Informatica

Informatica® Development Platform
10.2 HotFix 2

# Developer Guide

Informatica Development Platform Developer Guide
10.2 HotFix 2
April 2019

Publication Date: 2019-04-11

# Table of Contents

# Preface

The *Informatica Development Platform Developer Guide* provides information about the APIs and SDKs available in the Informatica Development Platform and how to use them to develop adapters and plug-ins for PowerCenter®. It provides tutorials and examples you can use when you develop your adapters and plug-ins. The *Developer Guide* is written for independent software vendors, consulting organizations, and developers who want to use the Informatica Development Platform to develop adapters to integrate PowerCenter with other applications.

This guide assumes you have a working knowledge of PowerCenter and are familiar with application programming interfaces.

# Informatica Resources

## Informatica Network

Informatica Network hosts Informatica Global Customer Support, the Informatica Knowledge Base, and other product resources. To access Informatica Network, visit https://network.informatica.com.

As a member, you can:

- Access all of your Informatica resources in one place.
- Search the Knowledge Base for product resources, including documentation, FAQs, and best practices.
- View product availability information.
- Review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

## Informatica Knowledge Base

Use the Informatica Knowledge Base to search Informatica Network for product resources such as documentation, how-to articles, best practices, and PAMs.

To access the Knowledge Base, visit https://kb.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

## Informatica Documentation

To get the latest documentation for your product, browse the Informatica Knowledge Base at https://kb.informatica.com/_layouts/ProductDocumentation/Page/ProductDocumentSearch.aspx.

If you have questions, comments, or ideas about this documentation, contact the Informatica Documentation team through email at infa_documentation@informatica.com.

## Informatica Product Availability Matrixes

Product Availability Matrixes (PAMs) indicate the versions of operating systems, databases, and other types of data sources and targets that a product release supports. If you are an Informatica Network member, you can access PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

## Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services. Developed from the real-world experience of hundreds of data management projects, Informatica Velocity represents the collective knowledge of our consultants who have worked with organizations from around the world to plan, develop, deploy, and maintain successful data management solutions.

If you are an Informatica Network member, you can access Informatica Velocity resources at http://velocity.informatica.com.

If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

## Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that augment, extend, or enhance your Informatica implementations. By leveraging any of the hundreds of solutions from Informatica developers and partners, you can improve your productivity and speed up time to implementation on your projects. You can access Informatica Marketplace at https://marketplace.informatica.com.

## Informatica Global Customer Support

You can contact a Global Support Center by telephone or through Online Support on Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link: http://www.informatica.com/us/services-and-training/support-services/global-support-centers.

If you are an Informatica Network member, you can use Online Support at http://network.informatica.com.

CHAPTER 1

# Introduction to PowerCenter Data Integration

This chapter includes the following topics:

## Introduction to PowerCenter Data Integration Overview

As the volume, variety, and need to share data increases, more and more enterprise applications require data integration capabilities. Data integration requirements can include the following capabilities:

- The ability to access and update a variety of data sources in different platforms
- The ability to process data in batches or in real time
- The ability to process data in different formats and transform data from one format to another, including complex data formats and industry specific data formats
- The ability to apply business rules to the data according to the data format
- The ability to cleanse data to ensure the quality and reliability of information

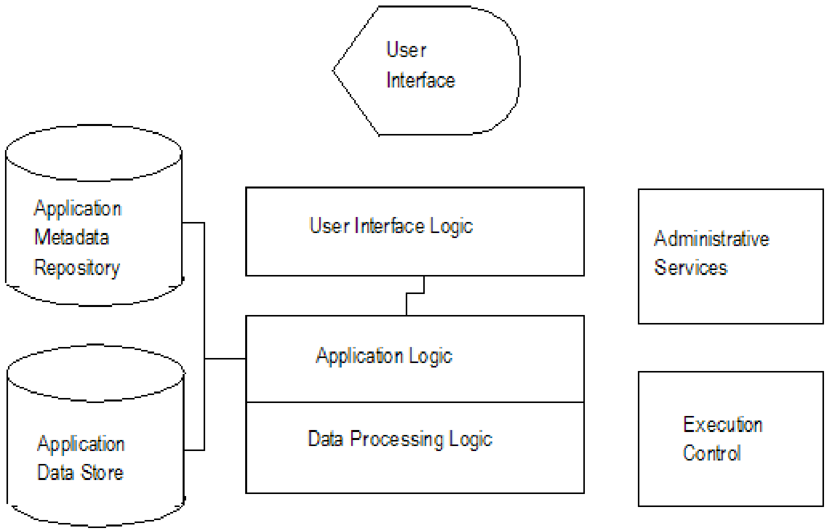
Application development can become complex and expensive when you add data integration capabilities to an application. Data issues such as performance and scalability, data quality, and transformation are difficult to implement. Applications fail if the these complex data issues are not addressed appropriately.

PowerCenter data integration provides application programming interfaces (APIs) that enable you to embed data integration capabilities in an enterprise application. When you leverage the data processing capabilities of PowerCenter data integration, you can focus development efforts on application specific components and develop the application within a shorter time frame.

PowerCenter data integration provides all the data integration capabilities that might be required in an application. In addition, it provides a highly scalable and highly available environment that can process large volumes of data. An integrated advanced load balancer ensures optimal distribution of processing load. PowerCenter data integration also provides an environment that ensures that access to enterprise data is secure and controlled by the application. It uses a metadata repository that allows you to reuse processing logic and can be audited to meet governance and compliance standards.

# Enterprise Application Components

The following figure shows the common logical components of a typical enterprise application:



An enterprise application typically has the following components:

- **User interface logic.** Controls the user interface for the application. This component works with the application logic component to carry out end-user functions and utilities.
- **Application logic.** Controls the core business rules and processes for the application and determines the application behavior. The application logic works with the user interface layer to drive user interactions and control the user interface of the application. It also interfaces with the data processing layer to carry out the data processing functions of the application.
- **Data processing logic.** Performs all the data interactions of the application. The data processing logic can be tightly embedded with the application logic or spread across different layers. The data processing layer performs the following services:
  - Accesses and updates the application data stores and other external data sources on which the application depends.
  - Transforms data into various formats. It can transform data from external data formats to application specific data formats or from application specific data formats to external data formats.
  - Fixes errors in data and verifies the quality of data received from the user interfaces and other data adapters.
- **Application metadata repository.** Contains the application metadata that drives application behavior. The metadata repository can be a database or a set of configuration files. The application data catalog, which includes descriptions of application and external data structures, is typically stored in the metadata repository.
- **Application data store.** Contains the data required by the application and is stored in a relational database, XML files, or other types of data storage or format. The data processing logic accesses the application data store through SQL, web services, or APIs that provide access to the application data.
- **Administrative component.** Provides administrative services for the application, including application setup and configuration, user security and management, data access security, deployment and migration, and backups.

- **Execution control.** Drives the operation and application monitors for the system. This component invokes and monitors data processes, either through a schedule or manual execution.

Depending on the implementation of an application, these components can be combined or further broken up into smaller components.

# Embedded Data Integration

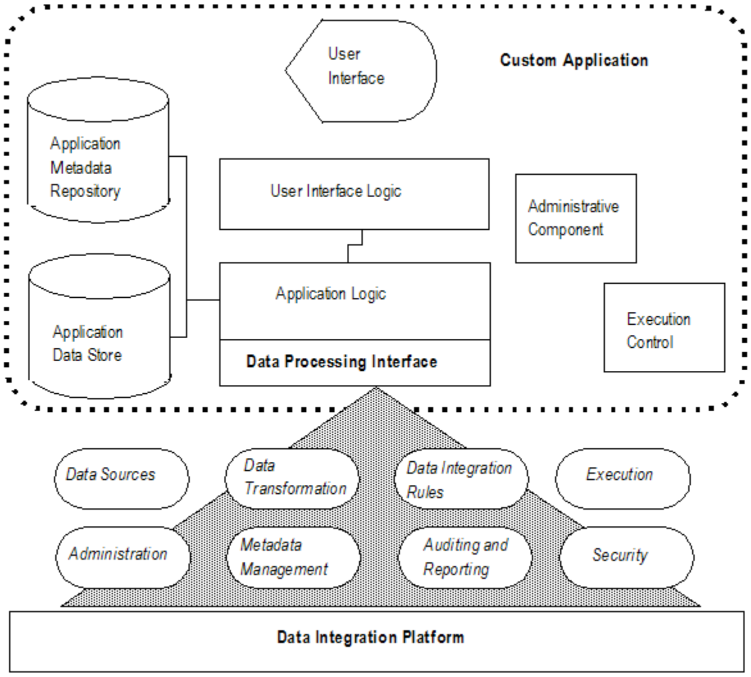
The enterprise application must be able to control application behavior and data processing functions. If an application uses a third-party data integration component to perform data processing functions, the application must be able to embed the third-party component within its own processes. The application must be able to invoke and control any functionality provided by the third-party data integration component through the application logic. Likewise, the application user interface must be able to expose and handle the functions provided by the third-party data integration component.

For an enterprise application to successfully integrate a third-party data integration component into its logic, the application logic must be able to manage and control the data integration functions on multiple levels.

PowerCenter data integration provides management and control at the following levels:

- **Data source access.** PowerCenter provides adapters to access common data sources such as relational databases, flat files, XML, and mainframe data sources. Additionally, PowerCenter data integration provides a way to extend connectivity to custom data sources or data stores specific to the application.
- **Transformation processing.** PowerCenter data integration manages the data transformation requirements of an application. It also provides a way to extend the transformation capabilities of an enterprise application to include plugs-ins that handle application specific transformations.
- **Data processing rules and metadata management.** Typically, an application uses configuration files to control application behavior. PowerCenter data integration provides interfaces to allow the application to control the data processing logic through configuration files or the user interface. In addition, PowerCenter provides an interface for the application to correlate application-specific metadata with the data integration metadata. This permits a single point of maintenance for all the metadata.
- **Execution.** PowerCenter data integration provides an interface to allow the application to invoke, monitor, and control the execution of data integration processes. It can capture run-time statistics and provide reports on the status of the processes. PowerCenter also provides a web service interface that allows the application to invoke external web services and extend the data integration capabilities with standard web services technology.
- **Security and access control.** PowerCenter data integration supports enterprise application security protocols for network connections and data transmission to ensure security in application and data access. It establishes application level permissions and restrictions to drive user access to the application functionality and data.
- **Administration.** An application must be able to administer data integration functions and processes, including installation and configuration, user administration, backup, and migration. PowerCenter data integration provides an interface to allow administration of the data integration process through its application interface or through an external application.
- **Auditing and reporting.** PowerCenter data integration provides interfaces to access information about changes to operational metadata. It provides a reporting interface to operational information and statistics such as date and time the last data load was performed or the number of rows of data processed.

The following figure shows the services that a data integration platform should provide through a data integration interface to control the application components.



# PowerCenter Data Integration

PowerCenter provides programming interfaces that enable you to access its data integration functionalities and manage and control all aspects of application behavior. You can use the interfaces to embed data integration capabilities in your application.

The following table summarizes the capabilities provided by PowerCenter through various programming interfaces:

| Capability | Interface | Description |
|---|---|---|
| Extensible data sources | PowerExchange® API | Provides connectivity to custom data sources and data formats. Available in Java and C++. |
| Extensible transformation processing | Transformation API | Allows you to invoke application specific APIs to extend transformation processing. Available in Java and C. |
| Application driven data integration logic | Design API | Allows you to control data integration logic in the application and to create metadata for PowerCenter objects without a user interface. |
| Execution control and monitoring | - Operations API<br>- Command line interface (*pmcmd*)<br>- Batch Web Services | Allows you to drive execution and monitoring of PowerCenter integration processes through an API, the command line, or web services. |

| Capability | Interface | Description |
|---|---|---|
| Security and access control | Command line interface (*pmrep*) | Allows you to administer PowerCenter user accounts and manage application connections. |
| Administration | Command line interface (*pmrep*) | Enables you to administer the data integration metadata, perform backups, and migrate data integration components across environments. |

CHAPTER 2

# PowerCenter Interfaces

This chapter includes the following topics:

## PowerCenter Interfaces Overview

This chapter describes the PowerCenter interfaces and how you can use them to embed the capabilities of PowerCenter data integration into your application.

You can use the following types of interfaces to embed PowerCenter data integration capabilities in your enterprise application:

- **Informatica Development Platform.** Includes all PowerCenter APIs and SDKs.
- **Command line programs.** Commands to manage PowerCenter workflows and administer the repository and domain services.
- **Web services.** Web service operations that allow access to the Integration Service processes and repository metadata.

### Informatica Development Platform

The Informatica Development Platform makes the PowerCenter application programming interfaces (APIs) and software development kits (SDKs) available to enable you to integrate PowerCenter functionality in any application.

The following application APIs and SDKs comprise the Informatica Development Platform:

- **PowerExchange API.** Create plug-ins for PowerCenter to extend its functionality.
- **Transformation API.** Create custom transformations that call procedures outside of PowerCenter.
- **Operations API.** Access the Integration Service from a Java or C application and manage workflow, task, reporting, and log requests.

- **Design API.** Generate PowerCenter metadata and XML documents containing mappings, sessions, and workflows.
- **Custom Function API.** Develop functions written in C and add them to the Expression and Aggregator transformations.

### Installation

You can install the Informatica Development Platform from the following sources:

- **Informatica Development Platform installation DVD.** Run the Informatica Development Platform installer to install the PowerCenter APIs and SDKs. You can install all the SDKs or install only the SDKs that you want to use. To install all the SDKs in one process, select the Complete installation option. To install specific SDKs, select the Custom installation option.
- **Informatica electronic software download site.** When you purchase PowerCenter and choose to download the software, you receive a site link, user ID, and password to access the Informatica electronic software download site. Follow the instructions in the download site to download the Informatica Development Platform installation file.
- **Informatica Technology Network.** If you are a registered user of the Informatica Technology Network, you can download the Informatica Development Platform installation file from the Informatica Development Platform page. When you download the file, the Informatica Development Network provides you with a password. Use this password when you extract the files from the download file.

For more information about running the Informatica Development Platform installer, see the *Informatica Installation and Configuration Guide*.

## Command Line Programs

PowerCenter provides a number of command line programs that you call from your application to manage the Integration Service and Repository Service.

To control data integration processes and manage the repository metadata from your application, use the following command line programs:

- ***pmcmd*.** Use *pmcmd* to manage workflows. You can use *pmcmd* to start, stop, schedule, and monitor workflows. This command enables you to manage the services in the PowerCenter domain from an external application.
- ***pmrep*.** Use *pmrep* to perform repository administration tasks such as listing repository objects, creating and editing groups, and restoring and deleting repositories. This command enables you to manage the PowerCenter repository from an external application.

The PowerCenter installation includes the command line programs. After you install PowerCenter, you can use the command line programs to manage PowerCenter services and repositories from any machine in the PowerCenter environment.

## Web Services

The Web Services Hub is available in the PC domain. The Web Service Hub is a web service gateway that allows a client application to use web service standards and protocols to access PowerCenter functionality.

The Web Services Hub enables you to turn PowerCenter workflows into web services. You can manage data integration processes within the PowerCenter framework through requests to PowerCenter web services.

The Web Services Hub also provides web service operations that allow you to monitor and control PowerCenter processes and get repository information.

The PowerCenter installation includes the Web Services Hub. After you install PowerCenter, use the Administration Console to create a Web Services Hub. Configure workflows to run as web services on the Web Services Hub.

# Informatica Development Platform Application APIs and SDKs Support

Informatica Development Platform application APIs and SDKs does not support forward compatibility. For example, you cannot use the IDP libraries developed for any version of 10.x to connect to the PowerCenter server of any version of 9.6.1.x.

Informatica Development Platform application APIs and SDKs supports backward compatibility.

The following table lists the backward compatibility for each version of the IDP libraries and PowerCenter servers:

| IDP Libraries | PowerCenter Servers |
|---|---|
| 9.6.1.x | 9.6.1.x, 10.0, 10.1, 10.1.1, 10.1.1 HotFix 1, 10.1.1 HotFix 2 |
| 10.0 | 10.0, 10.1, 10.1.1, 10.1.1 HotFix 1 , 10.1.1 HotFix 2 |
| 10.1 | 10.1, 10.1.1, 10.1.1 HotFix 1, 10.1.1 HotFix 2 |
| 10.1.1 | 10.1.1, 10.1.1 HotFix 1, 10.1.1 HotFix 2 |
| 10.1.1 HotFix 1 | 10.1.1 HotFix 1, 10.1.1 HotFix 2 |
| 10.1.1 HotFix 2 | 10.1.1 HotFix 2 |
| 10.2 | 10.2 |
| 10.2 HotFix 1 | 10.2 HotFix 1 |

**Note:** Informatica recommends that you use the same version of the IDP libraries and PowerCenter server to connect.

# PowerExchange API

**Note:** It is recommended to use the Informatica Connector Toolkit instead of PowerExchange API to create adapters for PowerCenter. For information about creating adapters by using Informatica Connector Toolkit, see the *Informatica Connector Toolkit Developer Guide*.

The PowerExchange API includes interfaces to the PowerCenter Client, Integration Service, and the PowerCenter repository. Use the PowerExchange API to create custom adapters to extend PowerCenter functionality.

You can modify and extend the PowerCenter functionality in the following ways:

- Create adapters for new sources and targets.
- Modify and add to the Designer interface.
- Handle data exceptions from new sources and targets and write to the session logs.

The PowerExchange API is available in Java and C++.

# Usage

Use the PowerExchange API to build custom reader and writer adapters for new data sources. You can create the following types of adapters to extend PowerCenter capabilities:

- **Adapters for database appliances.** Typically, database appliances provide ODBC or JDBC adapters and provide bulk load and extract utilities. You can use the PowerExchange API to build custom connectors to seamlessly invoke the bulk load and extract utilities from the data integration processes.
- **Adapters for ERP and CRM applications.** ERP, CRM, and other custom applications typically provide APIs, web services, and other interfaces to the application data stores. Some applications may use proprietary data formats. For other applications, you may not be able to access the data store tables except through the applications. Use the PowerExchange API to build a connector to the applications and invoke the application API methods.
- **Adapters for messaging middleware.** Some enterprises may deploy a messaging middleware to allow communication between applications. If the messaging middleware does not use standard messaging protocols such as JMS, you can use the PowerExchange API to build adapters to read and publish messages for the middleware.

# Requirements

When you use the PowerExchange API to develop a plug-in, complete the following requirements:

- Get the repository ID attributes for the plug-in.
- Define the metadata for the plug-in.
- Register the metadata for the plug-in.

## Repository ID Attributes

Before you develop a plug-in using the PowerExchange API, contact Informatica to obtain the PowerCenter repository ID attributes for the plug-in. Informatica assigns unique repository ID attributes to each plug-in.

If you develop a plug-in that will not be distributed outside your organization, you can define the repository ID attributes without contacting Informatica. You can set the repository ID attributes to the test values. When you distribute the plug-in outside your organization, contact Informatica to get the repository ID attributes. You cannot use repository ID attributes that conflict with those of another vendor.

## Plug-in Metadata

The repository ID attributes is the metadata of the plug-in. Create an XML file to contain the plug-in metadata. The PowerExchange API installation includes a sample metadata definition file named sdkdemo.xml. You can use the sdkdemo.xml file as a template to define the metadata for the plug-in.

### Metadata Registration

After you create the metadata definition file for the plug-in, register the metadata with the PowerCenter repository. Use the Administration Console to register the plug-in metadata with each repository where you plan to use the plug-in.

If you create a plug-in that modifies the PowerCenter Client, you must also register the plug-in metadata with the client machine. Register the plug-in in the Windows Registry on the client machine so that the Designer can load the plug-in library file.

# Transformation API

Use the Transformation API to create custom transformations that call procedures outside of PowerCenter. You can include the new custom transformation in a PowerCenter mapping as you would other transformations. You can build transformations that provide specific functions not included in other transformations provided by PowerCenter. For example, you need the PowerCenter workflow to interface with another application. You can write a C or Java program that interfaces with the application and performs the functions you require. Then use the methods and functions available in the Transformation API to turn the program into a PowerCenter transformation.

The Transformation API is available in Java andC.

## Usage

You can use the Transformation API to create transformations that invoke functions in external libraries. Use the Transformation API to add custom data processing capabilities to PowerCenter, such as geospatial analytical functions and statistical or mathematical functions. Create custom transformations with functions that process multiple rows of data or hierarchical data objects.

# Operations API

Use the Operations API to issue commands to the Integration Service from a third-party application. You can use the Operations API to manage the Integration Service and run or monitor workflows from a third-party application. You can get performance data and monitor the progress of a session as it runs or get details of workflows and sessions that have completed their runs. For example, you can run and monitor PowerCenter workflows and tasks using an external scheduler such as HP OpenView or an SNMP system.

Use the Operations API to perform the following types of tasks:

- **Connecting to the Integration Service.** Access the Integration Service.
- **Running and managing workflows and tasks.** Schedule, run, stop, or get details about workflows. If you want to run only part of a workflow, you can start a workflow from one of its tasks. You can also start, stop, or get details about individual tasks in a workflow
- **Monitoring and reporting.** Get details about the Integration Service, performance data on a running session, or details of the workflow or session that ran last.
- **Viewing logs.** View workflow and session logs.
- **Error handling.** Handle errors and warnings encountered when you run workflows or sessions.

The Operations API is available in Java and C. A subset of the methods available in the Operations API are also available as web service operations. You can call the web service operations from web service clients or Business Process Execution Language (BPEL) engines to perform the same tasks.

## Usage

You can use the Operations API to manage PowerCenter workflows and tasks in the following ways:

- **Integrate PowerCenter with external schedulers** Use the Operations API to add scheduling and monitoring capabilities to PowerCenter to provide more control over the execution of workflows and tasks. Likewise, you can use the Operations API to run PowerCenter workflows and tasks from external schedulers, enterprise monitoring applications, or Business Process Execution Language (BPEL) engines.
- **Control PowerCenter workflows and tasks from applications.** For enterprise applications with embedded PowerCenter capabilities, use the Operations API to manage and run workflows and tasks based on events and processes completed in the application. You can also use the Operations API to run workflows and tasks in response to requests made through the application user interface.
- **Automate execution of PowerCenter workflows and tasks**. Use the Operations API to create programs or scripts that automate control and execution of PowerCenter workflows and tasks.

The functionalities provided by the Operations API are also available through the PowerCenter command line programs. You can also implement a subset of the functionalities through web services.

# Design API

Use the Design API to create metadata for PowerCenter objects without a user interface. Create, read, and write objects in the PowerCenter repository, including sources, targets, transformations, mappings, sessions, and workflows. You can use the Design API to build PowerCenter mappings and workflows without using the PowerCenter Client tools. This allows you to use a custom application to build PowerCenter metadata or to build PowerCenter metadata based on metadata from other applications. You can also use the Design API to access PowerCenter objects from a user interface that matches the look and feel of another application.

The Design API is available in Java.

## Usage

You can use the Design API to read and write metadata in the PowerCenter repository in the following ways:

- **Create PowerCenter design objects from a custom interface.** Applications with embedded PowerCenter data integration capabilities often require that the user interface that calls PowerCenter processes match the user interface of the rest of the application. This provides a consistent user interface to end users. You can develop a user interface with the look and feel of the application and use the Design API to read and write PowerCenter metadata from the new user interface. For example, a CRM application with embedded PowerCenter data integration capabilities needs to generate the data integration logic without using the PowerCenter Client tools. You can use the Design API to programmatically generate the data integration logic for workflows and tasks and the runtime configuration objects required to run the workflows and tasks.
- **Administer PowerCenter mappings, transformations and workflows from an application.** Use the Design API to access objects in the PowerCenter repository and enable monitoring and reporting from an external administrative application.

- **Build add-on utilities for PowerCenter.** Use the Design API to build utilities such as mapping generators or test generators to increase user productivity. For example, you can use the Design API to generate multiple mappings based on user input and speed up mapping development.

# Custom Function API

Use the Custom Function API to add custom functions to the Expression and Aggregator transformations.

The Custom Function API is available in C++.

## Usage

You can use the Custom Function API to create custom functions such as encryption, statistical, and scientific functions for the Expression and Aggregator transformations. Create functions that operate on one or more scalar data attributes in a single data row. For example. you can create a custom encryption function that takes a string attribute as input and produces a binary value.

You can include custom functions in PowerCenter expressions that you add to a transformation.

# CHAPTER 3

# Developing a PowerExchange Adapter

This chapter includes the following topics:

## Developing a PowerExchange Adapter Overview

**Note:** It is recommended to use the Informatica Connector Toolkit instead of PowerExchange API to create adapters for PowerCenter. For information about creating adapters by using Informatica Connector Toolkit, see the *Informatica Connector Toolkit Developer Guide*.

You can use the PowerExchange API to build an adapter to extend data connectivity for PowerCenter. A common purpose for an adapter is to extract data from application systems or other data sources. The PowerExchange API has been used to develop adapters for PowerCenter, including adapters to extract data from customer relationship management (CRM) systems such as the PowerExchange for Siebel, adapters to extract data from real-time data sources such as PowerExchange for JMS and PowerExchange for MSMQ, and adapters to extract data from on-demand business applications such as PowerExchange for Salesforce.com.

An adapter can consist of one or more plug-ins, including a server plug-in and a client plug-in. When you use the PowerExchange API to develop a PowerCenter adapter for distribution, each plug-in that is part the adapter must have a unique identifier to distinguish the adapter from other PowerCenter adapters. Contact Informatica to obtain a unique identifier for your adapter. A plug-in must also have an associated plug-in definition file that contains the unique identifier assigned to the plug-in and other properties of the plug-in. Use the plug-in definition file to register the plug-in with a PowerCenter repository.

This chapter discusses the steps to develop a PowerCenter plug-in with the PowerExchange API.

# Step 1. Get the PowerCenter Repository ID Attributes

Before you develop a plug-in, email Informatica at devnet@informatica.com to get the PowerCenter repository ID attributes for the plug-in. Informatica assigns unique repository ID attributes to each PowerCenter plug-in. Use these repository ID attributes to identify the plug-in when you define the metadata for the plug-in.

If you develop a plug-in to be distributed only within your organization, you can define the repository ID attributes without contacting Informatica. You can use temporary test values for the attributes. For example, you develop and test a plug-in before using it in a production environment. You can set the repository ID attributes to the test values listed in "Step 1. Get the PowerCenter Repository ID Attributes" on page 22. If you develop multiple plug-ins, each plug-in must have unique repository ID attribute values.

The following table describes the repository ID attributes that define a plug-in:

| Repository ID Attribute | Description | Test Values |
|---|---|---|
| Vendor ID | Identifies the vendor that developed the plug-in. This value corresponds to the VENDORID attribute for the PLUGIN element in the plug-in definition file. | If you do not have a vendor ID, use 2001. |
| dbType | Identifies the database type for the application. This value corresponds to the ID attribute for the DBTYPE element in the plug-in definition file. | Use any value from 200,000 to 299,900, in increments of 100. |
| Datatype range | Provides a range of IDs that the vendor can associate with each datatype for the database type. You can use the values in this range in the ID attribute for the DATATYPE element in the plug-in definition file. | Use any value from the range dbType to dbType + 99, in increments of 1. For example, if you set the dbType repository ID attribute to 250,000, you can use any value from 250,000 to 250,099 for the datatype. |
| Extension subtype | Associates an ID with each reader and writer. This value corresponds to the EXTENSIONSUBTYPE attribute for the EXTENSION element in the plug-in definition file. | Use any value from the range dbType to dbType + 99, in increments of 1. For example, if you set dbType to 250,000, you can use any value from 250,000 to 250,099 for subtype. |
| Connection subtype | Associates an ID with the PowerCenter connection to the third-party application. This value corresponds to the CONNECTIONSUBTYPE attribute for the CONNECTION element in the plug-in definition file. | Use any value from the range dbType to dbType + 99, in increments of 1. For example, if you set dbType to 250,000, you can use any value from 250,000 to 250,099 for subtype. |
| Metadata extension domain ID | Groups metadata extensions into one domain. This value corresponds to the ID attribute for the MEDOMAIN element in the plug-in definition file. | Use any value from the range dbType to dbType + 99, in increments of 1. For example, if you set dbType to 250,000, you can use any value from 250,000 to 250,099 for the domain ID. |

**Note:** It is important that you obtain globally unique repository ID attributes from Informatica for your plug-in if it will be distributed outside your organization. Repository ID attributes are invalid if they conflict with those of another vendor. Invalid repository ID attributes will make your plug-in components unusable.

# Step 2. Define the Metadata for the Plug-In

The repository ID attributes comprises the metadata for the plug-in. The plug-in metadata is stored as elements in an XML file.

