------------------	--

<i>inputParameterNames</i>	Fully qualified input parameter names, including all the record structures and array indicators.
----------------------------	--

<i>inputFieldNames</i>	Fully qualified suggested input parameter signature names, including all the record structures and array indicators.
------------------------	--

<i>inputFieldTypes</i>	Input parameter data types, including all the array indicators.
------------------------	---

<i>hiddenInputFieldNames</i>	Hidden input parameter signature names, including all the record structures and array indicators.
------------------------------	---

<i>outputParameterNames</i>	Fully qualified output parameter names, including all the record structures and array indicators.
-----------------------------	---

<i>outputFieldNames</i>	Fully qualified suggested output parameter signature names, including all the record structures and array indicators. You cannot change these names.
-------------------------	--

<i>outputFieldTypes</i>	Output parameter data types, including all the array indicators.
-------------------------	--

<i>hiddenOutputFieldTypes</i>	Hidden output parameter data types, including all the array indicators.
-------------------------------	---

- The resource domains declared in the AccountTransaction adapter service template class are as follows:

Resource Domain Name	Description
----------------------	-------------

<i>serviceName</i>	Looks up the list of service names. For a list of the services, see “Banking Services” on page 444 .
--------------------	--

<i>inputParameterNames</i>	Looks up the fully qualified input parameter names, including all the record structures and array indicators.
----------------------------	---

<i>hiddenInputFieldTypes</i>	Looks up the hidden input parameter data types, including all the array indicators.
------------------------------	---

Resource Domain Name	Description
<i>inputFieldTypes</i>	Looks up the input parameter data types, including all the array indicators.
<i>outputParameterNames</i>	Looks up the fully qualified output parameter names, including all the record structures and array indicators.
<i>outputFieldTypes</i>	Looks up the output parameter data types, including all the array indicators.
<i>hiddenOutputFieldTypes</i>	Looks up the hidden output parameter data types, including all the array indicators.

Implementing the execute Method in the AccountEnquiry and AccountTransaction Class

When the flow service invokes an adapter service node, the service calls the `WmAdapterService.execute` method. This method receives a `WmManagedConnection` object and a `WmRecord` object, and returns a `WmRecord` object.

The `WmAdapterService.execute` method uses the *Sample Server* client API to create a request document with the input parameters and sends it to the *Sample Server*. The request can receive one of three possible responses:

- Success with output: The service succeeds, and receives an acknowledgment document and output. For example, a `getBalance` service returns an account balance.
- Success with no output: The service succeeds, and receives an acknowledgment document, but there is no output. For example, a `Deposit` service simply deposits an amount, but returns no output.
- Failure. The service fails and receives a negative acknowledgment document; an `AdapterException` is thrown with the appropriate error message.

Revising Code From Phases 1 and 2

In Phase 3, the existing classes from Phase 1 and Phase 2 are modified to include the following revisions. The classes contain comments that detail the changes.

Class	Revision
<code>WmSampleAdapterConstants</code>	Added string constants for the service property names.
<code>WmSampleAdapterResourceBundle</code>	Added entries for the service property configuration.
<code>WmSampleConnectionFactory</code>	Added a reference to <code>AccountEnquiry</code> and <code>AccountTransaction</code> class in the <code>fillResourceAdapterMetadataInfo</code> method.

Class	Revision
WmSampleConnection	<p>In the registerResourceDomain method, the sample code registers all resource domains declared by the AccountEnquiry and AccountTransaction service template classes.</p> <p>In the adapterResourceDomainLookup method, the sample code includes resource domain lookup code to request the service metadata from the <i>Sample Server</i> repository.</p>

Compiling the Phase 3 Implementation

To compile the Phase 3 implementation perform the following:

- Disable the WmSampleAdapter and MyWmSampleAdapter packages.

Important:

The WmSampleAdapter and MyWmSampleAdapter packages have the same adapter major code and conflict with each other if they are not disabled.

- Copy all the source code from the WmSampleAdapter\code\sourcePhase3 directory to the MyWmSampleAdapter\code\source directory.
- Compile using the procedure in [“Compiling the MyWmSampleAdapter Package” on page 448](#).
- Restart Integration Server.