Create an XML file that contains the repository ID attributes defined for the plug-in. Give the XML file a name to associate with your plug-in. You can create an XML file called <*PluginName*>.xml and add the repository ID attributes for the plug-in as elements in the XML file.

The PowerExchange API installation includes a sample plug-in definition file named sdkdemo.xml. The sdkdemo.xml file includes the elements and attributes that are required to define a database type. You can use the sdkdemo.xml file as a template to set up the database type definition for your plug-in. The sdkdemo.xml is installed in the following directory:

```
<PowerExchangeAPIInstallDir>/samples
```

When you register a plug-in definition in the PowerCenter repository, PowerCenter uses a Document Type Definition (DTD) file called plugin.dtd to validate the XML file. The PowerCenter installation includes the plugin.dtd file, installed in the PowerCenter Client directory. The plugin.dtd file defines the elements and attributes you can use in the XML file. When you create or modify the XML file for your plug-in, verify that it conforms to the structure of the plugin.dtd file.

# Step 3. Register the Plug-in Metadata

After you create the metadata definition file, you must register the plug-in with a PowerCenter repository. Use the Administration Console to register the plug-in with each PowerCenter repository where you want to use the plug-in.

When you register a plug-in with a repository, the Repository Service must be running in exclusive mode.

To register a PowerExchange plug-in:

1. In the Navigator of the PowerCenter Administration Console, select the Repository Service to which you want to add the plug-in.
2. Run the Repository Service in exclusive mode.
3. Click the Plug-ins tab.
4. Click the link to register a Repository Service plug-in.
5. On the Register Plugin for <*RepositoryService*> page, click Browse to locate the plug-in file.
6. If the plug-in was registered previously and you want to overwrite the registration, select the option to update the existing plug-in registration.
7. Enter your repository user name and password.
8. Click OK.

   The Repository Service registers the plug-in with the repository. The results of the registration operation appear in the activity log.
9. Run the Repository Service in normal mode.

# Step 4. Register the Client Plug-in in the Windows Registry

Before using a PowerCenter Client plug-in, register the plug-in with the Windows Registry on the PowerCenter Client machine. When you register the plug-in in the Windows Registry, the Designer is able to load the <*PluginName*>.dll file.

Use a REG file to register a plug-in with the Windows Registry. The PowerExchange API installation includes a sample REG file named sdk.reg. You can use the sdk.reg file as a template to create a REG file to register a client plug-in. The sdk.reg file adds an entry for the SDKDemo plug-in in the Windows Registry.

You can also register the plug-in manually. For example, to register the sdkdemocli.dll client plug-in, set "SDKDEMO"="sdkdemocli.dll" at the following location in the Windows Registry:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Informatica\PowerCenter Client Tools\<Version>\PlugIns
\<VendorName>
```

**Note:** You only need to register PowerCenter Client plug-ins in the Windows Registry. You do not need to register plug-ins for the Integration Service in the Windows Registry.

# Step 5. Set Up the Development Environment

After you define and register plug-in metadata in the PowerCenter repository and Windows Registry, set up your development environment.

To set up the development environment, complete the following tasks:

- Define the path for the plug-in DLL
- Select a compiler

## Defining the Path for the DLL

Before you build the client or server plug-ins, specify the path for the dynamic link libraries or shared objects that you will use. Specify the path in the environment variable appropriate for your development platform.

When you develop a plug-in in Java, set the CLASSPATH environment variable to include the absolute path of the folder where the pmserversdk.jar file is located. By default, the pmserversdk.jar file is located in the following directory:

```
<IDPInstallDir>/<PWXAPIInstallDir>/javalib
```

When you develop a plug-in in C++, set the path in one of the following environment variables, based on your development platform:

| Operating System | Environment Variable |
|---|---|
| Windows | PATH |
| Solaris, Linux | LD_LIBRARY_PATH |
| AIX | LIBPATH |

## Selecting a Compiler

When you build plug-ins for PowerCenter, select a compiler that is compatible with your development platform.

When you develop a plug-in in Java, use the compiler for the version of Java installed with PowerCenter.

The following table describes the Java compiler you can use based on the PowerCenter version:

| PowerCenter Version | Compiler |
|---|---|
| PowerCenter 8.61 | Java 1.5.0_11 |
| PowerCenter 9.x<br>PowerCenter 9.6.x | Java 1.6<br>Java 1.7.x<br>**Note:** 9.6.1 HotFix 4 uses Java 1.8.x for runtime. |
| PowerCenter 10.x | Java 1.7.x |
| PowerCenter 10.1.x | Java 1.8.x |

When you develop a plug-in in C++, use a compiler based on the operating system on which PowerCenter is installed.

The following table describes the C++ compiler you can use based on the operating system:

| Operating System | Compiler |
|---|---|
| Windows x86 or x64 | Visual Studio .NET 2013 |
| Solaris SPARC or Solaris x64 | Sun C++ 5.11 2010/08/13 |
| AIX PPC 64 | XL C/C++ for AIX, Version 12.1 |
| Linux RedHat x86 or x64<br>SuSe Linux 11 x64 | gcc version 4.4.7 20120313 (Red Hat 6.5)<br>gcc version 4.1.2 20080704 (Red Hat 5.7) |

# Step 6. Build Server and Client Plug-ins

To read data from the application source or write data into the application target, build a server plug-in. You can develop server plug-ins on Windows or UNIX.

To modify the PowerCenter Designer interface to support your application source or target, build a Client plug-in. The Designer runs only on Windows, so you can build Client plug-ins only on Windows.

## Compiling the DLL on Windows

You must use Visual C++ to build a plug-in on Windows.

To build a DLL on Windows:

1. Start Visual C++.

2. Choose File-Open.
3. Locate the sdkdemocli.sln file in the sdkdemocli directory in the PowerExchange API installation.
4. Click OK to open the sdkdemocli.sln solution.
5. Click Tools-Options.
6. Click the Directories tab.
7. Select Include Files for the Show Directories for option.
8. Add the following directories by clicking the Add button:
   - `<` *`PowerExchange APIInstallDir`* `>\Include`
   - `<` *`PowerExchange APIInstallDir`* `>\Include\sdkclient`
   - `<` *`PowerExchange APIInstallDir`* `>\Include\sdkcli`
   - `<` *`PowerExchange APIInstallDir`* `>\Include\sdksrv`
9. Select Library Files for the Show Directories option.
10. Click Add to add the PowerExchange API /lib directory.
11. Click OK.
12. Choose Build or press F7 to compile the plug-in.

## Debugging the Plug-in

While developing a plug-in to connect PowerCenter to a third-party application, you may need to debug the plug-in. Debugging a PowerCenter Client plug-in differs from debugging a Integration Service plug-in.

To debug a PowerCenter Client plug-in:

1. In Visual C++, click File > Open Workspace to open a solution.
2. Click Project > Properties.
3. Set the build type to release.
4. Click the C/C++ tab.
5. Select “Program Database for Edit and Continue” for the Debug info option.
6. Select “Default” for the Optimizations option.
7. Click the Link tab.
8. Select Debugging for the Category option.
9. Select the Generate Yes (/DEBUG) option.
10. Click OK.
11. Choose Build to build the plug-in.
12. Set a breakpoint to specify where to start debugging.
13. Choose Build-Execute and specify pmdesign.exe as the name of the executable file.

    Program execution stops in a few seconds at the selected breakpoint.

## Debugging the Plug-In on Windows

To debug an Integration Service plug-in on Windows:

1. In Visual C++, click File > Open Solution to open a workspace.
2. Click Project > Properties.

3. Set the build type to debug.
4. Add the following statement near the beginning of the plug-in source code to specify a Sleep call of 10 seconds:

   ```
   Sleep (10000)
   ```
5. Choose Build or press F7 to build the plug-in.
6. Start a PowerCenter session and attach the debugger to the PmDTM process.

   Verify that you attach the debugger to the PmDTM process, not the pmserver process.
7. Set a breakpoint immediately after the Sleep call.

## Compiling the Shared Library on UNIX

If you are building a plug-in on UNIX, you may not be able to access the directory containing the Informatica client tools directly. As a first step, you need to copy all the files needed for the shared library to the UNIX machine where you plan to perform the build.

To build shared libraries on UNIX:

1. Set the environment variable PM_HOME to the PowerCenter installation directory.
2. Enter the command to create the plug-in.

   The following table lists the command to compile a plug-in:

| UNIX platform | Command |
|---|---|
| Solaris | make -f makefile.sun |
| AIX (32-bit) | make -f makefile.aix |
| AIX (64-bit) | make -f makefile.aix64 |

### Debugging the Plug-In on UNIX

To debug an Integration Service plug-in on UNIX:

1. Add the following statement near the beginning of the plug-in source code to specify a Sleep call of 10 seconds:

   ```
   sleep (10)
   ```
2. Build the plug-in in debug mode.
3. Start a PowerCenter session and attach the debugger to the PmDTM process.

   For more information about attaching a debugger to the PmDTM process, see the integrated development environment (IDE) documentation.
4. Set a breakpoint immediately after the sleep call.

# Unregistering a PowerExchange Plug-in

If you do not need the PowerExchange plug-in, you can unregister it to remove it from the PowerCenter repository. Use the Administration Console to remove a PowerExchange plug-in from the PowerCenter repository.

If the Repository Service is not running in exclusive mode, the Remove buttons for plug-ins are disabled. Verify that all users are disconnected from the repository before you unregister a plug-in.

**Note:** If you unregistering a plug-in, objects you define with the plug-in can become unusable.

To unregister a plug-in:

1. In the Navigator of the PowerCenter Administration Console, select the Repository Service from which you want to remove the plug-in.
2. Run the Repository Service in exclusive mode.
3. Click the Plug-ins tab.

   The list of registered plug-ins appears.
4. Click the Remove button for the plug-in you want to unregister.
5. Enter a repository user name and password.

   The user must be the Administrator.
6. Click OK.
7. Run the Repository Service in normal mode.

CHAPTER 4

# Plug-in Metadata

This chapter includes the following topics:

## Plug-in Metadata Overview

To register a plug-in with a PowerCenter repository, create an XML file that contains the repository ID attributes of the plug-in. The repository ID attributes define the properties of the plug-in and provides a unique identity for the plug-in.

The root element of the XML file is POWERMART, which includes the REPOSITORY element. In the REPOSITORY element, you use the PLUGIN element to define the properties of the plug-in.

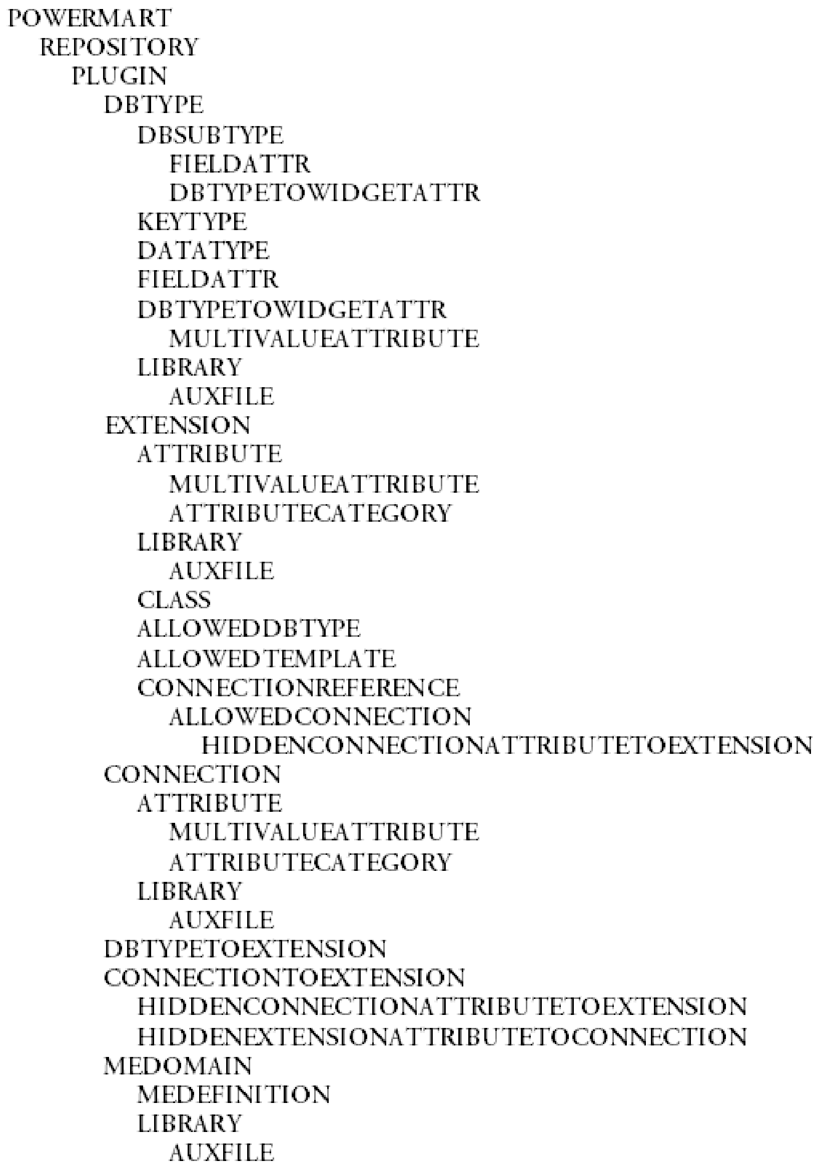
After you create the plug-in definition file, register the plug-in with a PowerCenter repository. You can use the Administration Console to register, update, or uninstall a plug-in from a repository.

You can also use the *pmrep* RegisterPlugin command to register or update the metadata definition with the PowerCenter repository. Use the *pmrep* UnregisterPlugin command to uninstall the plug-in from the PowerCenter repository.

## Structure of the Plug-in Definition File

Use any text editor or XML creation tool to create the plug-in definition file. The name of the XML file identifies the plug-in. The PowerCenter installs a DTD file named plugin.dtd. The plug-in definition file must conform to the rules of the plugin.dtd file. When you register the plug-in, PowerCenter validates the plug-in definition file against the plugin.dtd file.

The following element hierarchy shows the structure of the plugin.dtd:



```
POWERMART
    REPOSITORY
        PLUGIN
            DBTYPE
                DBSUBTYPE
                    FIELDATTR
                    DBTYPETOWIDGETATTR
                KEYTYPE
                DATATYPE
                FIELDATTR
                DBTYPETOWIDGETATTR
                    MULTIVALUEATTRIBUTE
                LIBRARY
                    AUXFILE
            EXTENSION
                ATTRIBUTE
                    MULTIVALUEATTRIBUTE
                    ATTRIBUTECATEGORY
                LIBRARY
                    AUXFILE
                CLASS
                ALLOWEDDBTYPE
                ALLOWEDTEMPLATE
                CONNECTIONREFERENCE
                    ALLOWEDCONNECTION
                        HIDDENCONNECTIONATTRIBUTETOEXTENSION
            CONNECTION
                ATTRIBUTE
                    MULTIVALUEATTRIBUTE
                    ATTRIBUTECATEGORY
                LIBRARY
                    AUXFILE
            DBTYPETOEXTENSION
            CONNECTIONTOEXTENSION
                HIDDENCONNECTIONATTRIBUTETOEXTENSION
                HIDDENEXTENSIONATTRIBUTETOCONNECTION
            MEDOMAIN
                MEDEFINITION
                LIBRARY
                    AUXFILE
```

When you create or modify the plug-in definition file, verify that it uses the structure of the plugin.dtd file. For example, the plugin.dtd file specifies that a session extension must either be a READER or a WRITER. The extension is invalid if you specify an extension type of BOTH.

# PLUGIN Element

In the XML file, you need to define a REPOSITORY element in the root element POWERMART. The DTD file requires these elements for validation.

The DTD file requires the root element POWERMART with the child element REPOSITORY. Add a PLUGIN element as a child of the REPOSITORY element. Use the PLUGIN element to define the metadata for the plug-in that you create. The attributes for the PLUGIN element uniquely identify the plug-in.

**Note:** The REPOSITORY element has a CODEPAGE attribute. Set this attribute to US-ASCII so that the plug-in will work with all Repository Services that use ASCII compatible code pages.

The following table describes the attributes of the PLUGIN element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| NAME | Required | Name of the plug-in. |
| ID | Required | Identifier for the plug-in. Use the ID attribute to distinguish plug-ins with identical VENDORID. For example, you develop multiple plug-ins for the same vendor. Use the same VENDORID but assign a unique ID for each plug-in. |
| VENDORNAME | Required | Name of the vendor. |
| VENDORID | Required | Identifier for the vendor obtained from Informatica. |
| DESCRIPTION | Optional | Description for the plug-in. |
| VERSION | Required | Version of the plug-in. Use this attribute to keep track of updates to the plug-in. |

After defining an identity for the plug-in, use the child elements of the PLUGIN element to define other properties of the plug-in. For example, the plug-in can extract data from TIBCO Rendezvous. Use the child elements of the PLUGIN element to identify the plug-in as a TIBCO reader that uses a specified TIBCO connection. The PLUGIN element has the following child elements:

- DBTYPE
- EXTENSION
- CONNECTION
- DBTYPETOEXTENSION
- CONNECTIONTOEXTENSION
- MEDOMAIN

# DBTYPE Element

Use the DBTYPE element to define the metadata for the plug-in. The attributes of the DBTYPE element uniquely identify the database type of the plug-in.

The following table describes the attributes of the DBTYPE element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| NAME | Required | Name of the third-party database that you want to define for the plug-in. |
| ID | Required | Identifier for the database type obtained from Informatica. This attribute identifies this DBTYPE. |
| BASEID | Required | Base ID for the datatypes that can be used with this DBTYPE. Use the lowest value from the datatype range obtained from Informatica. |
| DEFAULTDBSUBTYPE | Required | Identifier for the default subtype for this DBTYPE. For example, Siebel table and Siebel business component are subtypes of the Siebel DBTYPE. When you create a Siebel source, the Designer creates a Siebel table by default. If you do not want to specify a DBSUBTYPE, set this attribute to 0. |
| FIELDSEPARATOR | Optional | Character to use to separate field names from table names in this DBTYPE. For example, SAP uses a "-" (hyphen) to separate a field name from its table name. |
| INVALIDCHARS | Optional | Use this attribute to specify characters that cannot be used in table, field, transformation, or port names. For example, if the $ and & characters are invalid, set the value of this attribute to "$&". The PowerExchange API framework uses this attribute to perform validation. |
| INVALIDFIRSTCHARS | Optional | Use this attribute to specify characters that cannot be used as the first character in table, field, transformation, or port names. For example, if the @ and # characters are invalid as first characters, set the value of this attribute to "@#". The PowerExchange API framework uses this attribute to perform validation. |
| TYPE | Required | Type of PowerCenter object to associate with this DBTYPE. You can set this attribute to one of the following values:<br>- SOURCE<br>- TARGET<br>- BOTH |
| COMPONENTVERSION | Required | Version of this DBTYPE. Indicates that the attributes of the DBTYPE have changed since the previous version. Use this attribute to keep track of updates to the DBTYPE element.<br>Update this attribute only when the DBTYPE has changed. This attribute does not depend on the version of the plug-in. |
| DATETIMEFORMAT | Optional | Date and time format to use with this DBTYPE. |
| HASGROUPS | Optional | Indicates whether fields for this DBTYPE can be grouped.<br>Set to YES to enable groups for fields in an object with this DBTYPE. Set to NO to disable groups. |

| Attribute | Required/ Optional | Description |
|---|---|---|
| HASFIELDATTRS | Optional | Indicates whether fields of this DBTYPE can have attributes.<br>Set to YES to enable attributes for fields in an object with this DBTYPE. Set to NO to disable attributes.<br>If you set this attribute to NO, you cannot include a FIELDATTR child element for this DBTYPE. |
| HASKEYTYPE | Optional | Indicates whether this DBTYPE can have key types. Set to YES to enable key types for this DBTYPE and display columns for keys in the Designer. Set to NO to disable key types. If you set this attribute to NO, this DBTYPE cannot use any key. |
| HASNULLTYPE | Optional | Indicates whether this DBTYPE can have NULL fields. Set to YES to enable NULL assignment for fields in an object with this DBTYPE. Set to NO to disable NULL fields. |
| HASBUSINESSNAME | Optional | Indicates whether fields for this DBTYPE can have business names. Set to YES to enable business names for fields in an object with this DBTYPE. Set to NO to disable business names. |
| HASFLATFILE | Optional | Indicates whether to display flat file information for sources of this DBTYPE. Set to YES to display flat file information. Set to NO to disable flat file display. |
| EDITGROUPS | Optional | Indicates whether groups in this DBTYPE can be edited. Set to YES to enable editing of groups for fields in an object that uses this DBTYPE. Set to NO to disable editing of groups. |
| EDITFIELDATTRS | Optional | Indicates whether field attributes for this DBTYPE can be edited. Set to YES to enable editing of field attributes in an object that uses this DBTYPE. Set to NO to disable editing of field attributes. |
| EDITFIELDNAME | Optional | Indicates whether field names for this DBTYPE can be edited. Set to YES to enable editing of field names in an object that uses this DBTYPE. Set to NO to disable editing of field names |
| EDITDATATYPE | Optional | Indicates whether datatypes for this DBTYPE can be edited. Set to YES to enable editing of datatypes in an object that uses this DBTYPE. Set to NO to disable editing of datatypes. |
| EDITPRECISION | Optional | Indicates whether datatype precision for this DBTYPE can be edited. Set to YES to enable editing of datatype precision in an object that uses this DBTYPE. Set to NO to disable editing of datatype precision. |
| EDITSCALE | Optional | Indicates whether datatype scales for this DBTYPE can be edited. Set to YES to enable editing of datatype scales in an object that uses this DBTYPE. Set to NO to disable editing of datatype scales. |
| EDITKEYTYPE | Optional | Indicates whether key types for this DBTYPE can be edited.<br>Set to YES to enable editing of key types in an object that uses this DBTYPE. Set to NO to disable editing of key types. |
| EDITNULLTYPE | Optional | Indicates whether null fields for this DBTYPE can be edited. Set to YES to enable editing of NULL fields in an object that uses this DBTYPE. Set to NO to disable editing of NULL fields. |

| Attribute | Required/ Optional | Description |
|---|---|---|
| EDITBUSINESSNAME | Optional | Indicates whether business names for fields in this DBTYPE can be edited. Set to YES to enable editing of business names for fields in an object that uses this DBTYPE. Set to NO to disable editing of business names. |
| EDITFLATFILE | Optional | Indicates whether the information for flat files created from this DBTYPE can be edited. Set to YES to enable editing of flat file information. Set to NO to disable editing of flat file information. |
| ISRELATIONAL | Required | Indicates whether this DBTYPE is relational. Set to YES if the DBTYPE is relational. Set to NO to specify the DBTYPE as non-relational. |
| CANPREVIEWDATA | Optional | Set to YES to enable data preview for this DBTYPE. Set to NO to disable data preview. |
| FIXEDFIELDS | Optional | Set to YES to prevent editing or adding of fields to this DBTYPE. Set to NO to allow editing or adding fields. |
| CANCHANGEDBTYPETO | Optional | Set to YES to enable other DBTYPEs to change into this DBTYPE. Set to NO to disable other DBTYPEs from changing into this DBTYPE. |
| CANCHANGEDBTYPEFROM | Optional | Set to YES to enable this DBTYPE to change into other DBTYPEs. Set to NO to disable this DBTYPE from changing into other DBTYPEs. |
| CANCOPYFIELDSTO | Optional | Set to YES to enable copying fields to a source or target of this DBTYPE. Set to NO to disable copying fields into a source or target of this DBTYPE. |
| CANCOPYFIELDSFROM | Optional | Set to YES to enable copying fields from a source or target of this DBTYPE. Set to NO to disable copying fields from a source or target of this DBTYPE. |
| CANLINKFIELDSFROM | Optional | Set to YES to enable fields to link from an object of this DBTYPE. Set to NO to disable fields from linking from an object of this DBTYPE. |
| CANLINKFIELDSTO | Optional | Set to YES to enable fields to create primary key/foreign key links to an object of this DBTYPE. Set to NO to disable key fields from linking to an object of this DBTYPE. |
| CANBECREATED | Optional | Set to YES to enable the Designer to create sources and targets of this DBTYPE. Set to NO to disable the Designer from creating sources and targets of this DBTYPE. |
| CANADDNEWSOURCEFIELD | Optional | Set to YES to enable the addition of new source fields in the Source Analyzer. Set to NO to disable the addition of source fields in the Source Analyzer. |
| CANADDNEWTARGETFIELD | Optional | Set to YES to enable the addition of new target fields in the Target Designer. Set to NO to disable the addition of target fields in the Target Designer |

The DBTYPE element has the following child elements:

- DBSUBTYPE
- KEYTYPE
- DATATYPE
- FIELDATTR
- DBTYPETOWIDGETATTR
- LIBRARY

## DBSUBTYPE Element

Use the DBSUBTYPE element to define subtypes of the plug-in database. For example, you have a plug-in that can run on either Oracle or Microsoft SQL Server. Use the DBSUBTYPE element to define subtypes of each database.

If you define the DBSUBTYPE element differently from the DBTYPE element, the definition of the DBSUBTYPE element overrides the definition of the DBTYPE element. For example, the plug-in definition file defines a DBTYPE element that allows business names and a DBSUBTYPE element that disables the business names. When you create a source with the DBSUBTYPE, the object will not include business names.

The following table describes the attributes of the DBSUBTYPE element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| ID | Required | Identifier for the DBSUBTYPE. Use this attribute to distinguish between different categories of the same DBTYPE. Assign a unique ID for each DBSUBTYPE you define for a DBTYPE. |
| NAME | Required | Name of the DBSUBTYPE. |
| TYPE | Required | Type of PowerCenter object to associate with this DBSUBTYPE. You can set this attribute to one of the following values:<br>- SOURCE<br>- TARGET<br>- BOTH |
| HASGROUPS | Optional | Indicates whether fields for this DBSUBTYPE can be grouped. Set to YES to enable groups for fields in an object with this DBSUBTYPE. Set to NO to disable groups. |
| HASFIELDATTRS | Optional | Indicates whether fields of this DBSUBTYPE can have attributes. Set to YES to enable attributes for fields in an object with this DBSUBTYPE. Set to NO to disable attributes.<br>If you set this attribute to NO, you cannot include a FIELDATTR child element for this DBSUBTYPE. |
| HASKEYTYPE | Optional | Indicates whether this DBSUBTYPE can have key types. Set to YES to enable key types for this DBSUBTYPE and display columns for keys in the Designer. Set to NO to disable key types. If you set this attribute to NO, this DBSUBTYPE cannot use any key. |
| HASNULLTYPE | Optional | Indicates whether this DBSUBTYPE can have NULL fields. Set to YES to enable NULL assignment for fields in an object with this DBSUBTYPE. Set to NO to disable NULL fields. |

| Attribute | Required/ Optional | Description |
|---|---|---|
| HASBUSINESSNAME | Optional | Indicates whether fields for this DBSUBTYPE can have business names. Set to YES to enable business names for fields in an object with this DBSUBTYPE. Set to NO to disable business names. |
| HASFLATFILE | Optional | Indicates whether flat files can be created with this DBSUBTYPE. Set to YES to enable the creation of flat files with this DBSUBTYPE. Set NO to disable flat file creation. |
| EDITGROUPS | Optional | Indicates whether groups in this DBSUBTYPE can be edited. Set to YES to enable editing of groups for fields in an object that uses this DBSUBTYPE. Set to NO to disable editing of groups. |
| EDITFIELDATTRS | Optional | Indicates whether field attributes for this DBSUBTYPE can be edited. Set to YES to enable editing of field attributes in an object that uses this DBSUBTYPE. Set to NO to disable editing of field attributes. |
| EDITFIELDNAME | Optional | Indicates whether field names for this DBSUBTYPE can be edited. Set to YES to enable editing of field names in an object that uses this DBSUBTYPE. Set to NO to disable editing of field names |
| EDITDATATYPE | Optional | Indicates whether datatypes for this DBSUBTYPE can be edited. Set to YES to enable editing of datatypes in an object that uses this DBSUBTYPE. Set to NO to disable editing of datatypes. |
| EDITPRECISION | Optional | Indicates whether datatype precision for this DBSUBTYPE can be edited. Set to YES to enable editing of datatype precision in an object that uses this DBSUBTYPE. Set to NO to disable editing of datatype precision. |
| EDITSCALE | Optional | Indicates whether datatype scales for this DBSUBTYPE can be edited. Set to YES to enable editing of datatype scales in an object that uses this DBSUBTYPE. Set to NO to disable editing of datatype scales. |
| EDITKEYTYPE | Optional | Indicates whether key types for this DBSUBTYPE can be edited. Set to YES to enable editing of key types in an object that uses this DBSUBTYPE. Set to NO to disable editing of key types. |
| EDITNULLTYPE | Optional | Indicates whether null fields for this DBSUBTYPE can be edited. Set to YES to enable editing of NULL fields in an object that uses this DBSUBTYPE. Set to NO to disable editing of NULL fields. |
| EDITBUSINESSNAME | Optional | Indicates whether business names for fields in this DBSUBTYPE can be edited. Set to YES to enable editing of business names for fields in an object that uses this DBSUBTYPE. Set to NO to disable editing of business names. |
| EDITFLATFILE | Optional | Indicates whether the information for flat files created from this DBSUBTYPE can be edited. Set to YES to enable editing of flat file information. Set to NO to disable editing of flat file information. |
| ISRELATIONAL | Required | Indicates whether this DBSUBTYPE is relational. Set to YES if the DBSUBTYPE is relational. Set to NO to specify the DBSUBTYPE as non-relational. |

| Attribute | Required/ Optional | Description |
|---|---|---|
| CANPREVIEWDATA | Optional | Set to YES to enable data preview for this DBSUBTYPE. Set to NO to disable data preview. |
| FIXEDFIELDS | Optional | Set to YES to prevent editing or adding of fields to this DBSUBTYPE. Set to NO to allow editing or adding fields. |
| CANCHANGEDBTYPETO | Optional | Set to YES to enable other DBSUBTYPEs to change into this DBSUBTYPE. Set to NO to disable other DBSUBTYPEs from changing into this DBSUBTYPE. |
| CANCHANGEDBTYPEFROM | Optional | Set to YES to enable this DBSUBTYPE to change into other DBSUBTYPEs. Set to NO to disable this DBSUBTYPE from changing into other DBSUBTYPEs. |
| CANCOPYFIELDSTO | Optional | Set to YES to enable copying fields to a source or target of this DBSUBTYPE. Set to NO to disable copying fields into a source or target of this DBSUBTYPE. |
| CANCOPYFIELDSFROM | Optional | Set to YES to enable copying fields from a source or target of this DBSUBTYPE. Set to NO to disable copying fields from a source or target of this DBSUBTYPE. |
| CANLINKFIELDSFROM | Optional | Set to YES to enable fields to link from an object of this DBSUBTYPE. Set to NO to disable fields from linking from an object of this DBSUBTYPE. |
| CANLINKFIELDSTO | Optional | Set to YES to enable fields to create primary key/foreign key links to an object of this DBSUBTYPE. Set to NO to disable key fields from linking to an object of this DBSUBTYPE. |
| CANBECREATED | Optional | Set to YES to enable the Designer to create sources and targets of this DBSUBTYPE. Set to NO to disable the Designer from creating sources and targets of this DBSUBTYPE. |
| CANADDNEWSOURCEFIELD | Optional | Set to YES to enable the addition of new source fields in the Source Analyzer. Set to NO to disable the addition of source fields in the Source Analyzer. |
| CANADDNEWTARGETFIELD | Optional | Set to YES to enable the addition of new target fields in the Target Designer. Set to NO to disable the addition of target fields in the Target Designer |

The DBSUBTYPE element has the following child elements:

- FIELDATTR
- DBTYPETOWIDGETATTR

## KEYTYPE Element

Use the KEYTYPE element to define a key for the DBTYPE. The key can be a primary key, foreign key, or a new type of key that you define.

To use the KEYTYPE element, set the HASKEYTYPE attribute of the DBTYPE element to YES. If you define a KEYTYPE element and the HASKEYTYPE attribute of the DBTYPE element is set to NO, the plug-in registration will fail.

The following table describes the attributes of the KEYTYPE element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| NAME | Required | Name of the key type. |
| TABLETYPE | Required | Set to SOURCE if this type of key will be used for sources. Set to TARGET if this type of key will be used for targets. To use a key type for sources and targets, define a key type for sources and another for targets. |
| KEYTYPE | Required | Set to PRIMARY to create a primary key type. Set to FOREIGN to create a foreign key type. Set to CUSTOM to create a custom key type. |
| KEYTYPEBIT | Optional | Decimal value of the key type bits for the key type.<br>The first eight bits are reserved by Informatica. You can change the first two bits to indicate a primary or foreign key. Set the first bit to 1 to indicate that the key is a primary key. Set the second bit to 1 to indicate that the key is a foreign key. You can set any bit except the first 8 bits to indicate a custom key. For example, to create a user-defined key in PeopleSoft that is also a primary key, set the first bit and the ninth bit to 1. The resulting decimal value is 257.<br>Set this attribute only for custom key types. |

## DATATYPE Element

Use the DATATYPE element to define datatypes for the DBTYPE.

For example, you want to define a datatype named CBigInt for the DBTYPE. The following sample code shows the DATATYPE element with the attributes that define the CBigInt datatype:

```
<DATATYPE NAME="CBigInt" ID="300201" ODBCTYPE="SQL_BIGINT" READONLYPRECISION="10"
READONLYSCALE="0" HASSCALE="YES" CANEDITSCALE="NO" CANEDITPRECISION="NO"
INTERNALCONVERTABLE="NO"/>
```

The following table describes the attributes of the DATATYPE element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| NAME | Required | Name of the datatype. |
| ID | Required | Identifier for the datatype. The ID must be within the range of DATATYPE IDs provided by Informatica. |
| ODBCTYPE | Required | ODBC type of this datatype. Define a separate DATATYPE element for each ODBC type. |
| READONLYPRECISION | Optional | Indicates the default precision for this datatype. If the CANEDITPRECISION attribute for this DATATYPE is set to YES, set this attribute to “0”. |
| READONLYSCALE | Optional | Indicates the default scale for this datatype. If the CANEDITSCALE attribute for this DATATYPE is set to YES, set this attribute to “0”. |

| Attribute | Required/ Optional | Description |
|---|---|---|
| HASSCALE | Optional | Set to YES so this datatype can have a scale. Set to NO to disable the scale. |
| CANEDITSCALE | Optional | Set to YES to allow editing of the datatype scale. Set to NO to disable editing of the scale. |
| CANEDITPRECISION | Optional | Set to YES to allow editing of the datatype precision. Set to NO to disable editing of the precision. |
| INTERNALCONVERTABLE | Optional | Set to YES to internally convert the datatype to another datatype. The datatype converts to a different datatype that has the same ID and the INTERNALCONVERTABLE attribute set to NO. Set this attribute to NO to disable internal conversion of the datatype.<br>If you set this attribute to YES, define another datatype with the same ID and the INTERNALCONVERTABLE attribute set to NO. |

You must define at least one DATATYPE element for each of the following ODBC types:

- SQL_BIGINT
- SQL_BINARY
- SQL_BIT
- SQL_CHAR
- SQL_DATE
- SQL_DECIMAL
- SQL_DOUBLE
- SQL_FLOAT
- SQL_IDENTITY
- SQL_INTEGER
- SQL_LONGVARBINARY
- SQL_LONGVARCHAR
- SQL_MONEY
- SQL_NUMERIC
- SQL_REAL
- SQL_SMALLINT
- SQL_TIME
- SQL_TIMESTAMP
- SQL_TINYINT
- SQL_WCHAR
- SQL_WVARCHAR
- SQL_WLONGVARCHAR
- SQL_VARBINARY
- SQL_VARCHAR

# FIELDATTR Element

Use the FIELDATTR element to define fields for a DBTYPE or DBSUBTYPE.

The following example defines a field named Physical Table Name for the DBTYPE:

```
<FIELDATTR NAME="Physical Table Name" ID="300200" DESCRIPTION="Physical Table Name"
TYPE="BOTH" ISINT="NO" ISHIDDEN="NO"/>
```

The following table describes the attributes of the FIELDATTR element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| NAME | Required | Name for the field attribute. |
| ID | Required | Identifier for the field attribute. |
| DESCRIPTION | Optional | Description of the field attribute. |
| TYPE | Required | Type of PowerCenter object to associate with this field attribute. You can set this attribute to one of the following values:<br>- SOURCE<br>- TARGET<br>- BOTH |
| ISINT | Optional | Set to YES if the field attribute is an integer. Set to NO if the field attribute is not an integer. |
| ISHIDDEN | Optional | Set to YES if the field attribute is hidden. Set to NO if the field attribute is not hidden. |
| ISDEPRECATED | Optional | Set to YES if the field attribute is deprecated. |

# DBTYPETOWIDGETATTR Element

By default, the source or target for a plug-in has pre-defined properties. The values for these properties are also pre-defined. You can assign default values for these properties. Use the element DBTYPETOWIDGETATTR element to set the default values for the properties.

Define a DBTYPETOWIDGETATTR element for each property that you want to define default values.

The following table describes the pre-defined properties and their possible values:

| Property | Associated Object | Description | Pre-defined Values |
|---|---|---|---|
| Load Scope | Target | Determines when the writer plug-in loads processed rows into the external application. | This property can have one of the following values:<br>- Row. Loads a row after it is processed.<br>- Transaction. Loads all rows processed in a transaction on commit.<br>- All Input. Loads all rows at end of file.<br>Default is All Input. |
| Partial Load Recovery | Target | Specifies how the target handles a previous partial load during recovery. | This property can have one of the following values:<br>- None<br>- Append<br>- Truncate<br>Default is None. |

The following table describes the attributes of the DBTYPETOWIDGETATTR element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| OBJECTTYPE | Required | Type of PowerCenter object to associate with this DBTYPETOWIDGETATTR. You can set this attribute to one of the following values:<br>- SOURCE<br>- TARGET |
| WIDGETATTRIBUTENAME | Required | Name of the pre-defined property. |
| ISREADONLY | Optional | Set to YES to make the property read-only. Set to NO if the user can edit the value. |
| ISDISABLED | Optional | Set to YES to disable the property. Set to NO enable the property. |
| ISHIDDEN | Optional | Set to YES to hide the property in the Designer. Set to NO to display the property in the Designer. |
| ISEXPORTED | Optional | Set to YES if this attribute can be exported. Set to NO if this attribute cannot exported. |
| ALLOWALLVALUES | Optional | Set to YES to display all values that can be selected for the property. Set to NO to specify a subset of all values. Define a MULTIVALUEATTRIBUTE Element for each value to display. |

The DBTYPETOWIDGETATTR element has the following child element:

- MULTIVALUEATTRIBUTE

## MULTIVALUEATTRIBUTE Element

The MULTIVALUEATTRIBUTE element defines a value for an attribute with multiple possible values.

You can define a MULTIVALUEATTRIBUTE element as a child of a DBTYPETOWIDGETATTR element when you set the ALLOWALLVALUES attribute of the DBTYPETOWIDGETATTR element to NO. Use the MULTIVALUEATTRIBUTE element to specify the values of the pre-defined property to display.

You can also define a MULTIVALUEATTRIBUTE element as a child of an ATTRIBUTE element when you set the TYPE attribute of an extension or connection ATTRIBUTE element to MULTIVALUED.

The following table describes the attributes of the MULTIVALUEATTRIBUTE element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| NAME | Required | Enter one of the values of an attribute or property with multiple possible values. |

# LIBRARY Element

Use the LIBRARY element to specify a library or shared object to associate with the following objects:

- DBTYPE
- EXTENSION
- CONNECTION
- MEDOMAIN

The following example shows the definition of the LIBRARY element for an HTML_WRITER session extension:

```
<LIBRARY NAME = "wrtplugindll.dll" OSTYPE = "NT" />
```

The following table describes the attributes of the LIBRARY element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| NAME | Required | Name of the library to associate with a DBTYPE, EXTENSION, CONNECTION, or MEDOMAIN. |
| OSTYPE | Required | Operating system used to develop the library. Set to one of the following operating systems:<br>- NT<br>- SOLARIS<br>- AIX<br>- DEC<br>- Linux<br>- OS390. |
| TYPE | Optional | Type of library. Set to one of the following types:<br>- SERVER<br>- CLIENT<br>- VALIDATION<br>- REPAGENT<br>- UPGRADE<br>Default is SERVER. |

The LIBRARY element has the following child element:

- AUXFILE

<< Need description of AUXFILE>>

# EXTENSION Element

Use the EXTENSION element to specify the properties of the session extension. You can define your session extension as a reader or writer. For example, you can use the following XML code to create an HTML_WRITER extension:

```
<EXTENSION NAME= "HTML_WRITER" EXTENSIONTYPE= "WRITER"

EXTENSIONSUBTYPE = "500001" DESCRIPTION= "HTML WRITER"

COMPONENTVERSION = "1.0.0">
```

The following table describes the attributes of the EXTENSION element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| NAME | Required | Name of the session extension. |
| EXTENSIONTYPE | Required | Type of extension. Set to READER for a reader extension. Set to WRITER for a writer extension. |
| EXTENSIONSUBTYPE | Required | Extension subtype. You can obtain the value for this attribute from Informatica. |
| DESCRIPTION | Optional | Description of the session extension. |
| HASFILEINFO | Optional | Set to YES if the session extension requires a file description. Set to NO if the session extension does not require a file description.<br>**Note:** If you set the DISPLAYFILEINFO attribute to YES, set the HASFILEINFO attribute to YES. |
| SUPPORTPARTITIONS | Optional | Set this attribute to one of the following values:<br>- Across Grid<br>- Locally<br>- None<br>If this extension supports partitions and can run on a PowerCenter grid, set the value to Across Grid. If the extension supports partitions but cannot run on a grid, set the value to Locally. |
| DISPLAYFILEINFO | Optional | Set to YES to enable the display of file information for the session extension. Set to NO to disable the display of file information.<br>**Note:** If you set the DISPLAYFILEINFO attribute to YES, set the HASFILEINFO attribute to YES. |
| COMPONENTVERSION | Required | Version of the EXTENSION. Indicates that the attributes of the EXTENSION have changed since the previous version.Use this attribute to keep track of updates to the EXTENSION element. |
| LANG | Optional | Language in which the plug-in is developed. Set to one of the following values:<br>- CPP<br>- JAVA<br>Default value is CPP. |

The EXTENSION element has the following child elements:

- ATTRIBUTE
- LIBRARY
- CLASS
- ALLOWEDDBTYPE
- ALLOWEDTEMPLATE
- CONNECTIONREFERENCE

# ATTRIBUTE Element

Use the ATTRIBUTE element to define an attribute of the extension or connection that you want to create. For example, to define a Stylesheet Name attribute for the HTML_WRITER extension, you can use the following code:

```
<ATTRIBUTE NAME = "Stylesheet Name" ID = "1" TYPE = "PROPERTY" DATATYPE = "STRING"
REFERENCELEVEL = "TARGET" ISREQUIRED = "YES" ISSESSIONOVERRIDABLE = "YES"
ISINSTANCEOVERRIDABLE = "YES" ISPARTITIONOVERRIDABLE = "YES" ISSESSIONVARSALLOWED =
"YES" ISSERVERVARSALLOWED = "YES" ISVARPREFIXALLOWED = "YES" ISVARFULLNAMEALLOWED =
"YES" VARIABLEPREFIX = "varpfx"/>
```

The following table describes the attributes of the ATTRIBUTE element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| NAME | Required | Name of the attribute. |
| ID | Required | Identifier for the attribute. |
| TYPE | Required | Type for the attribute. Set to one of the following values:<br>- SQL<br>- PROPERTY<br>- BOOLEAN<br>- MULTIVALUED<br>- PASSWORD<br>- FILENAME<br>The value for the TYPE attribute determines the value for the DATATYPE attribute. For a list of the DATATYPE attribute values that correspond to the TYPE attribute values, see “ATTRIBUTE Element” on page 45. |
| DATATYPE | Required | Set to NUMBER or STRING based on the value of the TYPE attribute. For more information, see “ATTRIBUTE Element” on page 45. |
| REFERENCELEVEL | Required | Transformation level that the attribute applies to. When you define a reader extension or a reader connection, you can set this attribute to SOURCE or DSQ. When you define a writer extension or a writer connection, set this attribute to TARGET. |
| ISINSTANCEOVERRIDABLE | Optional | Set to YES to enable overriding reusable session instances. Set to NO to disable instance overrides. |
| ISSESSIONOVERRIDABLE | Optional | Set to YES to enable session overrides. Set to NO to disable session overrides. |
| ISPARTITIONOVERRIDABLE | Optional | Set to YES to enable partition overrides. Set to NO to disable partition overrides. |
| ISSESSIONVARSALLOWED | Optional | Set to YES to allow session variables in the attribute. Set to NO to disable session variables. |
| ISSERVERVARSALLOWED | Optional | Set to YES to allow server variables in the attribute. Set to NO to disable server variables. |

| Attribute | Required/ Optional | Description |
|---|---|---|
| ISREQUIRED | Optional | Set to YES if the attribute is required for the extension or connection. Set to NO to make the attribute optional for the extension or connection. |
| ISVARPREFIXALLOWED | Optional | Set to YES to enable variable prefixes for the attribute. Set to NO to disable variable prefixes for the attribute. |
| ISVARFULLNAMEALLOWED | Optional | Set to YES to enable variable full names for the attribute. Set to NO to disable variable full names for the attribute. |
| VARIABLEPREFIX | Optional | Variable prefix for the extension or connection attribute. |
| MINVALUE | Optional | Minimum value for the extension or connection attribute. |
| MAXVALUE | Optional | Maximum value for the extension or connection attribute. |
| GROUPID | Optional | Identifier for the group to which the extension or connection attribute belongs. You can assign a number from 1 to16 as group ID. |
| GROUPPOLICY | Optional | Defines the number of attributes that can be in one group. Set to one of the following values:<br>- NONE<br>- EXACTLYONE<br>- ATMOSTONE. |
| ISDEPRECATED | Optional | Set to YES if this attribute is deprecated.<br>Default is NO. |

The following table shows the possible values for the TYPE attribute and the corresponding DATATYPE values:

**Table 1. Values for TYPE and DATATYPE Attributes of the ATTRIBUTE Element**

| TYPE | DATATYPE |
|---|---|
| BOOLEAN | NUMBER |
| SQL | STRING |
| PASSWORD | STRING |
| FILENAME | STRING |
| PROPERTY | NUMBER or STRING |

Use the following guidelines when you set the attributes of the ATTRIBUTE element:

- If you set the ISSESSIONVARALLOWED attribute or the ISSERVERVARALLOWED attribute to YES, you must enter YES for either the ISVARPREFIXALLOWED attribute or the ISVARFULLNAMEALLOWED attribute. You cannot set both the ISVARPREFIXALLOWED and ISVARFULLNAMEALLOWED attributes to YES at the same time.
- You must set the VARIABLEPREFIX attribute to YES when you set the ISVARPREFIXALLOWED attribute or the ISVARFULLNAMEALLOWED attribute to YES.
- If you define the GROUPPOLICY attribute, you must set the GROUPID attribute. However, you can define the GROUPID attribute without setting the GROUPPOLICY attribute.

The ATTRIBUTE element has the following child elements:

- MULTIVALUEATTRIBUTE
- ATTRIBUTECATEGORY

### ATTRIBUTECATEGORY Element

In the Workflow Manager, attributes are divided into two groups: Memory Properties or Files, Directories and Commands. Use the ATTRIBUTECATEGORY element to indicate to which group the ATTRIBUTE element belongs. To indicate that the ATTRIBUTE element belongs to the Memory Properties group, set the value to MEMORY. To indicate that the ATTRIBUTE element belongs to the Files, Directories, and Commands group, set the value to FILESANDDIRECTORIES.

## LIBRARY Element

For more information about the LIBRARY Element, see "LIBRARY Element" on page 42.

## CLASS Element

Use the CLASS element to specify the class for the session extension when the library is developed in JAVA.

For example:

```
<CLASS NAME ="com/informatica/powerconnect/jms/server/reader/JMSReaderPlugin" />
```

The following table describes the attributes of the CLASS element:

| Attribute | Required/Optional | Description |
|---|---|---|
| NAME | Required | Name of the Java library class. |

## ALLOWEDDBTYPE Element

Use the ALLOWEDDBTYPE element to define the valid DBTYPEs for the session extension. Include an ALLOWEDDBTYPE element in the EXTENSION element for each DBTYPE you want to use with the extension. For example, to build a reader extension for a TIBCO Rendezvous source, define a reader extension and use the ALLOWEDDBTYPE element to associate the TIBCO DBTYPE with the TIBCO reader. You can also use the ALLOWEDDBTYPE element to make the TIBCO DBTYPE the default database for the TIBCO reader.

The following table describes the attributes of the ALLOWEDDBTYPE element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| DBTYPE | Required | ID of the DBTYPE you want to use with the session extension. Informatica provides a list of predefined DBTYPE IDs for known databases. For a list of the DBTYPE IDs for databases, see "ALLOWEDDBTYPE Element" on page 47. |
| ISDEFAULT | Optional | Set to YES if the DBTYPE is the default type for the session extension. Set to NO if the database type is not the default type. |

The following table shows the DBTYPE IDs for databases:

| Database | DBTYPE ID |
|---|---|
| Sybase | 2 |
| Oracle | 3 |
| Informix | 4 |
| Microsoft SQL Server | 5 |
| IBM DB2 | 6 |
| Flatfile | 7 |
| ODBC | 8 |
| XML | 12 |
| Teradata | 15 |

## ALLOWEDTEMPLATE Element

You can define this element when you define an extension for a Custom transformation. If you define an ALLOWEDTEMPLATE element for the EXTENSION element, do not define an ALLOWEDDBTYPE element.

## CONNECTIONREFERENCE Element

The CONNECTIONREFERENCE element defines the association between a connection and a session extension.

You can use the CONNECTIONREFERENCE element to create a group of connections to associate with a session extension. You can group connections used by the session extension for a particular task. For example, you want to extract data from an SAP R/3 system into PowerCenter. An application connection extracts data from the SAP R/3 system and creates a flat file in the SAP R/3 system. An FTP connection transfers this flat file to PowerCenter. Use the CONNECTIONREFERENCE element to define a connection group for the reader extension that includes the application and FTP connections.

The following table describes the attributes of the CONNECTIONREFERENCE element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| NAME | Required | Name of the connection. |
| CONNECTIONNUMBER | Required | Priority of the connection.<br>For example, you define three CONNECTIONREFERNCE elements to associate three connections with a session extension. Set the CONNECTIONNUMBER to “1”, “2”, or “3”. The group with the CONNECTIONNUMBER value of 1 has the highest connection priority and is the default connection. If you set the CONNECTIONNUMBER attribute of a CONNECTIONREFERENCE element to 1, you must also set the ISDEFAULT attribute of the ALLOWEDCONNECTION child element to YES. |
| ISSESSIONOVERRIDABLE | Optional | Set to YES to enable session overrides. Set to NO to disable session overrides. |
| ISPARTITIONOVERRIDABLE | Optional | Set to YES to enable partition overrides. Set to NO to disable partition overrides. |
| REFERENCELEVEL | Required | Set the object level to which the CONNECTIONREFERENCE applies. For a reader extension, set this attribute to SOURCE or DSQ. For a writer extension, set this attribute to TARGET. |
| ISREQUIRED | Optional | Set to YES to make CONNECTIONREFERENCE a requirement for the extension. Set to NO to make it optional. |
| ISSESSIONVARSALLOWED | Optional | Set to YES to allow session variables for the extension. Set to NO to disable session variables. |
| ISVARPREFIXALLOWED | Optional | Set to YES to enable variable prefixes for the extension. Set to NO to disable variable prefixes for the extension. |
| ISVARFULLNAMEALLOWED | Optional | Set to YES to enable variable full names for the extension. Set to NO to disable variable prefixes for the extension. |
| VARIABLEPREFIX | Optional | Variable prefix for the extension. |
| ISDATACONNECTION | Optional | Set to YES to indicate that the extension is a connection to a data source or target. |

The CONNECTIONREFERENCE has the following child element:

- ALLOWEDCONNECTION

## ALLOWEDCONNECTION Element

Use the ALLOWEDCONNECTION element to define the connection subtypes that can be used for the CONNECTIONREFERENCE element. If a CONNECTIONREFERENCE element requires multiple connections, you can use the ALLOWEDCONNECTION element to define the connections to group together within a CONNECTIONREFERENCE element. The connections you include in a group must be defined in the CONNECTION element.

The following table describes the attributes of the ALLOWEDCONNECTION element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| CONNECTIONTYPE | Required | Type of connection. Set to one of the following values:<br>- RELATIONAL<br>- APPLICATION<br>- FTP<br>- EXTERNALLOADER<br>- QUEUE. |
| CONNECTIONSUBTYPE | Required | Identifier for the connection subtype. For a list of predefined connection subtype IDs, see "ALLOWEDCONNECTION Element" on page 49. |
| ISDEFAULT | Optional | Set to YES if the connection is the default connection for the session extension. Set to NO if the connection is not the default for the session extension.<br>If CONNECTIONNUMBER attribute for the parent CONNECTIONREFERENCE element is set to 1, set this attribute to YES. |
| SUPPORTPARTITIONS | Optional | Set to YES if partitions can use the connection. Set to NO if partitions cannot use the connection.<br>If you set this attribute to NO, do not set the ISFORALLPARTITIONS attribute to YES. |
| ISFORALLPARTITIONS | Optional | Set to YES if all partitions can use the connection. Set to NO if not all partitions can use the connection. |

The following table shows the predefined IDs for the CONNECTIONSUBTYPE attribute:

**Table 2. Predefined IDs for the CONNECTIONSUBTYPE Attribute**

| Connection Subtype | Type of Connection | Connection Subtype ID |
|---|---|---|
| Oracle | Relational/External Loader | 101 |
| Sybase | Relational/External Loader | 102 |
| Informix | Relational | 103 |
| Microsoft SQL Server | Relational | 104 |
| DB2 | Relational | 105 |
| ODBC | Relational | 106 |
| Teradata | Relational | 107 |
| SAP BW | Application | 100001 |
| SAP R/3 | Application | 100101 |
| PeopleSoft on Oracle | Application | 100201 |

| Connection Subtype | Type of Connection | Connection Subtype ID |
|---|---|---|
| PeopleSoft on Sybase | Application | 100202 |
| PeopleSoft on Informix | Application | 100203 |
| PeopleSoft on Microsoft SQL Server | Application | 100204 |
| PeopleSoft on DB2 | Application | 100205 |
| Siebel on Oracle | Application | 100301 |
| Siebel on Sybase | Application | 100302 |
| Siebel on Informix | Application | 100303 |
| Siebel on Microsoft SQL Server | Application | 100304 |
| Siebel on DB2 | Application | 100305 |
| Teradata | External Loader | 103 |
| Teradata_TPUMP | External Loader | 104 |
| DB2EE_LOAD | External Loader | 105 |
| DB2EEE_AUTOLOAD | External Loader | 106 |
| Teradata_FASTLOAD | External Loader | 107 |
| Teradata_WB | External Loader | 108 |
| MQ Series | Queue | 101 |

The ALLOWEDCONNECTION element has the following child element:

- HIDDENCONNECTIONATTRIBUTETOEXTENSION

## HIDDENCONNECTIONATTRIBUTETOEXTENSION Element

Use the HIDDENCONNECTIONATTRIBUTETOEXTENSION element to hide a connection attribute from an extension.

You can define a HIDDENCONNECTIONATTRIBUTETOEXTENSION element as a child of an ALLOWEDCONNECTION element to hide an attribute of the connection associated with a session extension.

You can also define a HIDDENCONNECTIONATTRIBUTETOEXTENSION element as a child of a CONNECTIONTOEXTENSION element to hide an attribute of a connection associated with a session extension of another plug-in. The CONNECTIONTOEXTENSION element defines the connection that contains the attribute to hide and the session extension from which to hide the attribute that contains the attribute with the attribute to hide.

The following table describes the attributes of the HIDDENCONNECTIONATTRIBUTETOEXTENSION element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| CNXATTRIBUTEID | Required | Attribute ID of the connection attribute to hide from a session extension. |

# CONNECTION Element

Use the CONNECTION element to define a connection for a plug-in. After you register a plug-in with a defined connection, the connection information appears in the Connection Object Browser in the Workflow Manager.

The following table describes the attributes of the CONNECTION element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| NAME | Required | Name of the connection. |
| CONNECTIONTYPE | Required | Type of connection. Set to one of the following values:<br>- RELATIONAL<br>- APPLICATION<br>- FTP<br>- EXTERNALLOADER<br>- QUEUE. |
| CONNECTIONSUBTYPE | Required | Identifier for the connection subtype. For a list of predefined connection subtype IDs, see "ALLOWEDCONNECTION Element" on page 49. |
| RUNTIMECHECK | Optional | License key for validating the plug-in.<br>This attribute is currently not used. |
| HASUSERNAME | Optional | Set to YES if the connection requires a username. Set to NO if the connection does not require a username. |
| HASUSERPASSWORD | Optional | Set to YES if the connection requires a password. Set to NO if the connection does not require a password. |
| HASCONNECTSTRING | Optional | Set to YES if the connection requires a connect string. Set to NO if the connection does not require a connect string. |
| HASCODEPAGE | Optional | Set to YES if the connection has a code page. Set to NO if the connection does not have a code page. |
| HASPERMISSIONS | Optional | Set to YES to display the permission properties in the Designer. Set to NO to disable the permission properties. |

| Attribute | Required/ Optional | Description |
|---|---|---|
| ISRELATIONAL | Optional | Indicates whether this is an SQL connection to a relational database. Set to YES if the connection is relational. Set to NO if the connection is non-relational. |
| COMPONENTVERSION | Required | Version of the CONNECTION. Indicates that the attributes of the CONNECTION have changed since the previous version. Use this attribute to keep track of updates to the V element. |

The CONNECTION element has the following child elements:

- ATTRIBUTE
- LIBRARY

# DBTYPETOEXTENSION Element

When you register a new plug-in in the PowerCenter repository, you can define a relationship between the DBTYPE and the session extension of another plug-in registered in the repository. For example, a registered plug-in has a TIBCO_READER extension. When you register a new plug-in that contains a TIBCO_ORDERS DBTYPE, you can associate the TIBCO_ORDERS DBTYPE with the TIBCO_READER extension. The following example shows a DBTYPETOEXTENSION element defining the relationship between a DBTYPE and an extension:

```
<DBTYPETOEXTENSION EXTENSIONTYPE="READER" EXTENSIONSUBTYPE="300000" DBTYPE="300000"
ISDEFAULT="YES" />
```

You can also update the metadata of a registered plug-in to associate the session extension with a DBTYPE. Update the ALLOWEDDBTYPE element of an extension to associate it with a DBTYPE.

The following table describes the attributes of the DBTYPETOEXTENSION element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| EXTENSIONTYPE | Required | Set to READER if the referenced session extension is a reader. Set to WRITER if the referenced session extension is a writer. |
| EXTENSIONSUBTYPE | Required | Set the session extension subtype for the extension. |
| DBTYPE | Required | Set to the ID of the referenced DBTYPE. For a list of predefined DBTYPE IDs, see "ALLOWEDDBTYPE Element" on page 47. |
| ISDEFAULT | Optional | Set to YES if the DBTYPE is the default for the session extension. Set to NO if the DBTYPE is not the default for the session extension. |

# CONNECTIONTOEXTENSION Element

When you register a new plug-in in the PowerCenter repository, you can define a relationship between the connection and the session extension of another plug-in registered in the repository. For example, a registered plug-in has a TIBCO_WRITER extension. When you register a new plug-in, you can associate the connection defined for the plug-in with the TIBCO_WRITER extension. The following example shows a CONNECTIONTOEXTENSION element defining the relationship between a connection and an extension:

```
<CONNECTIONTOEXTENSION EXTENSIONTYPE= "WRITER" EXTENSIONSUBTYPE= "300200" CONNECTIONTYPE
"APPLICATION" CONNECTIONSUBTYPE= "300200" CONNECTIONNUMBER= "1" ISDEFAULT= "YES" />
```

You can also update the metadata of a registered plug-in to associate the a session extension with a connection. Update the ALLOWEDCONNECTION element of an extension to associate it with a connection.

The following table describes the attributes of the CONNECTIONTOEXTENSION element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| EXTENSIONTYPE | Required | Set to READER if the referenced session extension is a reader. Set to WRITER if the referenced session extension is a writer. |
| EXTENSIONSUBTYPE | Required | Set the session extension subtype for the extension. |
| CONNECTIONTYPE | Required | Connection type of the extension. Set to one of the following connection types:<br>- RELATIONAL<br>- APPLICATION<br>- FTP<br>- EXTERNALLOADER<br>- QUEUE. |
| CONNECTIONSUBTYPE | Required | Identifier for the connection subtype. For a list of predefined connection subtype IDs, see "ALLOWEDCONNECTION Element" on page 49. |
| CONNECTIONNUMBER | Required | Priority of the connection.<br>For example, you define three CONNECTIONTOEXTENSION elements to associate three connections with a session extension. Set the CONNECTIONNUMBER to "1", "2", or "3". The connection with the CONNECTIONNUMBER value of 1 has the highest connection priority and is the default connection. If you set the CONNECTIONNUMBER attribute of the CONNECTIONTOEXTENSION element to 1, you must also set the ISDEFAULT attribute to YES. |
| ISDEFAULT | Optional | Set to YES if the connection is the default connection for the session extension. Set to NO if the connection is not the default for the session extension.<br>If CONNECTIONNUMBER attribute is set to 1, set this attribute to YES. |

| Attribute | Required/ Optional | Description |
|---|---|---|
| SUPPORTPARTITIONS | Optional | Set to YES if partitions can use the connection. Set to NO if partitions cannot use the connection.<br>If you set this attribute to NO, do not set the ISFORALLPARTITIONS attribute to YES. |
| ISFORALLPARTITIONS | Optional | Set to YES if all partitions can use the connection. Set to NO if not all partitions can use the connection. |

The CONNECTIONTOEXTENSION element has the following child elements:

- HIDDENCONNECTIONATTRIBUTETOEXTENSION
- HIDDENEXTENSIONATTRIBUTETOCONNECTION

## HIDDENEXTENSIONATTRIBUTETOCONNECTION Element

Use the HIDDENEXTENSIONATTRIBUTETOCONNECTION element to hide an extension attribute from a connection associated with a session extension of another plug-in registered in the repository. The CONNECTIONTOEXTENSION element defines the session extension with the attribute to hide. It also defines the connection from which to hide the attribute.

The following table describes the attributes of the HIDDENEXTENSIONATTRIBUTETOCONNECTION element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| EXTNATTRIBUTEID | Required | Attribute ID of session extension attribute to hide from a connection. |

# MEDOMAIN Element

You can define a metadata extension domain to group metadata extensions. Use the MEDOMAIN element to define a metadata extension domain. For example, you can create the TIBTARGETS metadata extension domain for targets and the TIBSOURCES metadata extension domain for sources.

You can use the attributes for the MEDOMAIN element to enable or disable whether clients can view or edit the metadata extension domain.

The following table describes the attributes of the MEDOMAIN element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| NAME | Required | Enter a name for the metadata extension domain. |
| ID | Required | Enter an ID from the range of metadata extension domain IDs obtained from Informatica. |

| Attribute | Required/ Optional | Description |
|---|---|---|
| KEY | Optional | Enter an encrypted domain key for the metadata extension domain. You also use this key to access private metadata extensions. Use the *pmpasswd <password>* command to encrypt a password. |
| DESCRIPTION | Optional | Enter a description for the metadata extension domain. |
| CLIENTVISIBLE | Optional | Set to YES to enable the Designer to display the metadata extension domain. Set to NO to disable the Designer from displaying the metadata extension domain. |
| COMPONENTVERSION | Required | Enter the version of the MEDOMAIN. This allows you to keep track of updates to the MEDOMAIN element. |

The following example shows the MEDOMAIN element defining the TIBTARGETS metadata extension domain:

```
<MEDOMAIN NAME="TIBTARGETS" ID = "2" KEY = "KEY" DESCRIPTION = "TIBCO SOURCES"
CLIENTVISIBLE = "YES" CLIENTEDITABLE = "YES" ACCESSWITHOUTKEY = "YES" COMPONENTVERSION =
"1"/>
```

The MEDOMAIN element has the following child elements:

- MEDEFINITION
- LIBRARY

## MEDEFINITION Element

Metadata extensions extend the metadata stored in the repository by associating information with individual repository objects. You can use the MEDEFINITION element to define a metadata extension. For example, you have a third-party application and want to track the creation of new fields. You can create the USERNAME metadata extension to store the name of the user that creates a new field.

You can use the attributes for the MEDOMAIN element to enable the Designer to view or edit the metadata extension.

The following table describes the attributes of the MEDEFINITION element:

| Attribute | Required/ Optional | Description |
|---|---|---|
| NAME | Required | Enter a name for the metadata extension. |
| DATATYPE | Required | Enter a datatype for the metadata extension. You can enter STRING, NUMERIC, BOOLEAN, or XML.<br>**Note:** If you set the DATATYPE attribute to XML or STRING, you must set the MAXLENGTH attribute to a value greater than 0. |
| MAXLENGTH | Optional | Enter the maximum length for the metadata extension. You can specify a value up to 2,147,483,647.<br>**Note:** If you set the DATATYPE attribute to XML or STRING, you must set the MAXLENGTH attribute to a value greater than 0. |

| Attribute | Required/ Optional | Description |
|---|---|---|
| DBTYPE | Required | Enter the ID of the DBTYPE you want to use with the metadata extension. You can also enter ALL to make the metadata extension available to all DBTYPEs. |
| OBJECTTYPE | Required | Enter the name of the object type used with the metadata extension. You can enter SOURCE, TARGET, MAPPING, MAPPLET, SESSION, WORKFLOW, or WORKLET. You can also enter ALL to make the metadata extension available to all OBJECTTYPEs. |
| DEFAULTVALUE | Optional | Enter the default value for the metadata extension. |
| DESCRIPTION | Optional | Enter a description for the metadata extension. |
| ISSHAREREAD | Optional | Set to YES to enable shared reading for the metadata extension. Set to NO to disable share reading for the metadata extension. |
| ISSHAREWRITE | Optional | Set to YES to enable shared writing for the metadata extension. Set to NO to disable share writing for the metadata extension. |
| ISCLIENTVISIBLE | Optional | Set to YES to enable the Designer to display the metadata extension. Set to NO to disable the Designer from displaying the metadata extension. |
| ISCLIENTEDITABLE | Optional | Set to YES to enable the Designer to edit the metadata extension. Set to NO to disable the Designer from displaying the metadata extension. |

CHAPTER 5

# PowerExchange API Example: Java DB Adapter

This chapter includes the following topics:

## Java DB Adapter Overview

This chapter provides an example that shows how to use the PowerExchange API to develop an adapter for a data source. The adapter example connects, reads, and writes data to the Java DB database. Java DB is a JDBC compliant relational database based on the Apache Derby Java database.

The Java DB adapter uses the default PowerCenter Designer user interface to import source and target definitions for Java DB. It does not use a client plug-in. Alternatively, you can build a Java DB client plug-in that uses native drivers to get the JDBC data source.

The Java DB adapter consists of the following components:

- **Plug-in definition file.** The plug-in definition file for the Java DB adapter is named *pmJDBC.xml*. The definition file includes elements that describe the data source and define how to connect, read, and write to the data source.
- **Server plug-in.** The server plug-in file for the Java DB adapter is named *pmJDBCplugin.jar*. The server plug-in includes JDBC reader and writer extensions to access, read, and write to the Java DB relational database.

Using this example as a model, you can follow the same techniques to use the PowerExchange API to build adapters for other JDBC compliant relational databases.

### Plug-in Definition File

The pmJDBC.xml contains the following elements and attributes that define the adapter:

- Attributes that define how to connect to the database, including reader and writer properties that define the run-time configuration.
- Attributes that define the datatypes in Java DB and how the datatypes map to the PowerCenter datatypes.

- Names of the client, reader, and writer plug-in binaries and version information.
- Metadata and repository IDs specific to the plug-in.

The pmJDBC.xml must be registered with a PowerCenter repository so that the PowerCenter design and run-time environment can support the adapter.

## Server Plug-in

The server plug-in includes the run-time reader and writer components. The reader component connects to and reads from the data source. The writer component writes data to the target. The jar files for the reader and writer plug-in must be placed in the CLASSPATH or /javalib directory.

# Plug-in Definition for the Java DB Adapter

The pmJDBC.xml file conforms to the plugin.dtd structure and includes elements that define the properties of the Java DB adapter.

## PLUGIN

The PLUGIN element contains the PowerCenter repository ID attributes for the Java DB adapter. An adapter that is distributed outside an organization requires unique repository ID attributes assigned by Informatica. An adapter that is not distributed, such as the adapter example, can contain test values in the PLUGIN element.

The following table lists the PowerCenter repository ID attributes for the sample adapter:

| Attribute | Value |
|---|---|
| Plugin Id | 305050 |
| Plugin Name | PWX JDBC |
| Plugin Vendor | INFA |
| PC Version | 8.7 |

## DBTYPE

The DBTYPE element represents the database type of the source or target and contains attributes to uniquely identify the type of database.

For example, the NAME attribute identifies the name of the DBTYPE. The ID attribute refers to the database type ID. The BASEID identifies the base ID for the datatypes of this database type.

The following code shows the DBTYPE definition in the pmJDBC.xml file:

```
<PLUGIN ID="305050" NAME="JDBC" VERSION="8.7.0" VENDORID="1" VENDORNAME="Informatica"
    DESCRIPTION="PWX JDBC" >
    <DBTYPE NAME="JDBC" ID="305050" BASEID="305050"
        <DATATYPE ID ="305051" NAME ="CHAR" ODBCTYPE ="SQL_CHAR"

           INTERNALCONVERTABLE ="NO"  CANEDITPRECISION ="YES"  CANEDITSCALE ="NO"
           READONLYPRECISION ="0"  READONLYSCALE ="0"/>
:
:
    </DBTYPE>
```

DBTYPE has a DATATYPE child element that contains the attributes of the datatypes supported by Java DB. The DATATYPE child element maps JDBC datatypes in Java DB to the ODBC datatypes supported by PowerCenter.

The following code shows the some of the DATATYPE definitions in the pmJDBC.xml file:

```
<DATATYPE ID ="305051" NAME ="CHAR" ODBCTYPE ="SQL_CHAR" INTERNALCONVERTABLE ="NO"

        CANEDITPRECISION ="YES" CANEDITSCALE ="NO" READONLYPRECISION ="0"

        READONLYSCALE ="0"/>
<DATATYPE ID ="305051" NAME ="CHAR" ODBCTYPE ="SQL_WCHAR" INTERNALCONVERTABLE ="YES"/>

<DATATYPE ID ="305052" NAME ="VARCHAR" ODBCTYPE ="SQL_VARCHAR" INTERNALCONVERTABLE ="NO"

        CANEDITPRECISION ="YES" CANEDITSCALE ="NO" READONLYPRECISION ="0"

        READONLYSCALE ="0"/>
<DATATYPE ID ="305052" NAME ="VARCHAR" ODBCTYPE ="SQL_LONGVARCHAR"

        INTERNALCONVERTABLE ="YES"/>
<DATATYPE ID ="305052" NAME ="VARCHAR" DBCTYPE ="SQL_WVARCHAR"

        INTERNALCONVERTABLE ="YES"/>

<DATATYPE ID ="305053" NAME ="LONGVARCHAR" DBCTYPE ="SQL_LONGVARCHAR"

        INTERNALCONVERTABLE ="NO” CANEDITPRECISION ="YES" CANEDITSCALE ="NO"

        READONLYPRECISION ="0"  READONLYSCALE ="0"/>
<DATATYPE ID ="305053" NAME ="LONGVARCHAR" ODBCTYPE ="SQL_WLONGVARCHAR"

        INTERNALCONVERTABLE ="YES"/>
```

The following table lists the JDBC datatypes and their corresponding PowerCenter ODBC datatypes. Each of the datatype mapping has a DATATYPE element definition in the pmJDBC.xml:

| JDBC Datatypes | PowerCenter ODBC Datatypes |
|---|---|
| CHAR | SQL_CHAR |
| CHAR | SQL_WCHAR |
| VARCHAR | SQL_VARCHAR |
| VARCHAR | SQL_LONGVARCHAR |
| VARCHAR | SQL_WVARCHAR |
| LONGVARCHAR | SQL_LONGVARCHAR |
| LONGVARCHAR | SQL_WLONGVARCHAR |
| NUMERIC | SQL_DECIMAL |
| NUMERIC | SQL_NUMERIC |
| DECIMAL | SQL_DECIMAL |

| JDBC Datatypes | PowerCenter ODBC Datatypes |
|---|---|
| DECIMAL | SQL_MONEY |
| BIT | SQL_BIT |
| BOOLEAN | SQL_BIT |
| TINYINT | SQL_TINYINT |
| SMALLINT | SQL_SMALLINT |
| INTEGER | SQL_INTEGER |
| BIGINT | SQL_BIGINT |
| REAL | SQL_FLOAT |
| REAL | SQL_REAL |
| FLOAT | SQL_DOUBLE |
| DOUBLE | SQL_DOUBLE |
| BINARY | SQL_BINARY |
| VARBINARY | SQL_VARBINARY |
| LONGVARBINARY | SQL_LONGVARBINARY |
| LONGVARBINARY | SQL_IDENTITY |
| DATE | SQL_DATE |
| TIME | SQL_TIME |
| TIMESTAMP | SQL_TIMESTAMP |
| CLOB | SQL_LONGVARCHAR |
| BLOB | SQL_BINARY |

## DBTYPETOWIDGETATTR

The following code shows the DBTYPETOWIDGETATTR element definition in the pmJDBC.xml file:

```
<DBTYPETOWIDGETATTR

        OBJECTTYPE="TARGET"

        WIDGETATTRIBUTENAME="Load Scope"

        ISREADONLY="YES"

        ISDISABLED="YES"

        ISHIDDEN="NO"

        ISEXPORTED="NO"
        ALLOWALLVALUES="NO">

    <MULTIVALUEATTRIBUTE NAME="transaction"/>
</DBTYPETOWIDGETATTR>
```

The DBTYPETOWIDGETATTR element defines the Load Scope type used in the Java DB adapter to commit the target object. The Load Scope type depends on the commit type and commit interval configured in the Workflow Manager for the JDBC session. In this example, the Load Scope value is transaction.

# EXTENSION

To indicate that the Java DB adapter uses the PowerExchange API, the following attributes must be defined for the reader extension and writer extension:

- **LANG attribute.** Specifies that the programming language for the reader and writer extensions is “JAVA”.
- **CLASS NAME attribute.** Specifies the fully qualified class name for the reader and writer extensions.

For example, the Java DB adapter has a reader extension that defines the LANG and CLASS NAME attributes for the extension.

The following code shows the EXTENSION element definition in the pmJDBC.xml file:

```
<EXTENSION

        NAME ="JDBC Reader"

        DESCRIPTION ="JDBC"

        EXTENSIONTYPE ="READER"

        COMPONENTVERSION ="8.7.0"

        EXTENSIONSUBTYPE ="305050"

        SUPPORTPARTITIONS ="YES"

        LANG = "JAVA">
:
:
<CLASS NAME ="com/informatica/powerconnect/JDBC/server/reader/JDBCReaderPlugin" />
:
:
</EXTENSION>
```

## Reader Extension

The Integration Service uses the reader extension to read from a data source. You can define more than one reader extension for a data source if the data source provides multiple interfaces. The Java DB adapter requires one reader extension definition.

Use the ATTRIBUTE child element to define session attributes for the reader extension. The reader session attributes you define for an adapter are accessible from the session editor in the Workflow Manager.

The following table describes the session attributes defined for the Java DB reader extension in the pmJDBC.xml file:

| Session Attribute | Description |
|---|---|
| User Defined Join | Specifies a custom join. |
| Number Of Sorted Ports | The number of ports to be used for sorting. |
| Select Distinct | Select distinct values only. |
| Tracing Level | Tracing level for the log messages to send to the session log. |
| Pre SQL | SQL statements to run before an SQL select statement is run on the source. The Pre SQL statements use the same connection as the select statements. |
| Post SQL | SQL statements to run after an SQL select statement is run on the source. The Post SQL statements use the same connection as the select statements. |
| SQL Query | Overrides the default SQL query with a custom SQL query. |
| Source Filter | Sets a filter for the rows in the data source. |

The following code shows an ATTRIBUTE element for the reader extension in the pmJDBC.xml file:

```
<EXTENSION
        NAME ="JDBC Reader"

        DESCRIPTION ="Reader Component - PowerExchange for JDBC"

        EXTENSIONTYPE ="READER"

        COMPONENTVERSION ="1.0.0"

        EXTENSIONSUBTYPE ="305050"

        SUPPORTPARTITIONS ="Locally"

        LANG = "JAVA">

    <ATTRIBUTE NAME="User Defined Join"

        ID="1"

        TYPE="SQL"

        DATATYPE ="STRING"

        REFERENCELEVEL="DSQ"

        DESCRIPTION = "Specify custom join"

        ISREQUIRED ="NO"

        DEFAULTVALUE =""

        ISSESSIONOVERRIDABLE ="YES"

        ISINSTANCEOVERRIDABLE="YES"

        ISPARTITIONOVERRIDABLE="YES"

        ISSESSIONVARSALLOWED ="YES"

        ISSERVERVARSALLOWED="YES"
        ISVARPREFIXALLOWED="YES"

        VARIABLEPREFIX="$PWX" />
:
:

    <CLASS NAME ="com/informatica/powerconnect/JDBC/server/reader/JDBCReaderPlugin" />

    <ALLOWEDDBTYPE ISDEFAULT ="YES" DBTYPE ="305050"/>

    <ALLOWEDDBTYPE ISDEFAULT="NO" DBTYPE="15"/>
:
:
</EXTENSION>
```

## Writer Extension

The Integration Service uses the writer extension to write to a data source. You can define more than one writer extension for a data source if the data source provides multiple interfaces. The Java DB adapter requires one writer extension definition.

Use the ATTRIBUTE child element to define session attributes for the writer extension. The writer session attributes you define for an adapter are accessible from the session editor in the Workflow Manager.

The following table describes the session attributes defined for the Java DB writer extension in the pmJDBC.xml file:

| Session Attribute | Description |
|---|---|
| Insert | Insert data into the target. |
| Update | Update strategy to use when updating target data. You can use one of the following update strategies:<br>- Update as Update. Perform an update on the target.<br>- Update as Insert. Perform an insert on the target.<br>- None. Perform no operation on the target.<br>- Update else Insert. Perform an update on the target. If the update is not successful, perform an insert. |
| Delete | Delete data from the target. |
| Truncate target option | Truncate the target table before performing any operation. |
| Pre SQL | SQL statements to run before an SQL select statement is run on the target. The Pre SQL statements use the same connection as the select statements. |
| Post SQL | SQL statements to run after an SQL select statement is run on the target. The Post SQL statements use the same connection as the select statements. |

The following code shows an ATTRIBUTE element for the writer extension in the pmJDBC.xml file:

```
<EXTENSION
        NAME ="JDBC Writer"

        EXTENSIONTYPE ="READER"

        EXTENSIONSUBTYPE ="305050"

        DESCRIPTION ="Reader Component - PowerExchange for JDBC"
        COMPONENTVERSION ="1.0.0"
        SUPPORTPARTITIONS ="Locally"
        LANG = "JAVA">
    <ATTRIBUTE ID="1"
        NAME="Insert"
        DESCRIPTION = "Process updates and deletes as inserts"
        TYPE="BOOLEAN"
        DATATYPE ="NUMBER"
        ISREQUIRED ="NO"
        DEFAULTVALUE ="1"
        REFERENCELEVEL="TARGET"
        VARIABLEPREFIX="$PWX"
        ISVARPREFIXALLOWED="YES"
        ISSERVERVARSALLOWED="YES"
        ISSESSIONOVERRIDABLE ="YES"
        ISINSTANCEOVERRIDABLE="YES"
        ISPARTITIONOVERRIDABLE="YES"/>
:
:
    <CLASS NAME ="com/informatica/powerconnect/JDBC/server/reader/JDBCWriterPlugin" />
    <ALLOWEDDBTYPE ISDEFAULT ="YES" DBTYPE ="305050"/>
    <ALLOWEDDBTYPE ISDEFAULT="NO" DBTYPE="15"/>
:
:
</EXTENSION>
```

# CONNECTION

The CONNECTION element defines the attributes of the connection. The connection attributes you define for an adapter are accessible from the Connection Tab in the Workflow Manager. You can create a new JDBC connection that is relational and that can be used for a JDBC reader or writer. The connection string takes a JDBC URL that points to the database location. For example: `jdbc:derby://localhost:1527/firstdb`

## JDBC Connection Attributes

The JDBC connection contains attributes such as the user name, password, and the connection string URL. The code page is disabled. By default, the JDBC driver converts data from the database to UTF-16 format.

The following table describes the connection attributes defined in the pmJDBC.xml file:

| Connection Attribute | Description |
|---|---|
| JDBC Driver Name | JDBC driver class name to load for JDBC calls. To add the JDBC driver jar file to the CLASSPATH and load the class by default, copy the JDBC driver jar file to the following directory: `server/bin/javalib`<br>For the Java DB database, the JDBC driver name is `“org.apache.derby.jdbc.ClientDriver”` |
| Connection Environment SQL | Connection environment SQL to run each time Integration Service connects with the JDBC database. You can use this attribute to set up the environment for subsequent transactions.<br>This is an optional attribute. |
| Transaction Environment SQL | Transaction environment SQL to run each time a new transaction is started in the external database.<br>This is an optional attribute. |
| Connection Retry Period | Length of time in seconds that the Integration Service attempts to re-connect to the database if the connection fails. |

The following code shows a CONNECTION element defined in the pmJDBC.xml file:

```
<CONNECTION NAME ="PWX JDBC"
        HASCODEPAGE ="YES"
        HASUSERNAME ="YES"
        CONNECTIONTYPE ="RELATIONAL"
        HASPERMISSIONS ="NO"
        HASUSERPASSWORD ="YES"
        COMPONENTVERSION ="1.0.0"
        HASCONNECTSTRING ="YES"
        CONNECTIONSUBTYPE ="305050">
    <ATTRIBUTE ID ="1"

        NAME ="JDBC Driver Name"
        :
    </ATTRIBUTE>
</CONNECTION>
```

The reader and writer extensions are associated with the JDBC connection through the connection subtype and the corresponding extension subtype. The Connection subtype and Extension subtype are set to '305050'.

## CONNECTIONTOEXTENSION

The following code shows a CONNECTIONTOEXTENSION element defined in the pmJDBC.xml file that sets the connection properties for the reader and writer extensions:

```
<CONNECTIONTOEXTENSION
    EXTENSIONTYPE="READER"
    EXTENSIONSUBTYPE="305050"
    CONNECTIONTYPE="RELATIONAL"
    CONNECTIONSUBTYPE="305050"
    CONNECTIONNUMBER="1"
    ISDEFAULT="YES"
    SUPPORTPARTITIONS="YES"
    ISFORALLPARTITIONS ="YES"/>