Note:

If the following error appears when compiling,

`\code\source\com\wm\adapter\wmSampleAdapter\util\DocumentHelper.java uses unchecked or unsafe operations. Recompile with -Xlint:unchecked for details.`

then perform the following:

1. Check if the classes for all Java files are created in code\classes directory.
2. If the classes are created, ignore the error and run the command: `jcode fragall MyWmSampleAdapter`
3. If the classes are not created, redo the steps.

You can use the AccountEnquiry and AccountTransaction adapter service template classes to create adapter service nodes that use any of the banking services including Deposit, GetBalance, Withdraw. At a minimum, configure nodes for Deposit and either Withdraw or Transfer. You must execute these nodes later, when you test the notifications.

The following procedures describe how to configure and test the Deposit adapter service node. To configure and test other nodes, repeat these procedures, substituting the appropriate service name in each node.

Configuring the Adapter Service Nodes

Perform the following procedure to configure the Deposit adapter service node.

Configuring and Enabling a Connection Node

To create a connection node for the Deposit adapter service:

1. Start Integration Server Administrator.
2. Navigate to **Adapters > Sample Adapter**.
The Sample Adapter management screen appears.
3. Select **Connections**.
4. Configure the *sampleConnection* connection node in the *TestMyWmSampleAdapter* package, and enable the connection, as described in [“Configuring the Connection Node” on page 451](#).
5. Navigate to **Package > Management** and reload the *MyWmSampleAdapter* package.

Configuring the Adapter Service Node

To configure a Deposit adapter service:

1. Start Designer.
2. Select **File > Refresh**.
3. In the **Package Navigator**, navigate to the **TestMyWmSampleAdapter** package.
4. Create a *services* folder.
5. Select **File > New**.
6. Select **Adapter Service** from the list of elements.
7. In the **Create a New Adapter Service** screen, type *EnquiryService* in the **Element name** field and click **Next**.
8. In the **Select Adapter Type** screen, select **Sample Adapter** and click **Next**.
9. In the **Select an Adapter Connection Alias** screen, select **connections:sampleConnection** and click **Next**.
10. In the **Select a Template** screen, select **Enquiry** and click **Finish**.

- Repeat the procedure to create another adapter service **DepositService** using the template **Transaction**.

Testing the Adapter Service Nodes

Perform the following procedure to test the adapter service node.

> To test the adapter service node

- Start Designer.
- In the **Package Navigator**, navigate to the **TestMyWmSampleAdapter > services** folder.
- Click the **EnquiryService** service.
- In the **Run** menu, select **Run As > Run Service**.
- Enter data in the fields of the pop-up menu that appears as follows, and click **OK**.

Field	Value
AccountNumber	1

In the **Results** section, the following details appear.

Field	Value
AccountNumber	1
Balance	1000.0

- Click the **DepositService** service.
- In the Adapter Service Editor's **TRANSACTION** tab, select **Deposit** in the **Service Name** field.
- Select **File > Save**.

Note:

To create additional adapter service nodes, repeat this procedure, selecting the appropriate service names, such as **Transfer**, **Withdraw** or **Clear Cheque**.

- In the **Run** menu, select **Run As > Run Service**.
- Enter data in the fields of the pop-up menu that appears as follows, and click **OK**.

Field	Value
ToAccount	1
Amount	29
CheckNumber	0

Note:
Any value less than or equal to 0 specifies a cash deposit. Specify 0 for this test because a check deposit amount is not added to the balance until the check deposit is cleared (approved).

11. Run the **EnquiryService** service with **AccountNumber** value as 1.

In the **Results** section, the following details appear.

Field	Value
AccountNumber	1
Balance	1029.0

Disabling the Phase 3 Implementation

After you compile and test the Phase 3 code, you must disable or delete the `MyWmSampleAdapter` package before you compile and test Phase 4. Not disabling or deleting the package results in a conflict of major codes when you compile and test Phase 4.

Phase 4: Adding Polling Notifications

In this phase, the sample provides a polling notification template that you can configure polling notification nodes to poll the *Sample Server* and determine whether the checks have cleared or bounced, or whether accounts have negative balances. This section describes how to:

- Implement the polling notification template.
- Revise adapter definition, and adapter connection classes.
- Compile the phase 4 implementation.
- Configure two polling notification nodes (one for check clearing/bouncing, the other for negative balances).

Each node generates a document that will be used to contain the affected portion of the *Sample Server* data, and to inform the Integration Server of the changes.

- Create an Integration Server trigger and a flow service for each polling notification node.

The notifications publish the resulting documents to the triggers. Upon receiving a document generated by the polling notification, the trigger causes the Integration Server to invoke a flow service registered with the trigger to process the document's data. In the *Sample Adapter*, the flow service invokes the `pub.flow:savePipelineToFile` service. This service simply saves the contents of the pipeline from the polling notification event to a file. This service is used as a debugging tool. It is provided here simply to demonstrate the use of the notification. In a real adapter, you perform some kind of action with the notification data.

- Schedule and enabling the polling notification nodes.
- Test the polling notification nodes.
- Disable the phase 4 implementation

Implementing the Polling Notification Template

Define a polling notification by extending the `com.wm.adk.notification.WmPollingNotification` base class. The following packages, classes, and methods are added in the *Sample Adapter*; thereby allowing you to configure and monitor events

- A `MessagePolling` class created by extending the `com.wm.adk.notification.WmPollingNotification` base class.

You can monitor the following types of events using the polling notifications:

- `CheckDepositStatusChange`

The *Sample Server* records this event when a Clear Check or Bounce Check service clears (approves) or bounces (disapproves) a deposited check.

- `UnderBalance`

The *Sample Server* records this event when a Withdraw or Transfer service causes a negative account balance. The document is not published if the account's negative balance is greater than the available credit limit amount. For example, if the credit limit amount is 1000, and the account balance falls to -1001, the document is not published; an error is issued.

The bean properties declared in the `MessagePolling` class are as follows:

Property	Description
<code>pollingName</code>	Event to poll in the <i>Sample Server</i> .
<code>inputParameterNames</code>	Fully qualified input parameter names, including all the record structures and array indicators.
<code>inputFieldValues</code>	Unlike adapter services, polling notifications accept no runtime input data, other than the configured property values specified here.
<code>inputFieldTypes</code>	Input parameter data types, including all the array indicators.

Property	Description
<i>outputParameterNames</i>	Fully qualified output parameter names, including all the record structures and array indicators.
<i>outputFieldNames</i>	Fully qualified suggested output parameter signature names, including all the record structures and array indicators. You cannot change these names.
<i>outputFieldTypes</i>	Output parameter data types, including all the array indicators.

The resource domains declared in the `MessagePolling` class are as follows:

Resource Domain	Description
<i>pollingNames</i>	Looks up the list of <i>Sample Server</i> polling notification event names described in “Banking Event Queries” on page 444 .
<i>inputParameterNames</i>	Looks up the fully qualified input parameter names, including all the record structures and array indicators.
<i>inputFieldTypes</i>	Looks up the input parameter data types, including all the array indicators.
<i>outputParameterNames</i>	Looks up the fully qualified output parameter names, including all the record structures and array indicators.
<i>outputFieldTypes</i>	Looks up the output parameter data types, including all the array indicators.

Implementing the `runNotification` Method

The template includes the `runNotification` method, which is called by the Integration Server based on the polling notification node's schedule. This method has no arguments and no return values; it merely publishes documents.

This method constructs an event query document with the query criteria input, and sends it to the *Sample Server*, and then waits for the reply document from the *Sample Server*. The query can produce one of three possible responses:

- Success with output: The notification succeeds, and receives a notification document.
- Success with no output: The notification succeeds, but no event has occurred. The notification receives an acknowledgment document.
- Failure: The notification fails and receives a negative acknowledgment document; an `AdapterException` is thrown with the appropriate error message.

Revising Code From Phases 1, 2, and 3

In Phase 4, the existing classes from Phases 1, 2, and 3 are modified to include the following revisions. The classes contain comments that detail the changes.

Class	Revision
<i>WmSampleAdapterConstants</i>	Added string constants for the polling property names.
<i>WmSampleAdapterResourceBundle</i>	Added entries for the polling property configuration.
<i>WmSampleConnection</i>	<p>In the <code>registerResourceDomain</code> method, the adapter code registers all the resource domains declared by the <code>MessagePolling</code> class.</p> <p>In the <code>adapterResourceDomainLookup</code> method, the adapter code includes resource domain lookup code to request the polling metadata from <i>Sample Server</i> repository.</p>

Compiling the Phase 4 Implementation

To compile the Phase 4 implementation, perform the following:

- Disable the `WmSampleAdapter` and `MyWmSampleAdapter` packages.