<CONNECTIONTOEXTENSION
    EXTENSIONTYPE="WRITER"
    EXTENSIONSUBTYPE="305050"
    CONNECTIONTYPE="RELATIONAL"
```

```
        CONNECTIONSUBTYPE="305050"
        CONNECTIONNUMBER="1"
        ISDEFAULT="YES"
        SUPPORTPARTITIONS="YES"
        ISFORALLPARTITIONS ="YES"/>
```

## MEDOMAIN

The MEDOMAIN element contains all the metadata extension attributes. The metadata extensions enable you to add custom attributes to transformations, sources, and targets required to support the adapter. The metadata extension attributes you define for an adapter are accessible from the Source Qualifier editor in a mapping that contains a Java DB source.

The following table describes the metadata extensions defined for the Java DB writer extension in the pmJDBC.xml file:

| Metadata Extension | Description |
|---|---|
| User Defined Join | Specifies a custom join. |
| Number Of Sorted Ports | The number of ports to be used for sorting. |
| Select Distinct | Select distinct values only. |
| SQL Query | Overrides the default SQL query with a custom SQL query. |
| Source Filter | Sets a filter for the rows in the data source. |

The following code shows the MEDOMAIN element defined in the pmJDBC.xml file:

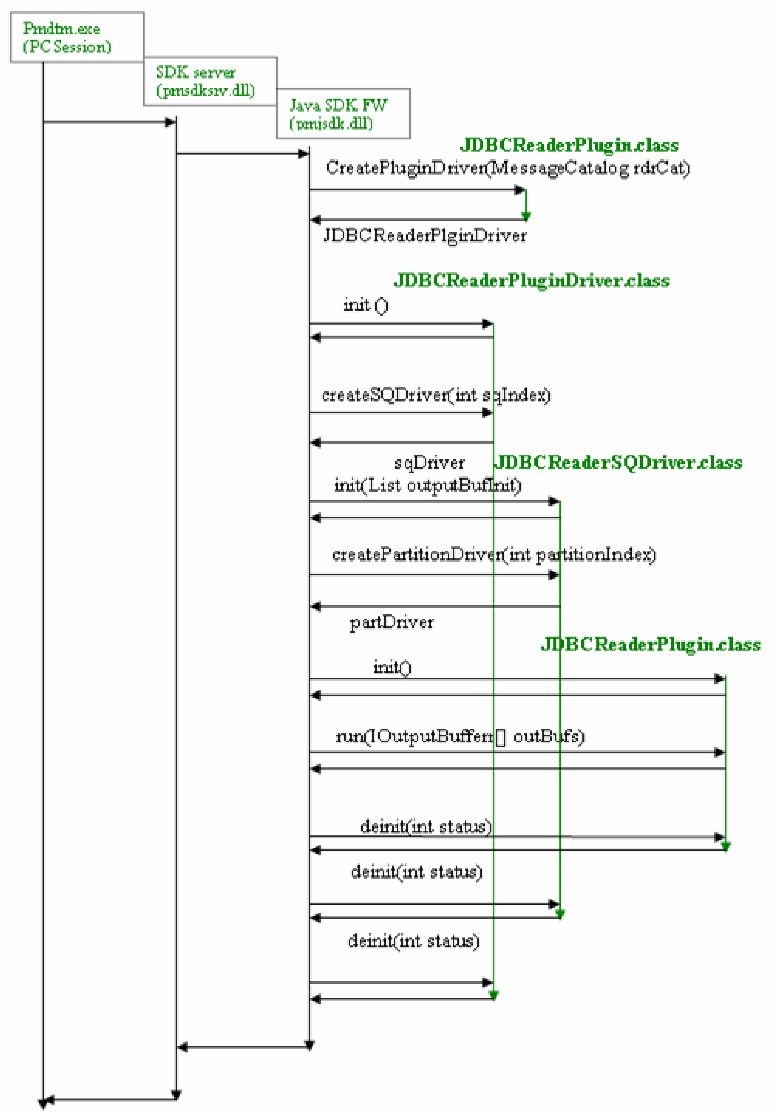
```
<MEDEFINITION
    NAME = "Select Distinct"
    DATATYPE = "STRING"
    MAXLENGTH = "3"
    OBJECTTYPE = "APPLICATIONDSQ"
    DEFAULTVALUE = ""
    DESCRIPTION = "Select Distinct"
    ISSHAREREAD = "YES"
    ISSHAREWRITE = "YES"
    ISCLIENTVISIBLE = "YES"
    ISCLIENTEDITABLE = "YES"/>
```

# Objects and Methods in the Java DB adapter

When you run a workflow in PowerCenter that reads or writes to a Java DB database, it starts a reader or writer session of the Java DB adapter. This section discusses the classes and methods involved in the reader session and the writer session.

## Reader Session

The following diagram shows the sequence of calls made during a reader session:



The JDBC reader session starts with the JDBC source in the mapping. The session (pmdtm.exe) loads the PowerExchange API server framework (pmsdksrv.dll) and the PowerExchange API for Java framework (pmjsdk.dll). The PowerExchange API for Java framework reads the Java class name from the repository as defined in the CLASSNAME of the reader extension section in the pmJDBC.xml:

```
<CLASS NAME ="com/informatica/powerconnect/JDBC/server/reader/JDBCReaderPlugin" />
```

The PowerExchange API for Java framework searches for this class in the CLASSPATH or PowerCenter / javalib directory and then calls the CreatePluginDriver() method. The method returns a JDBCReaderPluginDriver object `(com/informatica/powerconnect/JDBC/server/reader/JDBCReaderPlugindriver).`

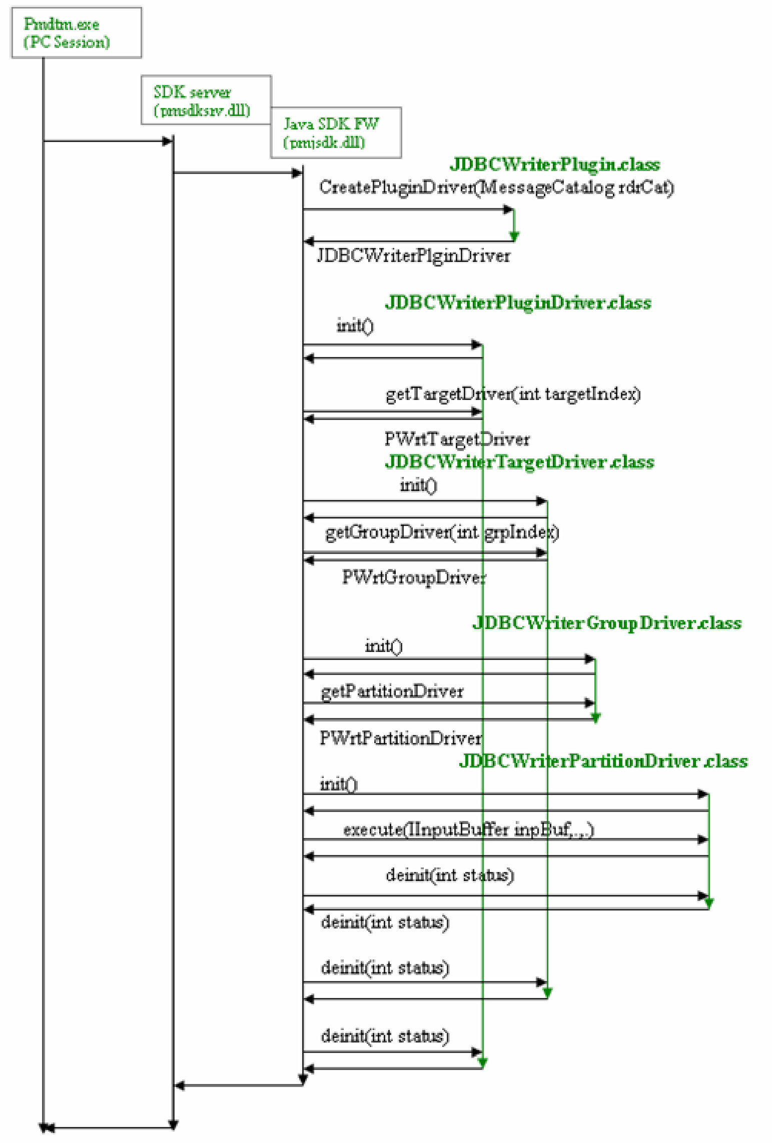
The PowerExchange API for Java framework initializes the JDBCReaderPlugindriver object and calls the CreateSQDriver method, which returns a JDBCReaderSQDriver object reference. The PowerExchange API for Java framework initializes the JDBCReaderSQDriver object and calls the createPartitionDriver method. The createPartitionDriver method gets the Source Qualifier field metadata from the session extension and creates the field list. The field list is equivalent to the field list in a JDBC source created in the Designer. The createPartitionDriver method also creates the main JDBCReaderPartitionDriver that runs the SQL Select query against the database specified in the connection string configured for the session.

The PowerExchange API for Java framework initializes the JDBCReaderPartitionDriver object and calls the run method with the OutputBuffer(IOutputBuffer) as parameter. The run method runs the reader query and loads the resultset in the OutputBuffer. The data in the OutputBuffer can be used in a transformation and then written to the target.

The PowerExchange API for Java framework deinitialize the JDBCReader objects in LIFO order.

## Writer Session

The following diagram shows the sequence of calls made during a writer session:


Pmdtm.exe (PC Session)
SDK server (pmsdksrv.dll)
Java SDK FW (pmjsdk.dll)
JDBCWriterPlugin.class
CreatePluginDriver(MessageCatalog rdrCat)
JDBCWriterPlginDriver
JDBCWriterPluginDriver.class
init()
getTargetDriver(int targetIndex)
PWrtTargetDriver
JDBCWriterTargetDriver.class
init()
getGroupDriver(int grpIndex)
PWrtGroupDriver
JDBCWriterGroupDriver.class
init()
getPartitionDriver
PWrtPartitionDriver
JDBCWriterPartitionDriver.class
init()
execute(IInputBuffer inpBuf,..)
deinit(int status)
deinit(int status)
deinit(int status)
deinit(int status)

The JDBC writer session is called when a mapping or session contains a JDBC target. The session (pmdtm.exe) loads the PowerExchange API server framework (pmsdksrv.dll) and the PowerExchange API for

Java framework (pmjsdk.dll). The PowerExchange API for Java framework reads the Java class name from the repository as defined in the CLASSNAME of the writer Extension section in the pmJDBC.xml:

```
<CLASS NAME ="com/informatica/powerconnect/JDBC/server/writer/JDBCWriterPlugin" />
```

The PowerExchange API for Java framework searches for this class in the CLASSPATH or PowerCenter /javalib directory and then loads the class from the jar file (pmJDBCplugin.jar). The PowerExchange API for Java framework initializes the JDBCWriterPlugin object and creates the JDBCWriterTargetDriver. The PowerExchange API for Java framework calls the getGroupDriver method, which creates the JDBCWriterGroupDriver.

The PowerExchange API for Java framework initializes the JDBCWriterGroupDriver object and calls the createWriterPartitionDriver method, which creates the linked target field vector and passes it to the WriterPartitionDriver object.

The PowerExchange API for Java framework initializes the JDBCWriterPartitionDriver object and calls the run method with the InputBuffer as parameter. It prepares the required SQL query, binds the data in the prepared queries and loads to the target table.

The PowerExchange API for Java framework deinitialize the JDBCWriter objects in LIFO order.

# Adapter Processes

## Datatype Conversion

The JDBC datatypes must be converted to the PowerCenter datatypes for processing. The Source Qualifier in the Designer displays the JDBC datatypes and converts them to the PowerCenter datatypes.

The following table lists the JDBC datatypes and the equivalent PowerCenter datatypes:

| JDBC Datatypes | PowerCenter Datatypes |
|---|---|
| Char | String |
| Varchar | String |
| LongVarchar | String |
| Numeric | Decimal |
| Decimal | Decimal |
| Real | Float |
| Double | Double |
| Float | Double |
| Date | Date/time |
| Int | Integer |
| Small int | Small integer |
| Long | Text |

| JDBC Datatypes | PowerCenter Datatypes |
|---|---|
| Binary | Binary |
| Varbinary | Binary |
| LongVarBinary | Longvarbinary |
| Bit | String (Precision 1) |
| Boolean | String (Precision 1) |
| Blob | Binary |
| Clob | Text |

The following table shows the possible datatype conversions from JDBC to PowerCenter:

| JDBC Datatypes | PowerCenter Datatypes | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | boolean | short | int | long | float | double | String | Date/ Time | Binary |
| boolean | X | | | | | | X | | |
| Short /<br>Small int | | X | X | X | | | X | | |
| Int | | | X | X | X | X | X | | |
| long | | | | X | X | X | X | | |
| float | | | | | X | X | X | | |
| double | | | | | | X | X | | |
| String | X | X | X | X | X | X | X | X | |
| Date/Time | | | | | | | X | X | |
| Binary | | | | | | | | | X |

## Reader and Writer Buffer

The JDBC reader and writer handle the buffer for the partition driver at run-time.

The JDBCReaderPartitionDriver object uses an IOutputBuffer outBuff(reader Buffer) as input parameter. The output buffer gets the data returned by the SELECT query from the reader data source. The JDBCReaderPartitionDriver object callsIOutputBuffer.readData() for each column and stores data in the output buffer for each row of the resultset object.

The JDBCWriterPartitionDriver object uses an IInputBuffer inpBuff(writer buffer) with data from the reader as input parameter. The prepared statements for insert, delete, and update processes store data in the input buffer and updates the target database.

## Commit Type and Commit Interval

JDBC provides functions to enable or disable auto commit on a database. You can manage the commit type and set an interval to delay the commit function. You can select a source-based or target-based commit and configure a commit interval in the Workflow Manager task properties.

During a source-based commit session, the Integration Service commits data to the target based on the number of rows from active sources in a target load order group. These rows are referred to as source rows. When the Integration Service runs a source-based commit session, it identifies the commit source for each pipeline in the mapping. The Integration Service generates a commit row from these active sources at every commit interval.

During a target-based commit session, the Integration Service commits rows based on the number of target rows and the key constraints on the target table. The commit point depends on the following factors:

- **Commit interval.** The number of rows to use as a basis for commits. Configure the target commit interval in the session properties.
- **Writer wait timeout.** The amount of time the writer waits before it issues a commit. Configure the writer wait timeout when you set up the Integration Service in the Administration Console.
- **Buffer Blocks.** Blocks of memory that hold rows of data during a session. You can configure the buffer block size in the session properties. You cannot configure the number of rows that the block holds.

When you run a target-based commit session, the Integration Service can issue a commit before, on, or after the configured commit interval. The Integration Service uses the following process to determine when to issue commits:

- When the Integration Service reaches a commit interval, it continues to fill the writer buffer block. When the writer buffer block fills, the Integration Service issues a commit.
- If the writer buffer fills before the commit interval, the Integration Service writes to the target, but waits to issue a commit. It issues a commit when one of the following conditions is true:
  - The writer is idle for the amount of time specified by the Integration Service writer wait timeout option.
  - The Integration Service reaches the commit interval and fills another writer buffer.

## Partition Support

The Java DB adapter implements pass-through partitions. You can create multiple pass-through partitions for a session. Each partition runs on a separate thread. For each partition, specify the JDBC reader and writer session attributes such as custom query and the number of sorted ports. All partitions share the same connection.

## Error Handling

All errors or exceptions are written to a session log. The Java DB adapter creates a message catalog object with a message file name. The message file contains the messages for each error, warning, or information.

The following code shows how the message catalog is created:

```
/** Message catalog intialization*/
MessageCatalog rdrCat = new MessageCatalog(JDBCCmnConstants.JDBC_MSG_PREFIX,

          ReaderConstants.JDBC_RDR_MSGFILE);
```

When an exception occurs, the Java DB adapter reads the error message from the message file and the logs it to the session log.

The following code shows how an error message is written to the session log:

```
//get the sq instnce
IAppSQInstance appSqInstace = (IAppSQInstance)lstSqInstances.get(sqIndex);
//get session extn for the dsq
ISessionExtension sessExtn = session.getExtension(EWidgetType.DSQ, sqIndex);
if(sessExtn == null){
    //there is no error or exception given by the frame work.
    //so we have to make this check and throw SDK Exception if needed
    String dsqName = appSqInstace.getName();
    SDKMessage msg = rdrCat.getMessage("2002_ERR_SESS_EXT", dsqName);
    throw new SDKException(msg);
}
```

The message for error 2002_ERR_SESS_EXT is extracted from JDBC_RDR_MSGFILE (JDBCRdrMsg.xml) and logged to the session log.

The following code shows another type of message written to the session log:

```
try
    {
    m_resultset = m_Stmt.executeQuery(jdbcStmt);
    }
catch (SQLException ex) {
    msg =rdrCat.getMessage("2027_SELECT_QRY_EXEC_FAIL",ex.getMessage());
    JDBCPluginException.JDBCSQLException(utilsSrv, ex);
    throw new JDBCPluginException(msg);
}
```

The generic exception in the JDBC reader is generated from running a query. The message for error 2027_SELECT_QRY_EXEC_FAIL is extracted from the JDBCRdrMsgs.properties file and logged to the session log.

The following code shows an informational message written to the session log:

```
SDKMessage msg = rdrCat.getMessage("2001_DSQ_CREATED",appSqInstace.getName());
utilsSrv.logMsg(ELogMsgLevel.INFO, msg);
```

# Using the Java DB Adapter

Before you use the Java DB Adapter, verify that you have the Java DB database installed and you have a user account with permissions to read and write to tables in the Java DB database.

You can use the Java DB adapter in the same way that you use other PowerExchange adapters.

To use the Java DB adapter, complete the following steps:

1. Install the PowerExchange API files.
2. Download and install the Java DB adapter files.

   You can download the Java DB adapter files from the Informatica Technology Network web site. Copy the server plug-in jar files in the /javalib directory and add the location to the CLASSPATH.
3. Register the server and client plug-ins.
4. Build the mapping and run the workflow as you would other PowerCenter mappings and workflows.

   In the Designer, import the source and target definitions for the Java DB tables and create the mapping to transform data based on your requirements.

   In the Workflow Manager, create and configure the Java DB connection, session, and workflow.

5. Run the workflow and verify that it has successfully accessed data in the Java DB database.

CHAPTER 6

# PowerExchange API Example: Bulk Loader Transformation

This chapter includes the following topics:

## Bulk Loader Transformation Example Overview

To load a large volume of data to a database, you can load the data in stages. In the first stage, move the data to a file. In the second stage, you can use a database bulk loader utility to move the data to a database. In PowerCenter, you can perform both stages of the process in a mapping. You can create a custom transformation to move data to a file and then call the bulk loader to move the data to a database. Keeping the process in a mapping allows you to easily define and manage the table structure and avoid manual processes.

This example shows how to create a custom transformation to load a large volume of data into a database and call the custom transformation from a PowerCenter mapping. The example loads data from a text file into a MySQL database using the MySQL bulk loader. You can use the same technique to load bulk data into other databases that have bulk loader utilities

The example uses the following components to perform the bulk load:

- Bulk loader custom transformation. This custom transformation writes source data to a text file and then calls the bulk loader utility to move the data to the MySQL database. The bulk loader custom transformation consists of server and client DLL files. It requires a plug-in XML that you must register in a PowerCenter repository.
- Mapping that calls the bulk loader custom transformation. After you register the bulk loader custom transformation, you can create a mapping to call the transformation. The attributes available for the custom transformation instance in the mapping correspond to the attributes you set in the plug-in XML.

# Bulk Loader Transformation

The bulk loader custom transformation encapsulates the processes of writing the data to a text file and invoking the bulk loader utility with the given parameters.

## Creating the Bulk Loader Transformation

Create the bulk loader custom transformation before you create the PowerCenter mapping. The custom transformation requires a plug-in XML file and client and server components. The attributes you define in the plug-in XML file and components determine the attributes that are available for the transformation when you use it in a PowerCenter mapping.

To create the bulk loader transformation, complete the following steps:

1. Determine the structure of the data to be moved to the database table.
2. Determine the command and options for the bulk loader utility of the database.
3. Create the plug-in XML file.
4. Create the custom transformation client component.
5. Create the custom transformation server component.

## Data Structure

Determine the source data you want to use. The structure of the source data determines the target ports and the structure of the text file you create. The text file determines the structure of the database table where you load the data. When you create the mapping in PowerCenter, you can create or import the structure of the source data.

## Bulk Loader Command and Options

The bulk loader transformation uses the PowerCenter file writer to write data to a text file. After it creates the file and receives the end-of-file notification, it uses the mysqlimport client program to write the data to the database. The mysqlimport client program is a utility provided by MySQL to call the LOAD DATA command of the MySQL bulk loader.

The mysqlimport utility creates a database table and names it based on the name of the text file that it reads the data from. In this example, the name of the database table is the same as the name of the text file.

The example uses the following command to run the MySQL bulk loader:

```
mysqlimport
--local
--fields-terminated-by=,
--lines-terminated-by="\r\n"
--host=localhost
--port=3306
MySQLDB
$OutputFileName
--user=root
--password=MyPassword
--delete
--columns=F1, F2, F3, F4, F5
```

The following table describes some components of the command:

| Option | Description |
|---|---|
| mysqlimport | Client program that provides a command line interface to the LOAD DATA command. The options for mysqlimport correspond to clauses of LOAD DATA command syntax. The mysqlimport client program is located in the /bin directory of the MySQL directory. |
| MySQLDB | Name of the MySQL database into which the bulk loader loads the data. |
| $OutputFileName | Text file from which the bulk loader reads the data it moves into the database table.<br>In this example, the variable $OutputFileName contains the name of the text file, which is equivalent to the name of the table to which mysqlimport utility writes the data. This variable is also used in the Output Filename attribute of the target and the Datafile attribute of the custom transformation in the PowerCenter session. |
| --user=root | User name to log in to the database. |
| --password=MyPassword | Password for the user name. |
| --columns=F1, F2, F3, F4, F5 | List of columns in the database table. The column names in the list must be the same as the column names for the target table and the port names in the custom transformation. In this example, you can drag and drop the ports from the source to the custom transformation to create ports with the correct names. |

The parameters for the mysqlimport utility are included in the plug-in XML file for the bulk loader transformation. They determine the metadata extension attributes available for the transformation in a PowerCenter mapping. The name of the text file from which you load data must match the target file name. The port names of the custom transformation must match the column names of the target in the mapping.

When you build a bulk loader custom transformation for another database based on this example, the parameter requirements may be different. Read the documentation for the database to determine the requirements of the bulk loader utility.

# Bulk Loader Transformation Plug-In XML File

To register a custom transformation with a PowerCenter repository, create a plug-in XML file that contains the attributes of the transformation. The plug-in XML defines the properties of the custom transformation and provides a unique identity for the transformation.

The plug-in XML for the example custom transformation includes the parameters for the MySQL bulk loader. If you implement a bulk loader custom transformation for another database, the plug-in XML must include the bulk loader parameters for the database that you are implementing.

Many attributes in the plug-in XML for the bulk loader custom transformation example are disabled to prevent misuse. If you set the session extension attribute ISSESSIONOVERRIDABLE and the metadata extension attribute ISCLIENTVISIBLE to YES, you can override the disabled attributes in the PowerCenter Client tools. The plug-in XML for the example custom transformation sets the ISSESSIONOVERRIDABLE and ISCLIENTVISIBLE attributes to YES so you can modify the values in the Designer and Workflow Manager.

**Note:** The following XML code snippets do not constitute the full content of the plug-in XML file and may not be syntactically correct.

The plug-in XML for the example custom transformation is named pmbulkloadtransform.xml and includes the following definitions and session extensions:

```
PLUGIN NAME ="Bulk Load Transformation" ID="305150" VENDORNAME ="Informatica" VENDORID
="1"   DESCRIPTION ="Bulk Load Transformation" VERSION ="8.6.1">
```

```
<!--**************** BulkLoad Template Extension *********************-->
<EXTENSION NAME="Bulkload Session Extension" DESCRIPTION="Bulkload Transform Session
Extension." EXTENSIONTYPE="TEMPLATEEXTENSION" COMPONENTVERSION="1.0.0"
EXTENSIONSUBTYPE="305150" SUPPORTPARTITIONS="NO">
   <ATTRIBUTE ID ="1"
       NAME="LOADERPATH"
       DESCRIPTION ="Full Directory path of the third party Loader Executable."
       TYPE ="PROPERTY"
       DATATYPE ="STRING"
       ISREQUIRED ="YES"
       DEFAULTVALUE ="C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqlimport"
       REFERENCELEVEL ="TRANSFORMATION"
       VARIABLEPREFIX =""
       ISVARPREFIXALLOWED ="NO"
       ISSERVERVARSALLOWED ="NO"
       ISSESSIONOVERRIDABLE ="YES"
       ISSESSIONVARSALLOWED ="NO"
       ISVARFULLNAMEALLOWED ="NO"
       ISINSTANCEOVERRIDABLE ="YES"
       ISPARTITIONOVERRIDABLE ="YES"/>
```

The plug-in XML file also defines other session extension attributes for the custom transformation. The following shows the attribute description and default value for the session attributes. The list does not show all the properties of the session attributes that are set in the plug-in XML file:

```
 <ATTRIBUTE ID ="2" NAME="LOCATION"
       DESCRIPTION ="Location of the Database remote/local"
       DEFAULTVALUE ="--local"

   <ATTRIBUTE ID ="3" NAME="FIELDSEPARATOR"
       DESCRIPTION ="Fields/Columns terminator in the data file."
       DEFAULTVALUE ="--fields-terminated-by=,"

   <ATTRIBUTE ID ="4" NAME="LINETERMINATOR"
       DESCRIPTION ="Field Data Lines terminator in the Data File"
       DEFAULTVALUE ="--lines-terminated-by=\r\n"

   <ATTRIBUTE ID ="5" NAME="HOSTNAME"
       DESCRIPTION ="Server Name, where the database resides."
       DEFAULTVALUE ="--host=localhost"

   <ATTRIBUTE ID ="6" NAME="HOSTPORT"
       DESCRIPTION ="Server Port Where the Database target DB resides."
       DEFAULTVALUE ="--port=3306"

   <ATTRIBUTE ID ="7" NAME="DATABASENAME"
       DESCRIPTION ="Database Name where the target table
resides."
       DEFAULTVALUE =""

   <ATTRIBUTE ID ="8" NAME="DATAFILE"
       DESCRIPTION ="File name where the intermediate data is stored before passing to
the Loader."
       DEFAULTVALUE ="$OutputFileName"

   <ATTRIBUTE ID ="9" NAME="USER"
       DESCRIPTION ="User Name of the database."
       DEFAULTVALUE ="root"

   <ATTRIBUTE ID ="10" NAME="PASSWORD"
       DESCRIPTION ="Password of the user of the Database."
       DEFAULTVALUE =""

   <ATTRIBUTE ID ="11" NAME="COLUMNS"
       DESCRIPTION ="Field names separated by Filed Terminator."
      DEFAULTVALUE ="--columns="

   <ATTRIBUTE ID ="12" NAME="OTHERPARAMS"
       DESCRIPTION ="Other Required parameters if any for the specific Loader."
       DEFAULTVALUE ="--delete"
   <ATTRIBUTE ID ="13" NAME="Enable Parameter File"
       DESCRIPTION ="This enables the CT to get the Loader parameters from the file
mentioned in the ParameterFileName attribute instead of Metadata/session
attributes."
       DEFAULTVALUE = "0"

   <ATTRIBUTE ID ="14" NAME="ParameterFileName"
       DESCRIPTION ="parameter file overrides the session/metadata Loader params if
enabled."
       DEFAULTVALUE ="BulkLoadParam.prm"

    <ALLOWEDTEMPLATE TEMPLATEID="305150"/>
</EXTENSION>

<!-- ************ Bulkload METADATA EXTENSION ATTRIBUTES *********** -->
<MEDOMAIN NAME="BULKLOAD_Transform_Domain"
   ID="305150"
   DESCRIPTION="BULKLOAD Transform Domain"
   CLIENTVISIBLE="YES"
   COMPONENTVERSION="1.0.0">

<! -- ************** Third party loader executable path ************* -->
<MEDEFINITION NAME="LOADERPATH"
   TEMPLATEID="305150"
   DATATYPE="STRING"
   MAXLENGTH="1024"
   DEFAULTVALUE = "C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqlimport"
   OBJECTTYPE="Custom Transformation"
   ISSHAREREAD="YES"
   ISSHAREWRITE="YES"
   ISCLIENTVISIBLE="YES"
   ISCLIENTEDITABLE="YES"/>
```

The plug-in XML file defines metadata extension attributes for the custom transformation. The following list shows the attribute description and default value for the metadata extension attributes. The list does not show all the properties of the metadata extension attributes that are set in the plug-in XML file:

```
<! -- ***************** Location of DB ***************** -->
<MEDEFINITION NAME="LOCATION"
   DEFAULTVALUE = "--local"

<! -- ***************** Field Terminator Comma ***************** -->
<MEDEFINITION NAME="FIELDSEPARATOR"
   DEFAULTVALUE = "--fields-terminated-by=,"

<! -- ***************** Line terminator \r\n ***************** -->
<MEDEFINITION NAME="LINETERMINATOR"
   DEFAULTVALUE = "--lines-terminated-by=\r\n"

<!-- ***************** Host Name of the DB ***************** -->
<MEDEFINITION NAME="HOSTNAME"
   DEFAULTVALUE = "--host=localhost"

<!-- ***************** Port of the DB Host ***************** -->
<MEDEFINITION NAME="HOSTPORT"
   DEFAULTVALUE = "--port=3306"

<!-- ***************** Databse Name ***************** -->
<MEDEFINITION NAME="DATABASENAME"
   OBJECTTYPE="Custom Transformation"

<!-- ************ Datafile path needs to be same as target file name ************* -->
<MEDEFINITION NAME="DATAFILE"
   DEFAULTVALUE = "$OutputFileName"

<!-- ***************** User Name ***************** -->
   <MEDEFINITION NAME="USER"
   DEFAULTVALUE = "--user="

<!-- ***************** Password **************** -->
<MEDEFINITION NAME="PASSWORD"
```

```
   DEFAULTVALUE = "--password="

<!-- ********* Column names will be populated from the CT ports programmatically
********* -->
<MEDEFINITION NAME="COLUMNS"
   DEFAULTVALUE = "--columns="

<!-- ************** Other parameters that the third party loader takes ************** -->
<MEDEFINITION NAME="OTHERPARAMS"
   DEFAULTVALUE = "--delete"
:
:
</MEDOMAIN>


        <!-- *************************** Bulkload MGEP
Template******************************** -->
        <TEMPLATE TYPE="CUSTOMTRANSFORMATIONTEMPLATE"
                      ID="305150"
                      NAME="Bulk Load Transformation"
                      COMPONENTVERSION="1.0.0"
                      DESCRIPTION="CT based Bulk Load Transformation" >

        <TEMPLATEATTRIBUTE NAME="Language" VALUE="C++"  ISCLIENTEDITABLE = "NO"/>
        <TEMPLATEATTRIBUTE NAME="Class Name" VALUE="BulkLoadCTPlugin" ISCLIENTEDITABLE =
"NO"/>
        <TEMPLATEATTRIBUTE NAME="Module Identifier" VALUE="pmbulkloadtrans"
ISCLIENTEDITABLE = "NO"/>
        <TEMPLATEATTRIBUTE NAME="Is Partitionable" VALUE="No"  ISCLIENTEDITABLE = "NO"/>
        <TEMPLATEATTRIBUTE NAME="Inputs Must Block" VALUE="NO"  ISCLIENTEDITABLE = "NO"/>
        <TEMPLATEATTRIBUTE NAME="Is Active" VALUE="YES"  ISCLIENTEDITABLE = "NO"/>
        <TEMPLATEATTRIBUTE NAME="Transformation Scope" VALUE="Row" ISCLIENTEDITABLE =
"NO"/>
        <TEMPLATEATTRIBUTE NAME="Generate Transaction" VALUE="NO"  ISCLIENTEDITABLE =
"NO"/>
        <TEMPLATEATTRIBUTE NAME="Update Strategy Transformation" VALUE="YES"
ISCLIENTEDITABLE = "NO"/>
        <TEMPLATEATTRIBUTE NAME="Output Is Repeatable" VALUE="Based On Input Order"
ISCLIENTEDITABLE = "NO"/>
        <TEMPLATEATTRIBUTE NAME="Requires Single Thread Per Partition" VALUE="NO"
ISCLIENTEDITABLE = "NO"/>
        <TEMPLATEATTRIBUTE NAME="Output Is Deterministic" VALUE="YES"  ISCLIENTEDITABLE
= "NO"/>

        <LIBRARY NAME = "pmbulkloadtransform" OSTYPE = "NT" TYPE = "CLIENT" />
        <LIBRARY NAME = "pmbulkloadvldn.dll" OSTYPE = "NT" TYPE = "VALIDATION" />
        <LIBRARY NAME = "libpmbulkloadvldn.sl" OSTYPE = "HPUX" TYPE = "VALIDATION" />
        <LIBRARY NAME = "libpmbulkloadvldn.so" OSTYPE = "SOLARIS" TYPE = "VALIDATION" />
        <LIBRARY NAME = "libpmbulkloadvldn.so" OSTYPE = "LINUX" TYPE = "VALIDATION" />
        <LIBRARY NAME = "libpmbulkloadvldn.a" OSTYPE = "AIX" TYPE = "VALIDATION" /
>

        </TEMPLATE>
 </PLUGIN>
```

## Bulk Loader Client

To modify the PowerCenter Client interface so that it supports the custom transformation, create a client plug-in. The client plug-in allows you to define the columns of the target as input or output ports and add the bulk loader parameters to the metadata extension for the custom transformation.

Compile the code for the client and copy the DLL file to the `\client\bin` directory of the PowerCenter Client.

After you create the client DLL file, register it in the Windows registry. Register the client DLL in the HKEY_LOCAL_MACHINE root key. The following registry values show the Windows registration for the bulk loader custom transformation example:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Informatica\PowerMart Client Tools\8.6.1\PlugIns
\Informatica]
"PC_BULKLOADT"="pmbulkloadtransform.dll"
```

# Bulk Loader Server

To enable the custom transformation to read data from a source or write data to a target, create a server plug-in. The server plug-in contains the methods to manipulate the input and output buffer and pass the data to the target. Compile the code for the server plug-in and copy the DLL file to the `/server/bin` directory in PowerCenter.

The server plug-in allows the custom transformation to pass data to the file writer and create the command to invoke the bulk loader. After the file is written, the custom transformation reads the command and required parameters and invokes the bulk loader during the deinit() call.

The bulk loader custom transformation example includes a file named BulkLoadCTPartitionDriver.cpp with the code to pass data to the file writer and invoke the MySQL bulk loader. To create a bulk loader custom transformation for another database, you can modify the code to call the bulk loader utility of the new database with the appropriate parameters.

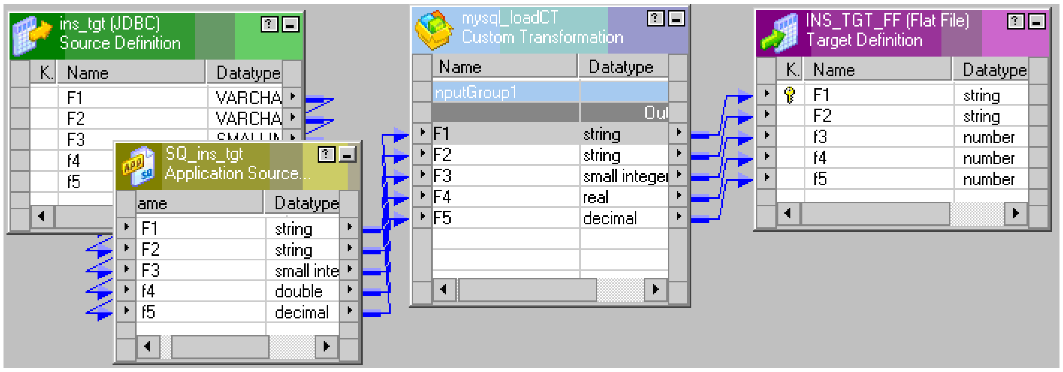
The following snippet from the server plug-in code shows how the custom transformation invokes the bulk loader:

```
ISTATUS BulkLoadCTPartitionDriver::deinit(void)
{
    ...

    //Third party loader process is created and executed.
    if(createLoaderProcess() != ISUCCESS)
    {
         return IFAILURE;
    }

    return ISUCCESS;

}

/* Creates the third party loader parameters and executes the process*/
ISTATUS BulkLoadCTPartitionDriver::createLoaderProcess()
{
    HANDLE     nProcessID;

    //Get the Bulkload CT session extension attributes defined as third party Loader
Parameters.
    IINT32 isParamFileEnabled = 0;

    //get target table Columns
    if (IFAILURE == m_pSessExtn->getAttributeInt(gBulkLoad_ParamFileEnabled,
isParamFileEnabled))
    {
          return IFAILURE;
    }

    //If true, read the Loader attributes from the parameter file else get from metadata
extension and session attributes
    if(isParamFileEnabled)
    {
        if (IFAILURE == m_pSessExtn->getAttribute(gBulkLoad_ParamFileName,
m_sBulkLoadParameterFileName))
        {
            return IFAILURE;
        }
        initParamFileAttrs(m_sBulkLoadParameterFileName);
    }
    else
    {
        //get the CT bulk load session extension attributes
        initBulkLoadSessAttrs();

        //get the CT bulk load metadata extension attributes.
        initBulkLoadMetaExtAttrs();
    }

    PmUString sExeName;
    //Override the CT bulk load session extension attribute "Loader Exe Path" with 
metadata extension attributes.
    if(m_sSessExtnLoaderExe.isEmpty() || m_sSessExtnLoaderExe.getLength() == 0)
    {
        sExeName = m_sLoaderExe + gBulkLoad_space;
    }
    else
    {
        sExeName = PmUString(m_sSessExtnLoaderExe) + gBulkLoad_space;
    }

    //Command line parameters for the Loader
    PmUString sCmdLineParams ;

    //Override CT bulk load session extension attribute "Location" with session 
extension attributes.
    if(m_sSessExtnLocation.isEmpty() || m_sSessExtnLocation.getLength() == 0)
    {
        sCmdLineParams += m_sLocation + gBulkLoad_space;
    }
    else
    {
        sCmdLineParams += PmUString(m_sSessExtnLocation) + gBulkLoad_space;
    }


        .
        .
        .


    IUString sCmnLine;
    if(!isParamFileEnabled)
    {
        if(m_pBulkLoadUtils->getUtilsServer()->expandString(sCmdLineParams.buffer(), 
sCmnLine,
            IUtilsServer::EVarParamSession, IUtilsServer::EVarExpandDefault) != ISUCCESS)
        {
            return IFAILURE;
        }
        //Start the Loader process by passing the command line parameters
        nProcessID = BLExtrProcess::start(sExeName,PmUString(sCmnLine), 
m_pILog,m_pCatalog);

    }
    else
    {
        //Start the Loader process by passing the command line parameters
        nProcessID = BLExtrProcess::start(sExeName,sCmdLineParams, m_pILog,m_pCatalog);


    }

        if ( nProcessID == 0 )
    {
        return IFAILURE;
    }
    if (BLExtrProcess::finish(nProcessID, m_pILog,m_pCatalog,sExeName) != ISUCCESS)
    {
        return IFAILURE;
    }

    return ISUCCESS;
}
```

# Bulk Loader Mapping

The example mapping for the bulk loader is a pass through mapping that reads data from the source and writes the data to a file.

The following figure shows the pass-through mapping with the bulk loader custom transformation:



ins_tgt (JDBC)
Source Definition

| K | Name | Datatype |
|---|---|---|
| | F1 | VARCHA |
| | F2 | VARCHA |
| | F3 | SMALLIN |
| | f4 | |
| | f5 | |

SQ_ins_tgt
Application Source...

| ame | Datatype |
|---|---|
| F1 | string |
| F2 | string |
| F3 | small inte |
| f4 | double |
| f5 | decimal |

mysql_loadCT
Custom Transformation

| Name | Datatype |
|---|---|
| nputGroup1 | |
| | Out |
| F1 | string |
| F2 | string |
| F3 | small intege |
| F4 | real |
| F5 | decimal |

INS_TGT_FF (Flat File)
Target Definition

| K | Name | Datatype |
|---|---|---|
| | F1 | string |
| | F2 | string |
| | f3 | number |
| | f4 | number |
| | f5 | number |

The target for the mapping is a flat file target. The target file is the text file that the custom transformation writes data to. When the custom transformation completes writing to the target, it invokes the bulk loader utility and writes the data from the text file to the database. The name of the text file is the same as the name of the database table that the bulk loader writes to.

Since the mapping is a pass through mapping, the column names are linked from the source to the custom transformation and from the custom transformation to the target.

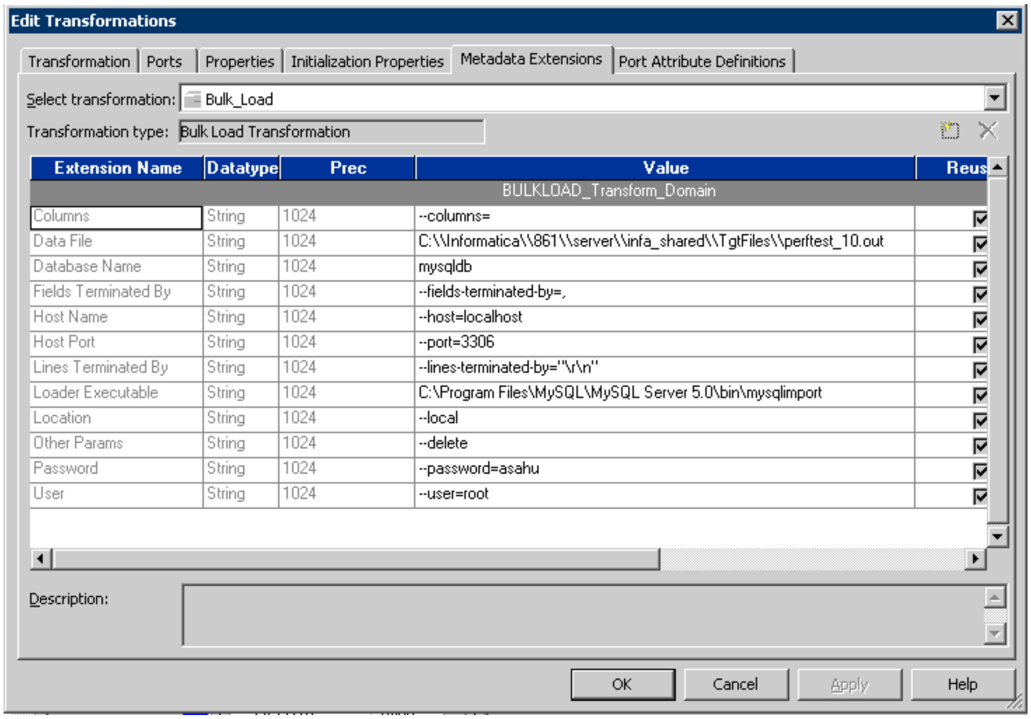
## Setting the Values for the Bulk Loader Parameters

When you create the mapping and workflow for the bulk loader, you must set the values of the parameters required by the custom transformation.

You can set the values of the custom transformation parameters in one of the following ways:

- In the mapping, set the parameter values in the Metadata Extension tab of the bulk loader custom transformation. The Metadata Extension tab displays the bulk loader parameters defined in the plug-in XML file.

  The following figure shows the parameters that display in the Metadata Extension tab when you edit the bulk loader custom transformation:



Select transformation: Bulk_Load

Transformation type: Bulk Load Transformation

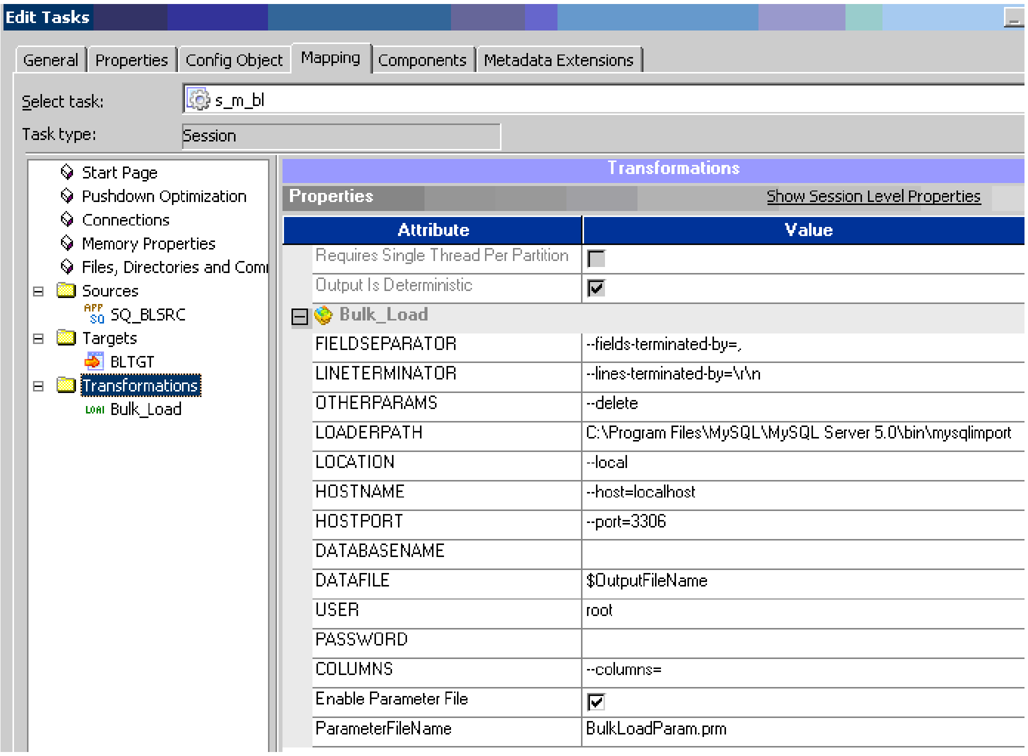
| Extension Name | Datatype | Prec | Value | Reus |
|---|---|---|---|---|
| BULKLOAD_Transform_Domain | | | | |
| Columns | String | 1024 | --columns= | ☑ |
| Data File | String | 1024 | C:\\Informatica\\861\\server\\infa_shared\\TgtFiles\\perftest_10.out | ☑ |
| Database Name | String | 1024 | mysqldb | ☑ |
| Fields Terminated By | String | 1024 | --fields-terminated-by=, | ☑ |
| Host Name | String | 1024 | --host=localhost | ☑ |
| Host Port | String | 1024 | --port=3306 | ☑ |
| Lines Terminated By | String | 1024 | --lines-terminated-by="\r\n" | ☑ |
| Loader Executable | String | 1024 | C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqlimport | ☑ |
| Location | String | 1024 | --local | ☑ |
| Other Params | String | 1024 | --delete | ☑ |
| Password | String | 1024 | --password=asahu | ☑ |
| User | String | 1024 | --user=root | ☑ |

Description:

- In the session task, set the parameter values in the Transformations node and the Targets node of the Mapping tab. The Transformations node and Targets node display attributes defined in the plug-in XML file.

  The value of the Datafile attribute of the transformation must be the same as the value of the Output Filename attribute of the target. In the example, the $OutputFileName variable is used for both attributes in the session. You can use the actual file name instead of the $OutputFileName variable. The file name must match the file name you provide in the plug-in XML and the bulk loader command.

  The following figure shows the parameters for the bulk loader transformation in the session:



Select task: s_m_bl

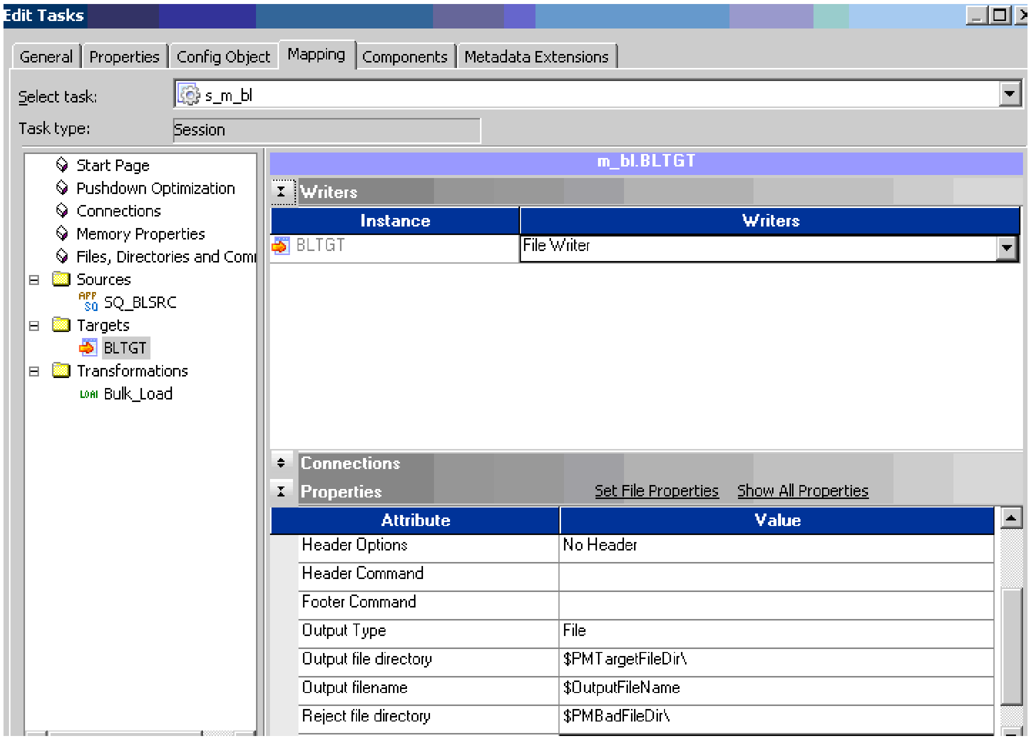
Task type: Session

Transformations

Properties — Show Session Level Properties

| Attribute | Value |
|---|---|
| Requires Single Thread Per Partition | ☐ |
| Output Is Deterministic | ☑ |
| Bulk_Load | |
| FIELDSEPARATOR | --fields-terminated-by=, |
| LINETERMINATOR | --lines-terminated-by=\r\n |
| OTHERPARAMS | --delete |
| LOADERPATH | C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqlimport |
| LOCATION | --local |
| HOSTNAME | --host=localhost |
| HOSTPORT | --port=3306 |
| DATABASENAME | |
| DATAFILE | $OutputFileName |
| USER | root |
| PASSWORD | |
| COLUMNS | --columns= |
| Enable Parameter File | ☑ |
| ParameterFileName | BulkLoadParam.prm |

The following figure shows the parameters for the flat file target in the session:



Select task: s_m_bl

Task type: Session

m_bl.BLTGT

Writers

| Instance | Writers |
|---|---|
| BLTGT | File Writer |

Connections

Properties — Set File Properties — Show All Properties

| Attribute | Value |
|---|---|
| Header Options | No Header |
| Header Command | |
| Footer Command | |
| Output Type | File |
| Output file directory | $PMTargetFileDir\ |
| Output filename | $OutputFileName |
| Reject file directory | $PMBadFileDir\ |

- If you select the Enable Parameter File attribute in the session, you can set the parameter values in a parameter file. The values in the parameter file override the values you set for the transformation in the session.

## Bulk Loader Transformation Parameter File

The example uses a parameter file named BulkLoadParam.prm to specify the parameters for loading data into the database. In the PowerCenter session, you must select the Enable Parameter File attribute and specify the name of this parameter file in the ParameterFileName attribute of the custom transformation.

The parameter file for the bulk loader custom transformation example has the following contents:

```
[Global]
[Service:DI_861]
[mysql.WF:wf_m_mysqlload]
[mysql.WF:wf_m_mysqlload.ST:s_m_mysqlload]
$Param_LOADERPATH=C:\Program Files\MySQL\MySQL Server 5.0\bin\mysqlimport
$Param_LOCATION=--local
$Param_FIELDSEPARATOR=--fields-terminated-by=,
$Param_LINETERMINATOR=--lines-terminated-by="\r\n"
$Param_HOSTNAME=--host=localhost
$Param_HOSTPORT=--port=3306
$Param_DATABASENAME=mysqldb
$Param_USER=--user=root
$Param_PASSWORD=--password=asahu
$Param_DATAFILE=ins_tgt
$Param_OTHERPARAMS=--delete
$Param_COLUMNS=--columns=
;This variable need not be prefixed with the Param_ as designed.
$OutputFileName=ins_tgt
```

# Running the Bulk Loader Example

The bulk loader custom transformation example was compiled on Windows 2003 with Visual Studio 2003 and tested with PowerCenter 8.6.1 and framework version 5.0.x. The example was not tested on UNIX. To build a bulk loader transformation for PowerCenter 9.0, change the product version to 9.0 and the framework version to 6.0.x in the header files, plug-in XML file, and the registry entry.

To run the example on other platforms, recompile the libraries on your platform with your compiler. Based on this example, you can create bulk loader transformations for other databases that have bulk load utilities.

You can run the bulk loader custom transformation example to see how it works. Before you run the bulk loader example, verify that your installation of the MySQL database has the bulk loader utility.

1. In the MySQL database server, create a database named mysqldb for the example. Set up the user account and password and verify the host name and port number for MySQL.
2. Extract the bulk loader example files to a temporary directory.
3. Copy the following client libraries from the <BulkLoaderTempDir>\client\release directory to the `\client\bin` directory of the PowerCenter Client:

   ```
   pmbulkloadvldn.dll
   pmbulkloadtransformres411.dll
   pmbulkloadtransformres409.dll
   pmbulkloadtransform.dll
   ```

4. Copy the following resource files from the <BulkLoaderTempDir>\server\release directory to the \client\bin directory of the PowerCenter Client:

   ```
   pmbulkload_ja.res
   pmbulkload_en.res
   ```

5. Copy the following server libraries from the <BulkLoaderTempDir>\server\release directory to the \server \bin directory in PowerCenter:

   ```
   pmbulkloadvldn.dll
   pmbulkload_ja.res
   pmbulkload_en.res
   pmbulkloadtrans.dll
   ```

6. Copy the parameter file from the root of the bulk loader temporary directory to the \server\bin directory in PowerCenter:

   ```
   BulkLoadParam.prm
   ```

7. Copy the data source file from the root of the bulk loader temporary directory to the `\server \infa_shared\SrcFiles` directory in PowerCenter:

   ```
   ins_src
   ```

8. Run the following SQL script in the MySQL database to create the table into which the bulk loader will load the data:

   ```
   ins_tgt.sql
   ```

   The SQL script is located in the root of the bulk loader temporary directory.

9. Import the mapping example located in the root of the bulk loader temporary directory into PowerCenter:

   ```
   wf_m_mysqlload.XML
   ```

   The XML file includes the workflow, mapping, and other PowerCenter objects used in the example.

10. Configure the example components to match your environment:

| Component | Modification |
|---|---|
| BulkLoadParam.prm | Modify the LOADERPATH parameter to point to the location of the mysqlimport client program. Modify the host name and port, database name, and user name and password to match your MySQL server and database information. |
| pmbulkloadtransform.xml | Modify the LOADERPATH parameter to point to the location of the mysqlimport client program. Modify the host name and port, database name, and user name and password to match your MySQL server and database information. Modify all instances of the attributes in the file. |
| Bulk loader custom transformation | After you import the mapping example into PowerCenter, edit the bulk loader custom transformation. In the Properties tab, set the Runtime Location attribute to point to the location of the PowerCenter libraries. The default location for the PowerCenter libraries is <PowerCenterDir>/server/bin. This is the directory where you copied the server libraries for the example in step 5. |

11. On the Administrator Tool, register the plug-in XML file from the <BulkLoaderTempDir>\repository directory to a repository service:

    ```
    pmbulkloadtransform.xml
    ```

12. Use the following file from the <BulkLoaderTempDir>\client directory to register the client libraries:

    ```
    bulkloadtransform.reg
    ```

13. Run the workflow.

# Troubleshooting the Bulk Loader Example

If the workflow fails to run, use the following guidelines to troubleshoot the problem:

- View the error messages in the session log files. If the session log shows an error in the bulk load command, verify that the parameters you pass to the mysqlimport client program are correct. Verify that the database user name and password and the columns for the table correspond to the database and table you are loading data into.
- Review the plug-in XML and verify that the following attributes are set to the correct values:
  - LOADERPATH. This attribute must point to the location of the bulk loader utility. In the example, the LOADERPATH must point to the location of the mysqlimport command.
  - DATAFILE. This attribute must be set to the name of the table where the bulk loader loads the data. In the example, the default value is set to the $OutputFileName variable. Verify that this variable is defined in the session. Otherwise change the value to the name of the target file name. The value of the DATAFILE attribute must be the same as the target file name without the extension.
  - DATABASENAME. Name of the MySQL database that contains the table where the bulk loader loads the data.
- If you enable parameter files for the session, verify that the values set for the parameters in the parameter file are correct. Verify that the folder, workflow, and session are correct in the file header.
- If the runtime libraries do not load correctly, recompile the example client and server libraries and link to the libraries included in the Informatica Development Platform (IDP). Ensure that the version of the APIs you link to is the same as the version of PowerCenter you run the example on. The API libraries are located in the folder <IDPInstallationDir/ SDK/PowerCenter_Connect_SDK/lib.

# CHAPTER 7

# Design API

This chapter includes the following topics:

## Design API Overview

You can use the Design API to read and create PowerCenter design objects. Use the Design API to build custom integration design interfaces, browse through PowerCenter metadata in the repository, or create PowerCenter objects. You can use the Design API to work with the following objects:

- Sources and targets
- Transformations
- Mappings and mapplets
- Sessions, tasks, and workflows
- Runtime parameter files

This chapter briefly describes and provides sample code to show how to use the Design API to perform the following tasks:

- Connecting to the PowerCenter repository and browsing metadata
- Browsing through metadata in the repository
- Creating PowerCenter objects
- Exporting and importing metadata

## Browsing Metadata in PowerCenter Repository

You can use the Design API to connect to the repository and browse through the content. The SlowChangingDimension application example for the Design API shows how to use a configuration file to store the configuration settings to connect to the repository. Alternatively, you can use the Design API functions to configure the connection to the repository.