Important:

The `WmSampleAdapter` and `MyWmSampleAdapter` packages have the same adapter major code and conflict with each other if they are not disabled.

- Copy all the source code from the `WmSampleAdapter\code\sourcePhase4` directory to the `MyWmSampleAdapter\code\source` directory.
- Compile using the procedure in [“Compiling the MyWmSampleAdapter Package” on page 448](#).
- Restart Integration Server.

Note:

On running the command `jcode makeall MyWmSampleAdapter`, if the error `\code\source\com\wm\adapter\wmSampleAdapter\util\DocumentHelper.java uses unchecked or unsafe operations. Recompile with -Xlint:unchecked for details.` appears, perform the following:

1. Check if the classes for all Java files are created in `code\classes` directory.
2. If the classes are created, ignore the error and run the command: `jcode fragall MyWmSampleAdapter`
3. If the classes are not created, redo the steps.

Configuring and Testing the Polling Notification Nodes

You can create polling notification nodes to send banking event queries to the *Sample Server*. To create these nodes, you use the `MessagePolling` polling notification template. The two banking event query types are:

- `UnderBalance`
- `CheckDepositStatusChange`

The following procedures describe how to configure and test a node for `UnderBalance`. To configure and test a node for `CheckDepositStatusChange`, repeat these procedures, substituting the polling name `CheckDepositStatusChange` for *UnderBalance*.

This section contains the following procedures:

- Configuring and enabling a connection node
- Configuring the `underBalancePolling` notification node
- Creating the Flow Service for the `underBalancePolling` notification node
- Creating the trigger for the `underBalancePolling` notification node
- Scheduling and enabling the `underBalancePolling` notification node
- Testing the `underBalancePolling` notification node

Configuring and Enabling a Connection Node

To create a connection node for the Deposit adapter service:

1. Start Integration Server Administrator.
2. Navigate to **Adapters > Sample Adapter**.

The Sample Adapter management screen appears.

3. Select **Connections**.
4. Configure the `sampleConnection` connection node in the `TestMyWmSampleAdapter` package, and enable the connection, as described in [“Configuring the Connection Node” on page 451](#).
5. Navigate to **Package > Management** and reload the `MyWmSampleAdapter` package.

Configuring the `underBalancePolling` Notification Node

Perform the following procedure to configure the `underBalancePolling` notification node.

- **To configure the `underBalancePolling` notification node**

1. Start Designer.
2. Go to **File > Refresh**.
3. In the **Package Navigator**, navigate to the **TestMyWmSampleAdapter** package.
4. Create a `pollingNotifications` folder.
5. Go to **File > New**.
6. Select **Adapter Notification** from the list of elements.
7. In the **Create a New Adapter Notification** screen, type `underBalancePolling` in the **Element name** field and click **Next**.
8. In the **Select Adapter Type** screen, select **Sample Adapter** and click **Next**.
9. In the **Select a Template** screen, select **Message Polling** and click **Next**.
10. In the **Select an Adapter Connection Alias** screen, select `connections:sampleConnection` and click **Next**.
11. In the **Publish Document Name** screen, click **Finish**.

Designer creates the notification and the publish document named **underBalancePollingPublishDocument**.

12. In the **Package Navigator**, navigate to **underBalancePolling**.
13. In the Adapter Notification Editor's **Message Polling** tab, select the polling event **UnderBalance** from the **Polling Name** field.
14. Specify values for the following fields in the **Input Field Value** column for the corresponding **Input Parameters**, and click **Save**.

Field	Value
User ID	Super user ID. Value is <code>suid</code> .
PIN	Super user PIN. Value is <code>spin</code> .
Account Number	Specify the account number or use <code>0</code> to enable polling against all accounts.

Note:

These values are specified for every account. For more information, see the `SampleServer.properties` property file, located in the `Integration Server_directory \ instances\<instance_name>\packages\WmSampleAdapter\backendResource\doc` directory.

Creating the Flow Service for the underBalancePolling Notification Node

Perform the following procedure to create the flow service for the underBalancePolling notification node.

➤ To create the flow service for the underBalancePolling notification node

1. Start Designer.
2. In the **Package Navigator**, navigate to the **TestMyWmSampleAdapter > pollingNotifications** folder.
3. Go to **File > New**.
4. Select **Flow Service** from the list of elements.
5. In the **Create a New Adapter Service** screen, type `underBalancePollingService` in the **Element name** field and click **Next**.
6. In the **Select the Source Type** screen, select **Empty Flow** and click **Finish**.
7. Click ➤ to insert a flow step.
8. Navigate to the **pub.flow:savePipelineToFile** service.

Note:

The `savePipelineToFile` service saves the polling notification event (the contents of the pipeline) to the file that you specify in the **fileName** parameter.



9. Click the **Pipeline** tab.
10. Open the **fileName** parameter in the pipeline and set its value to `underBalancePolling.log`.
Click **OK**.
11. Click **File > Save**.

Creating the Trigger for the underBalancePolling Notification Node

Perform the following procedure to create the trigger for the underBalancePolling notification node.

➤ To create the trigger for the underBalancePolling notification node

1. Start Designer.

2. In the **Package Navigator**, navigate to the **TestMyWmSampleAdapter > pollingNotifications** folder.
3. Go to **File > New**.
4. Select **webMethods Messaging Trigger** from the list of elements.
5. In the **Create a New webMethods Messaging Trigger** screen, type `underBalancePollingTrigger` in the **Element name** field and click **Finish**.
6. In the trigger editor, in the **Conditions** section, accept the default **Condition1**.
7. In the **Condition detail** section, in the **Service** field, select or type the flow service name `pollingNotifications:underBalancePollingService`.
8. Click  to insert document types. Select **underBalancePollingPublishDocument** and click **OK**.
9. Click  to save your trigger.

Scheduling and Enabling the underBalancePolling Notification Node

Perform the following procedure to schedule and enable the underBalancePolling notification node.

To schedule and enable the underBalancePolling notification node

1. Start Integration Server Administrator.
2. Navigate to **Adapters > Sample Adapter**.
The Sample Adapter management screen appears.
3. Select **Polling Notifications**.
4. Click **Edit Schedule**.
5. Set the **Interval** to 10 and click the **Save Schedule** button.
6. Enable the node by selecting **Enabled** in the **State** column.

Testing the underBalancePolling Notification Node

Invoke a Withdraw service or a Transfer service against an account to cause a negative account balance. You create a Withdraw service or a Transfer service similarly to the Deposit service. Perform the following procedure to test the underBalancePolling notification node.

➤ **Invoke the Transfer service against an account to cause a negative account balance.**

1. Start Designer.
2. In the **Package Navigator**, navigate to the **TestMyWmSampleAdapter > services** folder.
3. Click the **EnquiryService** service.
4. In the **Run** menu, select **Run As > Run Service** .
5. Enter data in the fields of the pop-up menu that appears as follows, and click **OK**.

Field	Value
AccountNumber	1

In the **Results** section, the following details appear.

Field	Value
AccountNumber	1
Balance	1000.0

6. Click the **DepositService** service.
7. In the Adapter Service Editor's **TRANSACTION** tab, select **Withdraw** in the **Service Name** field.
8. Select **File > Save**.

Note:

To create additional adapter service nodes, repeat this procedure, selecting the appropriate service names, such as Transfer, Withdraw or Clear Cheque.

9. In the **Run** menu, select **Run As > Run Service** .
10. Enter data in the fields of the pop-up menu that appears as follows, and click **OK**.

Field	Value
AccountNumber	1

Field	Value
Amount	1020

11. Search “[Creating the Flow Service for the underBalancePolling Notification Node](#)” on page 467.

- The `underBalancePolling.log` file is created in the `Integration Server_directory \ instances\<instance_name>\pipeline` directory for the notification message.
- This is the file name specified in the `fileName` parameter in the flow service.
- This polling notification uses the `savePipelineToFile` service to save the polling notification event to a file.
- A new polling notification event of the same type will overwrite the contents of the corresponding log file.

Note:

No banking event is generated.

Disabling the Phase 4 Implementation

After you compile and test the Phase 4 code, you must disable or delete the `MyWmSampleAdapter` and `TestMyWmSampleAdapter` package before you compile and test Phase 5. Not disabling or deleting the package results in a conflict of major codes when you compile and test Phase 4.