The following sample code shows how to connect to the repository and browse through the folders and their contents:

```
public void execute() throws Exception {
        // initialise the repository configurations.
        init();
        RepositoryConnectionManager repmgr = new PmrepRepositoryConnectionManager();
        rep.setRepositoryConnectionManager(repmgr);

        // get the list of folder names which satisfies filter condition
        List<Folder> folders = rep.getFolders(new INameFilter() {
             public boolean accept(String name) {
                 return name.equals("Oracletest");
             }
        });
        //folder count -in this case it is always 1
        int folderSize = folders.size();
        for(int i=0 ; i < folderSize; i++){
            List<Source> listOfSources = ((Folder)folders.get(i)).getSources(); //get
the list of sources
            int listSize = listOfSources.size();
            System.out.println(" ***** List of Sources ******");
            for(int j=0; j < listSize; j++){
               System.out.println(((Source)listOfSources.get(j)).getName());
            }
        }
        for(int i=0 ; i < folderSize; i++){
        List<Target> listOfTargets = ((Folder)folders.get(i)).getTargets(); //get the
list of targets
        int listSize = listOfTargets.size();
        System.out.println(" ***** List of Targets ******");
        for(int j=0; j < listSize; j++){
            System.out.println(((Target)listOfTargets.get(j)).getName());
            }
        }
        for(int i=0 ; i < folderSize; i++){
        List<Mapplet> listOfMapplets = ((Folder)folders.get(i)).getMapplets(); //get the
list of mapplets
        int listSize = listOfMapplets.size();
        System.out.println(" ***** List of Mapplets ******");
        for(int j=0; j < listSize; j++){
            System.out.println(((Mapplet)listOfMapplets.get(j)).getName());
            }
        }
        for(int i=0 ; i < folderSize; i++){
        List<Mapping> listOfMappings = ((Folder)folders.get(i)).getMappings(); //get the
list of mappings
        int listSize = listOfMappings.size();
        System.out.println(" ***** List of Mappings ******");
        for(int j=0; j < listSize; j++){
            System.out.println(((Mapping)listOfMappings.get(j)).getName());
            }
        }
    }
```

# Enabling Kerberos Authentication

You can use the Design API to connect to a repository and perform repository functions in a domain that uses Kerberos authentication. You can enable Kerberos authentication through the `pcconfig.properties` file or when you create a repository object.

## Enabling Kerberos Authentication Through the pcconfig.properties File

You can enable Kerberos authentication through the `pcconfig.properties` file. Set the KERBEROS_ENABLED property in the `pcconfig.properties` file to enable Kerberos authentication.

1. Navigate to the location of the `pcconfig.properties` file. For example, `<PowerCenterClientInstallationDirectory>clients\PowerCenterClient\MappingSDK\samples`.
2. Add or change the KERBEROS_ENABLED property and set its value to true. For example, KERBEROS_ENABLED = true.
3. Save the `pcconfig.properties` file.

## Enabling Kerberos Authentication When You Create a Repository Object

You can set the security domain and Kerberos property when you create a repository object.

The following sample code shows how to set the security domain and Kerberos property:

```
// Sets the security domain.
rep.getRepoConnectionInfo().setSecurityDomain(SecurityDomainName);

// Sets the Kerberos mode to true.
rep.getRepoConnectionInfo().setKerberosMode(true);
```

# Creating Objects

This section describes concepts involved in using the Design API to create and work with objects in PowerCenter.

## Creating a Repository and Folder

The repository object is a container for folders. A folder object contains metadata objects such as source and target, mappings, mapplets, transformations, sessions and workflows. When you create repository and folder objects, they are stored in memory until you save them to the PowerCenter repository.

The following sample code shows how to create repository and folder objects:

```
/**
* Creates a repository
*/
protected void createRepository() {
        rep = new Repository( "repo1", "repo1", "This repository contains API test
samples" );
}
/**
* Creates a folder
*/
protected void createFolder() {
     folder = new Folder( "Folder1", "Folder1", "This is a folder containing java
mapping samples" );
     rep.addFolder( folder );
}
```

## Creating Sources and Targets

PowerCenter mappings contain source and target objects. Source and target objects hold metadata describing tables and columns. Source and target metadata is typically derived from external database system catalogs and other enterprise system catalogs.

You can use the Design API to create source and target objects for the following data sources:

- **Flat file (fixed or delimited).** The Design API supports flat files for source and target objects.
- **Relational databases.** The Design API supports the following types of relational databases for source and target objects:
    - DB2
    - MS SQL Server
    - Sybase
    - Informix
    - Teradata
- **ODBC data sources.** Includes connections to Netezza and Neoview.

The following sample code shows how to create a file source object. This example uses the field object to hold the metadata for each field. A vector of fields contains all the fields for the source. The example also creates a flat file source object.

```
protected Source createOrderDetailSource() {
    List<Field> fields = new ArrayList<Field>();
    Field field1 = new Field("OrderID", "OrderID","", NativeDataTypes.FlatFile.INT,
"10", "0",FieldKeyType.FOREIGN_KEY, FieldType.SOURCE, false);
    fields.add(field1);
    Field field2 = new Field("ProductID", "ProductID","", NativeDataTypes.FlatFile.INT,
"10", "0", FieldKeyType.FOREIGN_KEY, FieldType.SOURCE, false);
    fields.add(field2); Field field3 = new Field("UnitPrice", "UnitPrice","",
NativeDataTypes.FlatFile.NUMBER, "28", "4", FieldKeyType.NOT_A_KEY, FieldType.SOURCE,
false);
    fields.add(field3);
    Field field4 = new Field("Quantity", "Quantity","", NativeDataTypes.FlatFile.INT,
"10", "0", FieldKeyType.NOT_A_KEY, FieldType.SOURCE, false);
    fields.add(field4);
    Field field5 = new Field("Discount", "Discount","", NativeDataTypes.FlatFile.INT,
"10", "0", FieldKeyType.NOT_A_KEY, FieldType.SOURCE, false);
    fields.add(field5);
    Field field6 = new Field("VarcharFld", "VarcharFld","",
NativeDataTypes.FlatFile.STRING, "5", "0", FieldKeyType.NOT_A_KEY, FieldType.SOURCE,
false);
    fields.add(field6);
    Field field7 = new Field("Varchar2Fld", "Varchar2Fld","",
NativeDataTypes.FlatFile.STRING, "5", "0", FieldKeyType.NOT_A_KEY, FieldType.SOURCE,
false);
    fields.add(field7);
    ConnectionInfo info = getFlatFileConnectionInfo();

info.getConnProps().setProperty(ConnectionPropsConstants.SOURCE_FILENAME,"Order_Details.c
sv");
Source ordDetailSource = new Source( "OrderDetail", "OrderDetail", "This is Order Detail
Table", "OrderDetail", info );
ordDetailSource.setFields( fields );
 return ordDetailSource;
}

protected ConnectionInfo getFlatFileConnectionInfo() {
    ConnectionInfo infoProps = new ConnectionInfo( SourceTargetType.Flat_File );
    infoProps.getConnProps().setProperty(ConnectionPropsConstants.FLATFILE_SKIPROWS,"1");

infoProps.getConnProps().setProperty(ConnectionPropsConstants.FLATFILE_DELIMITERS,";");
    infoProps.getConnProps().setProperty(ConnectionPropsConstants.DATETIME_FORMAT,"A 21
yyyy/mm/dd hh24:mi:ss");
```

```
    infoProps.getConnProps().setProperty(ConnectionPropsConstants.FLATFILE_QUOTE_CHARACTER,"D
    OUBLE");
        return infoProps;
    }
```

You can also create target tables with similar code. In addition, you can create target objects dynamically when you create mappings.

# Creating Mappings

Mappings are complex objects representing the data flow between sources and targets and the transformations to move data from sources to targets. The mapping object stores the links from the source objects through one or more transformation objects to the target objects. The links connect the ports from one object to the next object in the data flow.

## Transformation Inputs and Outputs

All transformations have inputs and outputs. Inputs are a set of ports that represent the data going into the transformation. Outputs are the set of ports that represent the data going out of the transformation. A source has output ports and a target has input ports. Links are objects that connect ports from sources to transformations to targets.

## Single-Group and Multi-Group Transformations

Most transformations, sources, and targets are single group. They contain one group of ports. For example, relational sources and transformations that operate on a row of data such as the Expression, Filter, and Aggregator transformations, are single-group. Hierarchical sources and targets are multi-group with a parent group and one or more child groups. Transformations that operate on a hierarchy of data are also multi-group. For example, XML sources and targets and XML transformations are multi-group. The Router and Union transformations are also multi-group since they work with one or more sets of input ports and output ports.

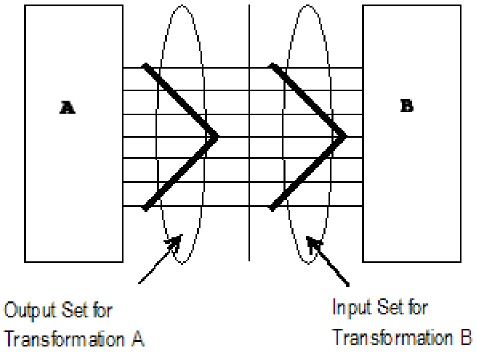
## Data Flow

Data flow linkage in the mapping is done on an exception basis. The Design API allows you to specify the dataflow to and from the transformations ports you want to use. The ports that are not necessary for the data transformation flow automatically. This approach simplifies programmatic specification of the mapping.

The following classes are used in the Design API:

- **Rowset.** A class that contains a collection of field objects that represents input to a transformation or target or output from a transformation or source. The rowset corresponds to a single group of ports in a transformation, source, or target.
- **Input set.** A class that contains a rowset that represents one group of input ports to a transformation. The class also has the corresponding propagation and linking context objects that determine what ports are propagated and how they are linked to a downstream transformation. The input set is used whenever a new transformation is created in the data flow, and defines the input ports to the new transformation. Note that multiple input sets will be needed for transformations and targets that are multi-group.
- **Output set.** This class encapsulates the output of a transformation. It could contain a single rowset or multiple rowsets depending on if it represents a single group or multi-group output. For example, the output set for a Filter transformation contains one rowset, but a Router transformation contains multiple rowsets.

The following diagram shows the input and output sets:


A
B
Output Set for Transformation A
Input Set for Transformation B

## Linking and Propagating Ports in a Mapping

How ports are propagated and linked from one transformation to the next transformation in a mapping is controlled by the port propagation context object. The port propagation context object is used to pass the object information needed for propagating ports. The values of the context object depends on the propagation type. The port propagation context object is used with input sets and define the strategy for propagating ports. The strategy determines which ports from an input set are propagated to a downstream transformation or target.

By default, all ports are propagated from the input set. You can use PortPropagationContextFactory class to define the propagation strategy and control which ports are propagated. You can use one of the following propagation strategies:

- **All.** Propagate all ports. This is the default propagation strategy.
- **Specific list of port names to be included.** Propagate only the ports in the list.
- **Specific list of port names to be excluded.** Do not propagate ports included in the list. Use this strategy if you want to exclude ports from a large list of ports.
- **Keytype of port (PK/FK).** Propagate ports based on a keytype. Use this strategy to propagate key ports to transformations such as a Lookup.
- **Datatype of port.** Propagate ports based on datatype. You can use this strategy to propagate ports with the same datatype to an expression transformation for data conversion purposes.
- **Regular expression pattern.** Propagate ports based on names matching a regex pattern. You can use this strategy to append prefixes or suffixes to target table port names.

The following code example shows how to use the Port Propagation context:

```
// create dsq transformation
OutputSet outputSet = helper.sourceQualifier(itemsSrc);
RowSet dsqRS = (RowSet) outputSet.getRowSets().get(0); PortPropagationContext
dsqRSContext  =   PortPropagationContextFactory.getContextForExcludeColsFromAll(new
String[] { "Manufacturer_Id" });


// create a lookup transformation
outputSet = helper.lookup(dsqRS, manufacturerSrc,"manufacturer_id = in_manufacturer_id",
"Lookup_Manufacturer_Table");
RowSet lookupRS = (RowSet) outputSet.getRowSets().get(0);
PortPropagationContext lkpRSContext =
PortPropagationContextFactory.getContextForIncludeCols(new String[]
{ "Manufacturer_Name" });
List<InputSet> vInputSets = new ArrayList<InputSet>();
vInputSets.add(new InputSet(dsqRS, dsqRSContext)); // remove //Manufacturer_id
```

```
// propagate  only  Manufacturer_Name
vInputSets.add(new InputSet(lookupRS, lkpRSContext));
```

The following code example shows how to use the exclude rule to propagate ports:

```
PortPropagationContext exclOrderCost
=PortPropagationContextFactory .getContextForExcludeColsFromAll(new String[]
{ "OrderCost" });
// exclude
```

- **Port link context object**. Context object for passing the object information needed for linking ports. The values of the context object depend on the link type. Port link context indicates which strategy is used to connect input ports to ports in the downstream transformation.

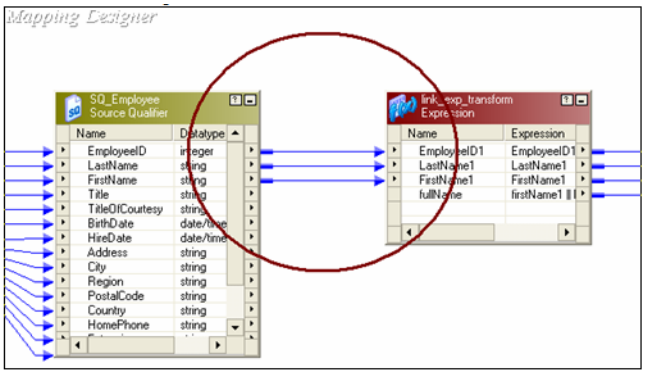
You can use one of the following linking strategies:

- **By Name.** Link ports based on matching names. Use this strategy when port names between the from and to transformations are the same. This is the default linking strategy.
- **By Position.** Link ports based on position. The first input port connects to the first port in the transformation, the second input port connects to the second port in the transformation. Use this strategy to link ports by matching their positions.
- **By Hashmap.** Link ports based on a map that lists the from and to ports. Use this strategy to link ports based on a pre-defined list of matched names. Use this strategy to connect ports to targets where the target ports are different from the incoming port names.

The following sample code show how to link ports by position. The ports are linked from the Source Qualifier transformation to the Expression transform, in the order of the ports in the Source Qualifier.

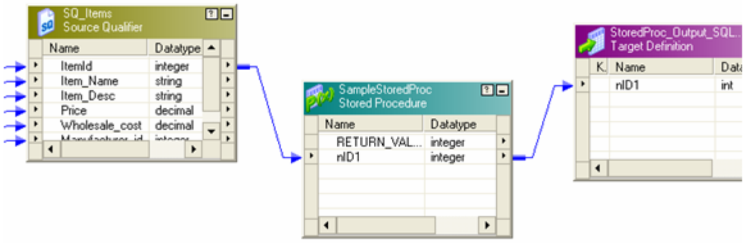
```
public List<Field> getLinkFields() {
    List<Field> fields = new ArrayList<Field>();
    Field field1 = new Field( "EmployeeID1", "EmployeeID1", "",
        TransformationDataTypes.INTEGER, "10", "0",
FieldKeyType.PRIMARY_KEY,FieldType.TRANSFORM, true );
    fields.add( field1 );
    Field field2 = new Field( "LastName1", "LastName1", "",
        TransformationDataTypes.STRING, "20", "0",
FieldKeyType.NOT_A_KEY,FieldType.TRANSFORM, false );
    fields.add( field2 );
    Field field3 = new Field( "FirstName1", "FirstName1", "",
           TransformationDataTypes.STRING, "10", "0", FieldKeyType.NOT_A_KEY ,
FieldType.TRANSFORM, false );
    fields.add( field3 );
    return fields;
}

// create link fields
List<Field> linkFields = getLinkFields();
// create the link
PortLinkContext portLinkContext =
PortLinkContextFactory.getPortLinkContextByPosition( linkFields );
InputSet linkInputSet = new InputSet( dsqRS, portLinkContext );
// create an expression Transformation
// the fields LastName and FirstName are concataneted to produce a new field fullName
String expr = "string(80, 0) fullName= firstName1 || lastName1";
TransformField outField = new TransformField( expr );
RowSet expRS = (RowSet) helper.expression( linkInputSet, outField,
"link_exp_transform").getRowSets().get( 0 );

// write to target
mapping.writeTarget( expRS, outputTarget );
```

The following mapping shows the ports linked by position:



The following sample code and mapping shows how to use a hashmap to link ports:

```
// create a stored procedure transformation
List<TransformField> vTransformFields = new ArrayList<TransformField>();
Field field1 = new Field( "RetValue", "RetValue", "This is return
value",TransformationDataTypes.INTEGER, "10", "0",
FieldKeyType.NOT_A_KEY,FieldType.TRANSFORM, false );
TransformField tField1 = new TransformField( field1, PortType.RETURN_OUTPUT);
vTransformFields.add( tField1 );
Field field2 = new Field( "nID1", "nID1", "This is the ID field",
TransformationDataTypes.INTEGER, "10", "0", FieldKeyType.NOT_A_KEY,
         FieldType.TRANSFORM, false );
TransformField tField2 = new TransformField( field2, PortType.INPUT);
// vTransformFields.add( tField2 );
Field field3 = new Field( "outVar", "outVar", "This is the Output
field",TransformationDataTypes.STRING, "20", "0",
FieldKeyType.NOT_A_KEY,FieldType.TRANSFORM, false );
TransformField tField3 = new TransformField( field3, PortType.INPUT_OUTPUT );
vTransformFields.add( tField3 );

java.util.Hashtable link = new java.util.Hashtable();
link.put( dsqRS.getField( "ItemId" ), field2 );

PortLinkContext linkContext = PortLinkContextFactory.getPortLinkContextByMap( link );

RowSet storedProcRS = (RowSet) helper.storedProc( new InputSet( dsqRS,
linkContext ),vTransformFields, "SampleStoredProc", "Sample Stored Procedure
Transformation" ).getRowSets().get( 0 );
```



## Related Topics:

- "Sample Patterns for Regular Expressions for Port Propagation" on page 177

# Creating Transformations

The Transformation helper class simplifies the process of creating transformations in a mapping object.

You can use the Design API to create the following types of transformations:

- Aggregator
- Application Source Qualifier
- Custom
- Data Masking
- Expression
- External Procedure
- Filter
- HTTP
- Input
- Java
- Joiner
- Lookup
- Mapplet
- Normalizer
- Rank
- Router
- Sequence Generator
- Sorter
- Source Qualifier
- SQL
- Stored Procedure
- Transaction Control
- Union
- Update Strategy
- XML Generator
- XML Parser
- XML Source Qualifier

The following sample code shows how use the transformation helper class to create a Lookup transformation. Note that only the Manufacturer_id is linked to the Lookup transformation, and the Manufacturer_Name is propagated to the target from the lookup.

```
// create dsq transformation
OutputSet outputSet = helper.sourceQualifier(itemsSrc);
RowSet dsqRS = (RowSet) outputSet.getRowSets().get(0); PortPropagationContext
dsqRSContext  =   PortPropagationContextFactory.getContextForExcludeColsFromAll(new
String[] { "Manufacturer_Id" });

// create a lookup transformation
outputSet = helper.lookup(dsqRS, manufacturerSrc,"manufacturer_id = in_manufacturer_id",
"Lookup_Manufacturer_Table");
RowSet lookupRS = (RowSet) outputSet.getRowSets().get(0);
PortPropagationContext lkpRSContext =
PortPropagationContextFactory.getContextForIncludeCols(new String[]
{ "Manufacturer_Name" });
List<InputSet> vInputSets = new ArrayList<InputSet>();
vInputSets.add(new InputSet(dsqRS, dsqRSContext)); // remove //Manufacturer_id
```

```
// propagate  only  Manufacturer_Name
vInputSets.add(new InputSet(lookupRS, lkpRSContext));

// write to target
mapping.writeTarget(vInputSets, outputTarget);
```

## Creating Sessions and Workflows

The Session object defines the run-time attributes of PowerCenter mappings. The Workflow object defines the orchestration of one or more PowerCenter sessions and other workflow tasks, including commands.

Use the Design API to create a Session object from the Mapping object. You can set the attributes of the Session object, including connectivity to the source and target. You can create a Workflow object with one or more tasks objects.

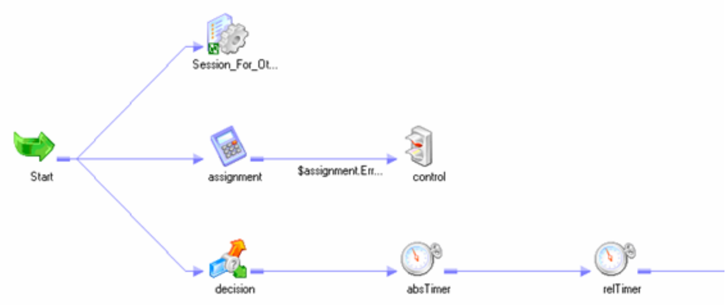
The following sample code shows how to create a workflow with a single session:

```
    /**
    * Create session
    */
protected void createSession() throws Exception {
    session = new Session( "Session_For_Filter", "Session_For_Filter",
                "This is session for filter" );
    session.setMapping( this.mapping ); }
   }

    /**
    * Create workflow
    */
protected void createWorkflow() throws Exception {
      workflow = new Workflow( "Workflow_for_filter", "Workflow_for_filter","This
workflow for filter" );
        workflow.addSession( session );
        folder.addWorkFlow( workflow );
    }
```

The following sample code shows how to create a workflow with multiple tasks:

```
private void createTasks() {
    assignment = new Assignment("assignment","assignment","This is a test assignment");
    assignment.addAssignmentExpression("$$var1", "1");
    assignment.addAssignmentExpression("$$var2", "$$var1 + 5");
    assignment.addAssignmentExpression("$$var1", "$$var2 -10");
        control = new Control("control","control","This is a test control");
    control.setControlOption(Control.ControlOption.ABORT_PARENT);
    assignment.connectToTask(control,"$assignment.ErrorCode != 0");
    decision = new Decision("decision","decision","This is a test decision");
    decision.setDecisionExpression("1 + 2");
       absTimer = new Timer("absTimer","absTimer","absolute timer",
TimerType.createAbsoluteTimer(new Date()));
     decision.connectToTask(absTimer);
     relTimer = new Timer("relTimer","relTimer","relative timer",
TimerType.createRelativeToPreviousTaskTimer(3, 5,
10,TimerType.TIMER_TYPE_START_RELATIVE_TO_TOPLEVEL_WORKFLOW));
       absTimer.connectToTask(relTimer);varTimer = new
Timer("varTimer","varTimer","variable timer", TimerType.createVariableTimer("$
$timerVar"));
     relTimer.connectToTask(varTimer);
     command = new Command("command","command","This is a test command");
     command.addCommand("command1", "ls"); command.addCommand("command2", "ls -lrt");
command.addCommand("command1", "df -k .");
     varTimer.connectToTask(command);
     email = new EMail("myEmail","myEmail","my email task");
     email.setEmailUsername("guest@informatica.com");
     email.setEmailSubject("Welcome to Informatica");
     email.setEmailText("This is a test mail");
     command.connectToTask(email);
}
protected void createWorkflow() throws Exception {
    workflow = new Workflow("Workflow_for_OtherTasks","Workflow_for_OtherTasks", "This 
workflow for other types of tasks");
    WorkflowVariable wfVar1 = new WorkflowVariable("$
$var1",WorkflowVariableDataTypes.INTEGER,"1","var1 ");
    WorkflowVariable wfVar2 = new WorkflowVariable("$
$var2",WorkflowVariableDataTypes.INTEGER,"1","var2 ");
    WorkflowVariable wfVar3 = new WorkflowVariable("$
$timerVar",PowerMartDataTypeConstants.TIMESTAMP,"","timerVariable ");
workflow.addWorkflowVariable(wfVar1);
    workflow.addWorkflowVariable(wfVar2);
    workflow.addWorkflowVariable(wfVar3);
    createTasks();
    workflow.addTask(assignment);
    workflow.addTask(control);
    workflow.addTask(decision);
    workflow.addTask(command);
    workflow.addTask(absTimer);
    workflow.addTask(relTimer);
    workflow.addTask(varTimer);
    workflow.addTask(email);
    workflow.addSession(session);
    folder.addWorkFlow(workflow);
}
```

The following workflow shows the tasks created by the previous code:



## Creating Connection Objects

Connection objects define connections to source and target data. The following sample code shows how to create a Connection object to a relational database:

```
Properties prop = 
ConnectionAttributes.getDefaultRelationalProperties(ConnectionAttributes.DB_CONN_TYPE_ORA
CLE);
prop.list(System.out);
prop.setProperty(ConnectionAttributes.CONN_ATTR_CONNECTION_NAME, "connection_name");
prop.setProperty(ConnectionAttributes.CONN_ATTR_USER_NAME, "connection_username");
prop.setProperty(ConnectionAttributes.CONN_ATTR_CONNECT_ENV_SQL,"");
prop.setProperty(ConnectionAttributes.CONN_ATTR_CODE_PAGE,"");
prop.setProperty(ConnectionAttributes.CONN_ATTR_CONNECT_STRING,"");
connectionObject connObj = new 
ConnectionObject("myConn",ConnectionAttributes.CONN_TYPE_RELATIONAL);
        connObj.setConnectionObjectAttr(prop);

try
{
myRepo.updateConnection(connObj);
} catch (RepoConnectionObjectOperationException e)
{
    e.printStackTrace();
}
```

# Exporting and Importing Metadata in the Repository

You can save metadata created with the Design API into an XML file that conforms to the powrmart.dtd. You can then use the PowerCenter Client tools or the pmrep command to import the metadata into the PowerCenter repository.

You can also use the Design API to export and import metadata in the PowerCenter repository. Use the pcconfig.properties file to specify the repository connection information and the import and export options.

The following example shows the contents of a sample pcconfig.properties file that includes attributes for connecting to the domain and repository.

```
PC_CLIENT_INSTALL_PATH=client path ;the path where PowerCenter Client is installed
PC_SERVER_INSTALL_PATH=server path ;the path where the PowerCenter Server is installed
TARGET_FOLDER_NAME=demomapp ;the folder name
TARGET_REPO_NAME=repo123 ;the repository containing the folder
REPO_SERVER_HOST=S158244 ;the host machine name on the network
REPO_SERVER_PORT=5001 ;the repository server port
ADMIN_USERNAME=Administrator ;admin username
ADMIN_PASSWORD=Administrator ;admin password
SERVER_PORT=4001 ;the server port on which the server is running. This is unused as of
now.
DATABASETYPE=Oracle ;the database type
```

The following sample code shows how to use the Design API to export mapping metadata from the repository:

```
  public void generateOutput() throws Exception {
        MapFwkOutputContext outputContext = new
MapFwkOutputContext(MapFwkOutputContext.OUTPUT_FORMAT_XML,MapFwkOutputContext.OUTPUT_TARG
ET_FILE,mapFileName);

        try {
            intializeLocalProps();
        }
        catch (IOException ioExcp) {
            System.err.println( "Error reading pcconfig.properties file." );
            System.err.println( "The properties file should be in directory where
Mapping Framework is installed.");
            System.exit( 0 );
        }
        boolean doImport = false;
        if (runMode == 1) doImport = true;
        rep.save(outputContext, doImport);
        System.out.println( "Mapping generated in " + mapFileName );
    }
```

CHAPTER 8

# Design API Example: Slowly Changing Dimension

This chapter includes the following topics:

## Design API Example Overview

This chapter discusses a sample application that shows how to use the Design API to create and run a workflow without using the PowerCenter Client tools. The sample application named SlowChangingDimension is a stand-alone application written in Java. It illustrates how you can automate creating data integration mappings and workflows and running a workflow to update dimension tables in a data warehouse.

In a data warehouse that uses a star schema, fact tables, such as a customer order or product shipment table, quickly grow in volume. Dimension tables, such as a customer or product table, are comparatively static and do not change very often. The example presented in this chapter uses a workflow to automate the tasks for maintaining the slowly changing dimension (SCD) tables.

The SlowChangingDimensions application is written in Groovy, a high level language for the Java platform. It calls the methods of the Design API to connect to a database table that you specify and extract metadata about table to create the sources and target for the application. It uses the Design API to generate the mapping logic to capture changes to the table and create the session and workflow to run the mapping.

The compiled classes run on JDK 1.8, which is installed with PowerCenter. You can run the application on the command line and use a configuration file to set parameters.

## Overview of the Process Flow

The main method of the sample application gives an overview of the process flow. The application first calls a method to initialize the application and connection properties. It gets the values of the properties from a configuration file or command line parameters. Then it creates a log to trace the process flow.