Phase 5: Adding Listener Notifications

In this phase, the sample provides a listener and listener notification template that you can use to configure listener and listener notification nodes. The listener nodes receive alerts immediately from the *Sample Server* when checks are deposited and when the accounts have negative balances. This section describes how to:

- Implement the listener.

A listener object is connected to the adapter resource, waiting for the system to deliver notifications.

- Implement an asynchronous listener notification template.

An asynchronous listener notification publishes a document to a webMethods Broker queue, using the `doNotify` method. You must create a trigger that receives the document and executes an Integration Server flow service to process the document's data.

- Revise adapter definition, and adapter connection classes.
- Compile the phase 5 implementation.

Note:

The Phase 5 implementation is not compiled as the `WmSampleAdapter` package contains the Phase 5 (final) implementation.

- Configure the listener node.
- Configure two asynchronous listener notification nodes (one for check deposits, the other for negative balances).

Each node generates a document that will be used to contain the affected portion of the *Sample Server* data, and to inform the Integration Server of the changes.

- Create an Integration Server trigger and a flow service for each listener notification node.

The notifications publish the resulting documents to the triggers. Upon receiving a document generated by the listener notification, the trigger causes the Integration Server to invoke a flow service registered with the trigger to process the document's data. In the *Sample Adapter*, the flow service invokes the `pub.flow:savePipelineToFile` service. This service simply saves the contents of the pipeline from the listener notification event to a file. This service is used as a debugging tool. It is provided here simply to demonstrate the use of the notification. In a real adapter, you perform some kind of action with the notification data.

- Schedule and enabling the listener notification nodes.
- Test the listener notification nodes.

Implementing the Listener

Define a listener by extending the `com.wm.adk.notification.WmConnectedListener` base class. The following packages, classes, and methods are added in the *Sample Adapter*; thereby allowing you to configure and monitor events

- A `WmSampleListener` class created by extending the `com.wm.adk.notification.WmConnectedListener` base class.

Features of the listener created for *Sample Server* are as follows:

- The listener is a *Sample Server* client that uses a connection in the passive listening mode.
- The listener calls the `retrieveConnection` method to retrieve a connection object that you specify when you configure the listener instead of creating a native connection to the backend system.
- In the `waitForData` method, the connection object waits and return the following: .
 - Returns the notification event if the listener is successful.
 - Returns `null` if the listener times out.

Note:

The time out of the connection while in the blocked reading mode is an important feature of the connection object. This enables the adapter to shut down the listener if it is disabled.

- The `WmSampleListener` class contains no bean properties.

Implementing the Asynchronous Listener Notification Template

Define an asynchronous listener notification template by extending the `com.wm.adk.notification.WmAsyncListenerNotification` base class. The following packages, classes, and methods are added in the *Sample Adapter*; thereby allowing you to configure and take action when the event occur.

- A `AsyncListening` class created by extending the `com.wm.adk.notification.WmAsyncListenerNotification` base class.

You will use the template to configure a node for each of the following alert types:

- `CheckDepositNotification`

The system publishes this notification document when the Deposit service successfully deposits a check. You can then invoke the Clear Check service or the Bounce Check service to approve or disapprove the check.

- `UnderBalanceNotification`

This notification operates under the same criteria as the UnderBalance polling notification.

The following `WmAsyncListenerNotification` methods are implemented:

Method Name	Description
<code>supports</code>	Compares the notification event name with the configured notification name to determine whether it must claim the event or not. Returns <code>true</code> if appropriate, else returns <code>false</code> .
<code>runNotification</code>	Processes the notification event and publishes the event to the webMethods Broker queue.

The `WmSampleListener` class contains no bean properties. The properties declared in the `AsyncListening` class are as follows:

Property	Description
<code>eventName</code>	Event notification type name.
<code>outputParameterNames</code>	Fully qualified output parameter names, including all the record structures and array indicators.
<code>outputFieldNames</code>	Fully qualified suggested output parameter signature names, including all the record structures and array indicators.

Note:
If you set the Boolean flag to `true` in the `createFieldMap` method, the adapter user is provided with the option to overwrite the suggested names. In this implementation, the user cannot change the names.

Property	Description
<i>outputFieldTypes</i>	Output parameter data types, including all the array indicators.

The resource domains declared in the AsyncListening class are as follows:

Resource Domain	Description
<i>notificationNames</i>	Looks up the list of the Sample Server listener notification event names described in “Banking Alerts” on page 445 .
<i>outputParameterNames</i>	Looks up the fully qualified output parameter names, including all the record structures and array indicators.
<i>outFieldTypes</i>	Looks up the output parameter data types, including all the array indicators.

Revising the Code from Phases 1, 2, 3, and 4

In Phase 5, the existing classes from Phases 1 through 4 are modified to include the following revisions. The classes contain comments that detail the changes.

Class	Revision
<i>WmSampleAdapterConstants</i>	Added string constants for the polling property names.
<i>WmSampleAdapterResourceBundle</i>	Added entries for the polling property configuration.
<i>WmSampleConnection</i>	<p>In the <code>registerResourceDomain</code> method, the sample code registers all the resource domains declared by the AsyncListening template.</p> <p>In the <code>adapterResourceDomainLookup</code> method, the sample code includes resource domain lookup code to request the listener notification metadata from the <i>Sample Server</i> repository.</p>

Configuring and Testing the Listener and the Listener Notification Nodes

You can use the AsyncListening listener notification template to create listener notification nodes that monitor the alert types generated by the *Sample Server*. The two alert types are:

- CheckDepositNotification
- UnderBalanceNotification

The following procedures describe how to configure and test a node for CheckDepositNotification. To configure and test a node for UnderBalanceNotification, repeat these procedures, substituting the polling name UnderBalanceNotification for CheckDepositNotification.

This section describes the following tasks:

- Configuring and enabling a connection node
- Configuring the listener node
- Configuring the checkDepositListener notification node
- Creating the flow service for the checkDepositListener notification node
- Creating the trigger for the checkDepositListener notification node
- Enabling the checkDepositListener notification node and the listener node
- Testing the checkDepositListener notification node

Configuring and Enabling a Connection Node

To create a connection node for the Deposit adapter service:

1. Start Integration Server Administrator.
2. Navigate to **Adapters > Sample Adapter**.
The Sample Adapter management screen appears.
3. Select **Connections**.
4. Configure the *listenerConnection* connection node in the *TestMyWmSampleAdapter* package, and enable the connection, as described in [“Configuring the Connection Node” on page 451](#).

Configuring the Listener Node

Perform the following procedure to configure the listener node.

➤ To configure the listener node

1. Start Integration Server Administrator.
2. Navigate to **Adapters > Sample Adapter**.
The Sample Adapter management screen appears.
3. Click **Configure New Listener**.
4. In the **Listener Types** screen, click **Sample Server Listener**.
5. Complete the **Configure Listener Type > Sample Adapter** section as follows:

Parameter	Description
Package	Package in which you create the listener. Select the namespace node package TestSampleAdapter .
Folder Name	Folder in which you create the listener. Type the folder name <code>listeners</code> .
Listener Name	Listener name. Type the listener name <code>sampleListener</code> .
Connection	Select the connection node. For example, connections:listenerConnection .
Retry Limit	Specifies the number of times that the system should attempt to start the listener if the initial attempt fails which is specifying the how many times to retry the <code>listenerStartup</code> method before issuing an <code>AdapterConnectionException</code> . The value <code>0</code> means that the system makes a single attempt. Accept the default value <code>5</code> .
Retry Backoff Timeout	Specifies the number of seconds the system must wait between each attempt to start the listener. Accept the default value <code>10</code> .

- Click **Save Listener**.

Note:

Enabling the listener before you configure and enable its corresponding listener notification node produces a warning.

Configuring the checkDepositListener Notification Node

Perform the following procedure to configure the checkDepositListener notification node.

➤ To configure the checkDepositListener notification node

- In the **Select an Adapter Connection Alias** screen, select **connections:sampleConnection** and click **Next**.
- Start Designer.
- Go to **File > Refresh**.
- In the **Package Navigator**, navigate to the **TestMyWmSampleAdapter** package.
- Create a `listenerNotifications` folder.
- Go to **File > New**.
- Select **Adapter Notification** from the list of elements.

8. In the **Create a New Adapter Notification** screen, type `checkDepositListener` in the **Element name** field and click **Next**.
9. In the **Select Adapter Type** screen, select **Sample Adapter** and click **Next**.
10. In the **Select a Template** screen, select **Asynchronous Listener Notification** and click **Next**.
11. In the **Select an Adapter Notification Listener** screen, select `listeners:sampleListener` and click **Next**.
12. In the **Publish Document Name** screen, click **Finish**.

Designer creates the notification and the publish document named **checkDepositListenerPublishDocument**.

13. Click **Finish**.
14. On the **Listener Notification** tab in the Adapter Notification Editor, select **CheckDepositNotification** in the **Monitor** field.
15. Select **File > Save**.

Creating the Flow Service for the checkDepositListener Notification Node

Perform the following procedure to create the flow service for the checkDepositListener notification node.

➤ To create the flow service for the checkDepositListener notification node

1. Start Designer.
2. In the **Package Navigator**, navigate to the `TestMyWmSampleAdapter > listenerNotifications` folder.
3. Go to **File > New**.
4. Select **Flow Service** from the list of elements.
5. In the **Create a New Adapter Service** screen, type `checkDepositListenerService` in the **Element name** field and click **Next**.
6. In the **Select the Source Type** screen, select **Empty Flow** and click **Finish**.
7. Click ➤ to insert a flow step.
8. Navigate to the `pub.flow:savePipelineToFile` service.

Note:


The `savePipelineToFile` service saves the listener notification event (the contents of the pipeline) to the file that you specify in the **fileName** parameter.

9. Click the **Pipeline** tab.
10. Open the **fileName** parameter in the pipeline and set its value to `checkDepositListener.log`.
Click **OK**.
11. Click **File > Save**.

Creating the Trigger for the checkDepositListener Notification Node

Perform the following procedure to create the trigger for the `checkDepositListener` notification node.

➤ To create the trigger for the checkDepositListener notification node

1. Start Designer.
2. In the **Package Navigator**, navigate to the **TestMyWmSampleAdapter > listenerNotifications** folder.
3. Go to **File > New**.
4. Select **webMethods Messaging Trigger** from the list of elements.
5. In the **Create a New webMethods Messaging Trigger** screen, type `checkDepositListenerTrigger` in the **Element name** field and click **Finish**.
6. In the trigger editor, in the **Conditions** section, accept the default **Condition1**.
7. In the **Condition detail** section, in the **Service** field, select or type the flow service name `listenerNotifications:checkDepositListenerService`.
8. Click  to insert document types. Select **checkDepositListenerPublishDocument** and click **OK**.
9. Click **File > Save** to save your trigger.

Enabling the checkDepositListener Notification Node and the Listener Node

Perform the following procedure to enable the `checkDepositListener` notification node and the listener node.

> To enable the checkDepositListener notification node and the listener node

1. Start Integration Server Administrator.
2. Navigate to **Adapters > Sample Adapter**.
The Sample Adapter management screen appears.
3. Select **Listener Notifications**.
4. Enable the **checkDepositListener** notification by clicking **No** in the **Enabled** column.
The **Enabled** column now shows **Yes** (enabled).
5. Click **Listeners**.
6. Enable the **sampleListener** by selecting **Enabled** in the **State** column.

Testing the checkDepositListener Notification Node

Invoke a Deposit service on an account to produce a check deposit event.

> Invoke the Deposit service on an account to produce a check deposit event.

1. Start Designer.
2. In the **Package Navigator**, navigate to the **TestMyWmSampleAdapter > services** folder.
3. Click the **DepositService** service.
4. In the Adapter Service Editor's **TRANSACTION** tab, select **Deposit** in the **Service Name** field.
5. Select **File > Save**.

Note:

To create additional adapter service nodes, repeat this procedure, selecting the appropriate service names, such as Transfer, Withdraw or Clear Cheque.

6. In the **Run** menu, select **Run As > Run Service**.
7. Enter data in the fields of the pop-up menu that appears as follows, and click **OK**.

Click  to add a row and specify values for the field as follows:

Field	Value
Deposit To Account	1
Deposit Amount	10
Check Number	1

The data you entered appears in the **Results** tab.

- The `checkDepositListener.log` file is created in the *Integration Server_directory* \ *instances* \ *<instance_name>* \ *pipeline* directory for the check deposit notification message.
- This is the file name specified in the *fileName* parameter in the flow service.
- This listener notification uses the `savePipelineToFile` service to save the polling notification event to a file.
- A new listener notification event of the same type will overwrite the contents of the corresponding log file.