The following sample code shows how the application is initialized with default values:

```
// Make helper object to call the methods like initLogger
SlowChangingDimensions sm = new SlowChangingDimensions()
// Read properties from file or use at least defaults
sm.initializeJmfdemoProps()
// Some auxiliary variables
def sourceConnection = props.defaultSourceConnection;
def targetConnection = props.defaultTargetConnection
def tableName = props.defaultTableName

// Initialize the logger to log information to console and file
sm.initLogger()
log.log(Level.FINE,"props="+props)
// Use the Groovy jarkarta command line interpreter // needs common.cli.jar !
def cli = new CliBuilder()
cli.h(longOpt: 'help','usage information (this)', required: false)
cli.c(longOpt: 'getconnections','get Connections from PowerCenter repository
(optional)', required: false)
cli.l(longOpt: 'listconnections','List connections from PowerCenter repository
(optional)', required: false)
cli.s(longOpt: 'sourceConnection', "sourceconnectstring : specify Source connection
(optional, default=${sourceConnection})",args:1,required:false)
cli.t(longOpt: 'targetConnection', "targetconnectstring :specify Target connection
(optional, default=${targetConnection})",args:1,required:false)
cli.n(longOpt: 'tablename', "tableName : specify table Name (optional, default=$
{tableName})",args:1,required:false)

// parse the command line
def options = cli.parse(args)
options.each{log.log(Level.FINE,"option="+it)}

if (options.h) { cli.usage();return}
if (options.s) sourceConnection = options.s
if (options.t) targetConnection = options.t
if (options.n) tableName = options.n

/* create a slow dimension synchronization mapping for all listed tables */
log.log(Level.FINE,"table "+tableName)
```

The application creates a DbTable object for the source. It uses the metadata interface of the JDBC driver to retrieve information about the source, including column and primary and foreign key definitions. Based on this information, the application calls Design API methods to generate the PowerCenter objects.

The following sample code shows how metadata is retrieved from the database:

```
// Create a DbTable Objects for the given table
DbTable tab= new DbTable(props.userName, tableName,

          props.url.split(":")[1].toUpperCase());

// Retrieve the column metadata via JDBC from Database
tab.readColumns(Sql.newInstance(props.url, props.userName.toUpperCase(),

          props.Password, props.driverName);

// check if table has at least one primary key
if (tab.pkColNameList.size <= 0) {
// exit with error message
def mesg = "Only tables with at least one primary key column are supported"
system.err.println mesg
log.log(Level.SEVERE, mesg)
return
}
```

The JMFSlowCD class encapsulates the main method to create the object hierarchy. The main method creates the repository and folder objects and uses *pmrep* to get information about the repository. It then creates the mapping, session, and workflow objects needed for the application.

To override the default settings for the JMFSlowCD class, configure the properties in the pmserver.properties file. The pmserver.properties file must be located in the same directory as the mapfwk.jar file.

The following sample code shows the sequence of calls in the main method:

```
// Now use the JMF to create the needed metadata
JMFSlowCD jmfFlow = new JMFSlowCD(tab, sourceConnection, targetConnection)

jmfFlow.identity { // use this object for the included method calls
   // create JMF Repository object
   createRepository()
   // create JMF folder object within the repository object
   createFolder()

   // read properties file for repository connection (to use pmrep for read & write)
   initializePmrepProps()

   // check if needed and stop if failed
   if (options.c)
getConnections()
   else
getConnectionsFromFile()

   if (options.l) { printAllConnections(); }

   //check existing connections and set the connection type
   // otherwise exit
   if(!checkConnections()) { return }
   setCloneSuffix(props.cloneSuffix)  // set table suffix for copy

   // create a mapping to transfer the tables from source to target
   // (Connection in source and target properties)
   createMapping()
  // create the session object
   createSession()
   // set workflows Intergration Service
   setPcisServerName(props.pcisServerName)
   // create a workflow
   createWorkflow()
   // now transfer all objects to repository
   generateOutput()
 }
```

The Design API allows access to the repository with the *pmrep* class, a wrapper for the *pmrep* command line utility. In this example, the application uses *pmrep* to access the PowerCenter connections in the repository and load and validate the mapping, session, and workflow.

The createMapping() method creates a mapping with all dependent objects such as sources and target. The createSession() method adds a session task to the workflow. The generateOutput() method enforces the Design API XMLWriter to write the PowerCenter objects to a PowerCenter export file.

## Using Outer Joins to Load the Dimension Table

To efficiently load the slowly changing dimension, the application reads the original source table and the slowly changing dimension copy of the table with a filter valid_To = 31.12.9999 in primary key order. Then the application uses the Join transformation to perform a full outer join between the original source table and the slowly changing dimension copy of the table.

**Note:** In this example, the slowly changing dimension copy of the source table is equivalent to the target table.

The outer join must handle the following situations with the key values in the data:

1. All primary key values from source table are set and key values from target table are not set.

   This indicates a new record, which has to be inserted into the table with valid_From = SESSIONSTARTTIME and valid_To = 31.12.9999.

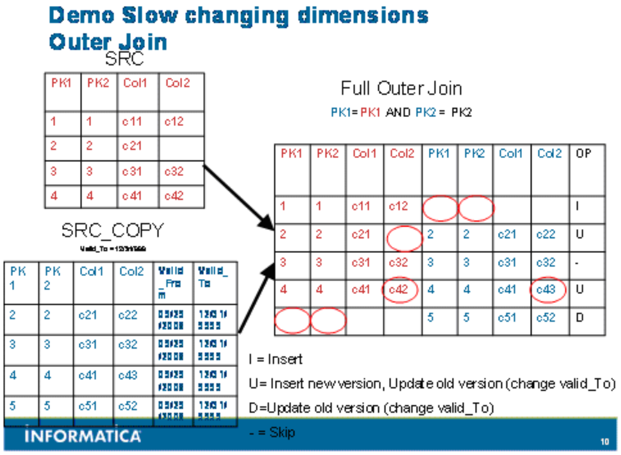
2. All primary key values from the source are not set but key values from the target table are set.

   This indicates that the record was deleted in the source table and has to be invalidated in the target table by updating valid_To = SESSIONSTARTSTIME – 1 ns.

3. If both primary key values are set, the non primary key columns have to be compared.

   If the primary keys are the same, there is no change in the data. If at least one new column value has changed, the existing target record has to be updated and the source record has to be inserted.

The following figure shows the results of the comparison:



**Demo Slow changing dimensions**
**Outer Join**

SRC

| PK1 | PK2 | Col1 | Col2 |
|---|---|---|---|
| 1 | 1 | c11 | c12 |
| 2 | 2 | c21 | |
| 3 | 3 | c31 | c32 |
| 4 | 4 | c41 | c42 |

SRC_COPY

Valid_To =12/31/9999

| PK 1 | PK 2 | Col1 | Col2 | Valid _From | Valid_ To |
|---|---|---|---|---|---|
| 2 | 2 | c21 | c22 | 05/25 /2008 | 12/31/ 9999 |
| 3 | 3 | c31 | c32 | 05/25 /2008 | 12/31/ 9999 |
| 4 | 4 | c41 | c43 | 05/25 /2008 | 12/31/ 9999 |
| 5 | 5 | c51 | c52 | 05/25 /2008 | 12/31/ 9999 |

Full Outer Join

PK1= PK1 AND PK2 = PK2

| PK1 | PK2 | Col1 | Col2 | PK1 | PK2 | Col1 | Col2 | OP |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | c11 | c12 | | | | | I |
| 2 | 2 | c21 | | 2 | 2 | c21 | c22 | U |
| 3 | 3 | c31 | c32 | 3 | 3 | c31 | c32 | - |
| 4 | 4 | c41 | c42 | 4 | 4 | c41 | c43 | U |
| | | | | 5 | 5 | c51 | c52 | D |

I = Insert

U= Insert new version, Update old version (change valid_To)

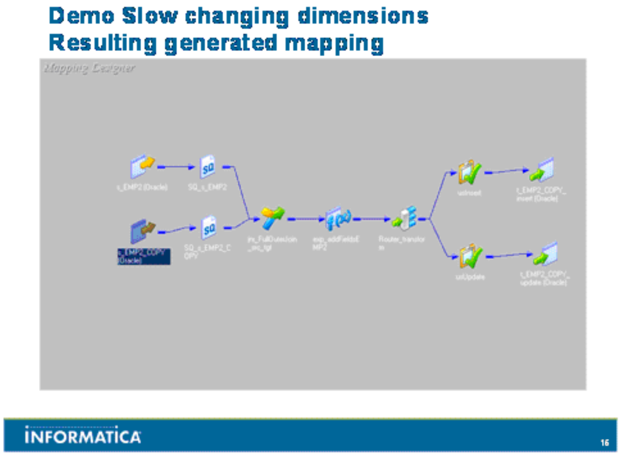
D=Update old version (change valid_To)

- = Skip

The OP column shows the resulting logical operation insert, update, and delete. The insert is a normal insert in the database. The update is split into an update of the former record in the target table with changed valid_To value and an insert of the new version. A delete is an update with an adjusted valid_To value. Depending on the frequency of changes and update runs, the majority of operations read both tables and perform few operations.

## Mapping Result

The following figure shows the mapping for the process:



Demo Slow changing dimensions
Resulting generated mapping

For efficiency, the data must be sorted in primary key order. The Source Qualifier transformation must have the number of sorted ports set to the number of primary key columns and all primary key columns must be sorted first. Also, the source copy Source Qualifier transformation must have the following source filter:

```
valid_to >= ’31.12.9999’
```

In the mapping, the Joiner transformation has a join condition for all primary key values depending on their numbers. The Sorted input property is switched on to allow fast processing.

The Expression transformation adds timestamp values for sessionstarttime and the timestamp before sessionstarttime (usually sessionstarttime – 1 ns). It also adds the name of the workflow that ran the mapping. The mapping name is stored in a mapping variable.

A Router transformation identifies the different cases for update or insert. The router conditions include a comparison of fields that must work with null values in the tables. The update strategy sets the insert or the update strategy for the resulting data flows.

# Design API Methods

## Retrieving Metadata

The constructor for DbTable tab gets the database user name, table name, and database type from the JDBC URL. The readColumns method uses the JDBC URL, user name, password, and driver name to get a JDBC connection type. The information is dynamically loaded and used at run-time.

The method readColumns calls the getMetaData() method to create a DatabaseMetaData instance that retrieves all the required handles for metadata retrieval. The getMetaData() method uses the user name in the JDBC connection to retrieve all column information for a given table in a schema. Similarly, the getPrimaryKeys() method retrieves the primary keys and getImportedKeys() retrieves the foreign keys for the table.

The following sample code shows how metadata is retrieved from the repository:

```
void readColumns(Sql myConnection) {
   // Get Metadata for the given Sql object
   DatabaseMetaData  dbmd = myConnection.connection.getMetaData();

   // Search for column information for this table in schema of given user
   ResultSet rst = dbmd.getColumns("",
this.schemaName.toUpperCase(),this.tableName.toUpperCase(),"%");
   // Search for PK information of table
   ResultSet rstpk = dbmd.getPrimaryKeys("",
this.schemaName.toUpperCase(),
this.tableName.toUpperCase());
   // Search for FK information of table
   ResultSet rstfk = dbmd.getImportedKeys("",
this.schemaName.toUpperCase(),
this.tableName.toUpperCase());
```

The getPrimaryKeys() and getImportedKeys()methods return Java ResultSets. The iteration loop through the ResultSet creates the DbColumn objects to store the information and add the corresponding attribute values. The objects are stored in a HashArray object instance of mapColumns and a List object instance for the column names.

The following sample code shows the iteration loop to create the DbColumn objects:

```
// generate DbColumn object for each column
while(rst.next())
{
String colname = rst.getString("COLUMN_NAME")
DbColumn col = new DbColumn(
    this.schemaName.toUpperCase(),
    this.tableName.toUpperCase(),
    colname,
    rst.getInt("ORDINAL_POSITION")
    )
col.typeName = rst.getString("TYPE_NAME")
col.columnSize = rst.getInt("COLUMN_SIZE")
col.decimalDigits = rst.getInt("DECIMAL_DIGITS")
col.remarks = rst.getString("REMARKS")
col.isNullable = rst.getString("IS_NULLABLE") == "YES" ? true : false
col.isPrimaryKey = false
mapColumns[colname] = col
colNameList << colname
}
```

The Source Qualifier reads the data in primary key order. This requires that the primary key fields are defined as the first ports, even if they are defined in the database in different order. The number of sorted ports must be set to the number of primary key columns. To ensure the order of the ports, an auxiliary list object pkColNameList stores the primary key column names in the right order. Fields that are not primary keys, which are also called payload columns, are stored in a payloadColNameList object.

The following sample code shows how to create the list of primary key and payload columns:

```
/**
 * For each PK component in PK result set pass over the found columns and set PK
attribute if needed
 */
while(rstpk.next())
{
    // get column name of PK
    String name=rstpk.getString("COLUMN_NAME")
    // remember pk in seperate array
    pkColNameList << name
    // set the attributes for the column as neeeded for Primary Keys
    mapColumns[name].identity{
it.isPrimaryKey = true
it.isNullable = false
    }
}
// make an own list of all non pk columns in order of database
// so pkColNameList and payloadColNameList are comple list of col names
colNameList.each {
    def isNotPrimaryKey = ! (mapColumns[it].isPrimaryKey)
    if (isNotPrimaryKey)
    {
        payloadColNameList << it
    }
}
```

Although the identification of the foreign key attributes of the tables is not required, it is added for completeness. Foreign key relationships are typically used only in cases where multiple source tables from the source database are joined.

The following sample code shows how foreign key columns can be identified:

```
/**
* For each PK component in PK resultset pass over the found columns and set PK
* attribute if needed
 */
//Foreign Keys from EMP2
//==========================================================
//PKTABLE_CAT|PKTABLE_SCHEM|PKTABLE_NAME|PKCOLUMN_NAME|FKTABLE_CAT|FKTABLE_SCHEM|
FKTABLE_NAME|FKCOLUMN_NAME|KEY_SEQ|UPDATE_RULE|DELETE_RULE|FK_NAME|PK_NAME|DEFERRABILITY|
//----------------------------------------------------------
//null|SCOTT|DEPT|DEPTNO|null|SCOTT|EMP2|DEPTNO|1|null|1|FK_DEPTNO2|PK_DEPT|7|
while(rstfk.next())
{
   // get column name of PK
   String name=rstfk.getString("FKCOLUMN_NAME")
   // Search over all columns, compare name and set attributes if column was found
   mapColumns[name].identity{
    it.refTable = rstfk.getString("PKTABLE_NAME")
    it.refField = rstfk.getString("PKCOLUMN_NAME")
    it.refSchema = rstfk.getString("PKTABLE_SCHEM")
    it.remarks = "FK "+rstfk.getString("FK_NAME")+" -> PK " +rstfk.getString("PK_NAME")
    }

}
```

# Calling the Design API Methods

The object instance JMFSlowCD encapsulates the calls to the Design API. The parameters for the constructor are the DbTable object with details of the table metadata and the source and target connection names. Optionally, the source and target connection types can be used. The default is the ODBC connection. The connection types are overwritten with those from the PowerCenter connection details retrieved from the repository.

## Creating Repository and Folder Containers

The Design API requires two container object instances, a repository object and a folder object. The repository object contains at least one folder object. During the import process, the objects in the container are transferred to the specified folder in the repository. Both repository and folder are specified in pcconfig.properties file.

The following sample code shows how to create the repository and folder instance:

```
rep = new Repository( "PCquick",
   "Powercenter repository for PCquick",
   "This repository is for the PCquick application" );
folder = new Folder( "PCQ",
   "PCquick folder",
   "This is a folder containing the objects from PCQ application" );
rep.addFolder( folder )
```

## Creating Mappings

This method creates the majority of the Design API objects. It creates a mapping object instance, which is added to the folder object instance. The mapping name is also used as a file name when the mapping is exported into a PowerCenter export file.

The following sample code shows how a mapping instance is created in a folder:

```
String name = "m_${dbTable.tableName}_SCD"

/* Create an empty mapping object */
mapping = new Mapping(name, name,
"This mapping synchronize SCD of the table "+dbTable.tableName)
// Set the filename for the deploy XML file for the mapping
setMapFileName(mapping);
// add the Mapping to the (existing) folder
folder.addMapping(mapping)
```

## Creating Sources

After the mapping instance is created, the application creates the source and target object instances. It uses the createSource and createSourceForTarget methods to create the source and target with the table metadata as parameters in the DbTable object.

The createSourceForTarget method creates a second source instance appropriate for the target table setting. The target connection types and names have to be specified, including the columns containing the valid_From and valid_To timestamps and runtime information.

The createSource and createSourceForTarget methods return instances of a source object, which are added to the folder. This step is not required. By default, the XML writer object in the Design API creates a copy of the sources in a mapping in the folder.

The following sample code shows how instances of the source object are created:

```
// Now add the source to read from to the folder. The fields are derived from the
dbTable object
Source source = createSource(dbTable)
folder.addSource(source)
// From the create source derive a variant which have additional adminstrative fields
incl valid_From/To
Source source_tgt = createSourceForTarget(source)
folder.addSource(source_tgt)
```

The createSource() and createSourceForTarget method are similar. They differ in the fields that they can contain. The createSource() method creates an empty Vector, which is filled with Field object instances, one per column. The primary key columns are added first and then the payload columns. For each column, the attributes are set to indicate primary or foreign keys. The Design API expects the key columns to be of type string, not integer.

The following sample code shows how a vector of source fields is created:

```
Vector fields = new Vector();
// Iterate over all columns of given table,
// starting with pk, then with payload columns
[ tab.pkColNameList, tab.payloadColNameList].flatten().each {
    log.log(Level.FINE,"Process column "+it)

def col = tab.mapColumns[it]
// Create a  field for this column
Field f = new Field(
it,          //String name
it,          //String busName
col.remarks==null? "": col.remarks,          //String description
col.typeName.toLowerCase(),                  //String dataType
col.columnSize,                 //String! prec, on certain field its overrules in class
col.decimalDigits,              //String! scale
col.isPrimaryKey?
```

```
    FieldConstants.PRIMARY_KEY:
    FieldConstants.NOT_A_KEY ,           /int keyType
Field.FIELDTYPE_SOURCE,           //int fieldType
!col.isNullable          //boolean notNull
   )
   // TODO: Check for inconsistencies: refTable set, refField not etc
   // set ref Table appropriately if a table is referenced (i.e. FK)
   if (col.refTable != null)  f.setReferencedSourceName(col.refTable)
   // if we have a reference field then set it
   if (col.refField != null)  {
f.setReferencedFieldName(col.refField)
// if also is a primary field then mark it as combined PK/FK
if (col.isPrimaryKey) {
    f.setKeyType(FieldConstants.PRIMARY_FOREIGN_KEY)
// otherwise we have a simple FOREIGN_KEY field
} else {
    f.setKeyType(FieldConstants.FOREIGN_KEY)
}
}
    // add the field to the vector
    fields.add(f)
} // each
```

To generate a source based on a Vector object, create a new source with a name derived from the table name and the instance name. Prefix the name with *s_* to avoid field name collisions with a target instance with the same table name. Then call the setFields() method to assign the fields vector and create the columns.

Assign a new ConnectionInfo object instance to the source. The ConnectionInfo object instance contains an instance of the class ConnectionProperties. Initialize the ConnectionProperties class with the source connection type properties. Additionally, the property element ConnectionPropsConstants.CONNECTIONNAME must be set to the value of the sourceConnectionName string. These settings must be configured in the source object instance.

The following sample code shows how the source object is generated based on a Vector:

```
// Create source object with type sourceConnectionType
// using the name but prefix instance name with s_
Source source = new Source (name, name, name,
"s_"+name, new ConnectionInfo(sourceConnectionType)) ;
// assign the field set to the source (add columns)
source.setFields( fields )
// create a connection info for the given type
ConnectionInfo connectionInfo = new ConnectionInfo(sourceConnectionType)
// get the properties location for this connection info
ConnectionProperties connectionProperties = connectionInfo.getConnProps()

// Assign the default values for sourceConnectionType to it
connectionProperties =
     ConnectionAttributes.getDefaultRelationalProperties(sourceConnectionType);
// Assign the target connection to it
connectionProperties.setProperty("Connection Name", sourceConnectionName)
// assign the connection information to the created source
source.setConnInfo(connectionInfo)
source.getConnInfo().getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME,s
ourceConnectionName)
```

## Creating Targets

Since the mapping writes to two instances of the target table in two streams, the application must create a target instance for inserts and another target instance for updates. The two instances of the target object have different names but use the same tables. The simplest way to create the targets is to create two empty target objects, assign the properties for the relational target type and connection name, and then fill them with the same fields used by the source. This ensures that all columns are included and both source and target object represent the same metadata. PowerCenter ensures that the internal datatypes are correctly mapped into the appropriate source and target types.

Assign both targets to the mapping, not the folder. By default, the transformation helper class method assigns the targets to the folder. If you assign the target to the folder, duplicate entries can occur.

The following sample code shows how the targets are created and assigned to the mapping:

```
/**
 * Create relational target
 */
Target createRelationalTarget( int type, String name, String prefix = "t_" ) {
    Target target = new Target(name,
name, name,
prefix+name, new ConnectionInfo( type ) ) ;

    target.getProps().
setProperty(ConnectionPropsConstants.CONNECTIONNAME,targetConnectionName);
    return target;
}

// create two empty target instances with same structure as source_tgt
// first for insert stream
Target insertTarget = this.createRelationalTarget(
targetConnectionType,
"${dbTable.tableName+cloneSuffix}".toString());
// clone the fields for the target from the target source
insertTarget.setFields(source_tgt.getFields())
// specify an instance name to avoid collisions
insertTarget.setInstanceName(insertTarget.getInstanceName()+"_insert")

// second target instace for update branch
Target updateTarget = this.createRelationalTarget(
targetConnectionType,
"${dbTable.tableName+cloneSuffix}".toString());
// also same columns
updateTarget.setFields(source_tgt.getFields())
// also an unique instance name instead of default
updateTarget.setInstanceName(updateTarget.getInstanceName()+"_update")

// add both targets to the mapping
mapping.addTarget(insertTarget)
mapping.addTarget(updateTarget)
```

## Using the TransformationHelper

The Design API contains a TransformationHelper class for a mapping instance, which simplifies the creation of transformations within a mapping. It creates the transformation object instance and assigns all existing ports to one of the input groups or directly to the transformations and connects the new ports with the ports of the previous transformation. This allows you to use one method call to create an object with the appropriate columns linked to the previous object. To use this functionality, create a helper instance in the mapping instance.

The following sample code shows how to use the transformation helper instance:

```
// Create helper to simplify creation of further transformations
TransformHelper helper = new TransformHelper(mapping);
```

## Creating a Source Qualifier

You can use the transformationHelper instance to create the Source Qualifier for a source object. The transformationHelper instance method creates RowSets, which reference the ports and a port Link or Propagation set. Obtain an object handle before you assign attributes to a transformation object instance. Use the mapping instance method getTransformation(), which requires the name of the generated mapping as input.

The Source Qualifier transformations have the same name as the associated source prefixed with “SQ_”. You can use this name to retrieve the transformation object reference. Setting the number of sorted ports to the number of primary key columns ensures that ports are sorted in the right order to read from the database. Set the source filter based on the Valid_To column so that only the rows to be updated are retrieved from the source.

The following sample code shows how the source qualifier is created and assigned the properties for sorted read:

```
// Pipeline creation, one per source defined in the folder (of this object model)
// create mapping source and DSQ with helper and use it via RowSet
RowSet dsqRS = (RowSet) helper.sourceQualifier(source).getRowSets().get(0);
RowSet dsqRStgt = (RowSet) helper.sourceQualifier(source_tgt).getRowSets().get(0);

// set Sorted ports = num primary keys fields of source for both sq
// (target have only one record with selection criterion)
// for targetSq additionally set section criteria to VALID_TO=31.12.9999

// get properties object for sq of source table via name of the transformation (prefix
SQ_s_)
def sqProps=mapping.getTransformation("SQ_s_${dbTable.tableName}").getProperties()
// set property value sorted ports
sqProps.setProperty(TransformPropsConstants.NO_OF_SORTED_PORTS,
dbTable.pkColNameList.size().toString())

// get properties object for sq of target table via name of the transformation (prefix
SQ_s_)
def sqPropsTgt=mapping.getTransformation("SQ_s_$
{dbTable.tableName}"+cloneSuffix).getProperties()
// set properties for #sorted ports
sqPropsTgt.setProperty(TransformPropsConstants.NO_OF_SORTED_PORTS,
dbTable.pkColNameList.size().toString())
// set selection criterion
sqPropsTgt.setProperty(TransformPropsConstants.SOURCE_FILTER,
dbTable.tableName+cloneSuffix+".VALID_TO >= TO_DATE('31.12.9999','DD.MM.YYYY' )")
```

## Creating a Full Outer Join

Use a full outer join in a Joiner transformation to merge the source streams. Use the constructor InputSet to specify the Join Properties JOIN_TYPE = “Full Outer Join” and SORTED_INPUT = “YES” and convert the RowSet into InputSet Objects. The InputSet constructor requires a RowSet and a PortPropagationContext object. The class PortPropagationContextFactory has a getContextForAllIncludeCols() method to create an input set based on all columns of a row set.

Additionally, the Joiner transformation requires the detail table to be input as InputSet packaged as a Vector with one element. The ports of the detail type are prefixed with “IN_” to prevent port name collisions. Compare the ports with prefixed names in the detail table with the ports with no prefix of the master table.

The join condition is created as string, which references the equality of all primary key columns in source and “target” source, combined with an AND clause. Shortening the Join string by 5 characters truncates the trailing “ AND ”.

The helper.join instance method generates a Joiner transformation with all the parameters, adds the properties, and connects the ports of the Source Qualifier to the Joiner transformation.

The helper method returns a RowSet object, which can be retrieved with the getRowSet() instance method. This method can return multiple output groups. Get the handle to the first group to get a handle for the output RowSet.

The following sample code shows how the Joiner transformation instance is created with a full outer join and sorted input:

```
// Full outer join, sorted input, PK values must be equal
// create a properties object
TransformationProperties props = new TransformationProperties(); // properties
// set full outer join
props.setProperty(TransformPropsConstants.JOIN_TYPE,"Full Outer Join")
// set sorted input to YES
```

```
props.setProperty(TransformPropsConstants.SORTED_INPUT ,"YES")
// create input sets as join helper needs input sets if the properties should be given
on creation
InputSet dsqIS = new InputSet(dsqRS,
    PortPropagationContextFactory.getContextForAllIncludeCols() )
InputSet dsqIStgt = new InputSet(dsqRStgt,
    PortPropagationContextFactory.getContextForAllIncludeCols() )

// the detail input set must be in a vector for the join helper (by which reasons ever)
Vector vInputSets = new Vector()
vInputSets.add(dsqIS); // collection includes only the detail

// construct the join condition by iterating over the PK
def joinCondition = ""
dbTable.pkColNameList.each{ joinCondition +="${it} = IN_${it} AND "}
// ignore the trailing " AND" (5 chars)
joinCondition = joinCondition[0..joinCondition.size()-5]

// now create the joiner
RowSet joinRS = (RowSet) helper.join(
    vInputSets,
    dsqIStgt,
    joinCondition,
    props,
    "jnr_FullOuterJoin_src_tgt").getRowSets().get(0)
```

## Creating an Expression Transformation with Additional Fields

You must add the new TransformField object instances of the Joiner transformation to an Expression transformation with new columns containing derived values.

For this example, the application requires an output only port NOW with the timestamp of session start time and another port JUSTBEFORESESSIONSTART containing the timestamp just before the session start time. The value for NOW is taken from the pre-defined variable SESSSTARTTIME. The value for JUSTBEFORESESSIONSTART is calculated from the same variable minus 1 ns. The application also sets an output port variable MAXDAY to the maximum available date. The rest of the added fields reference pre-defined Mapping variables. The newly created column names are suffixed with “_new”.

To create the Expression transformation object, use the RowSet object representing the Joiner transformation and the vector of the TransformField object as parameters of the helper method.

The following sample code shows the steps to create the Expression transformation instance:

```
// collect a set (vector) for the added transform fields in the coming expression
Vector vFields = new Vector()
vFields.add(new TransformField("date/time (29, 9) NOW= SESSSTARTTIME"))
vFields.add(new TransformField("date/time (29, 9) JUSTBEFORESESSIONSTART=
ADD_TO_DATE(SESSSTARTTIME,'NS',-1)"))
vFields.add(new TransformField("date/time (29, 9) DOOMSDAY=
MAKE_DATE_TIME(9999,12,31,23,59,59,999999999)"))
vFields.add(new TransformField("string (255, 0) pmmapping_new= \$PMMappingName"))
vFields.add(new TransformField("string (255, 0) pmsession_new= \$PMSessionName"))
vFields.add(new TransformField("string (255, 0) pmworkflow_new= \$PMWorkflowName"))
vFields.add(new TransformField("string (255, 0) pmreposervice_new= \
$PMRepositoryServiceName"))

// create an expression with helper to get them connected
RowSet expRS = (RowSet) helper.expression(joinRS,
    vFields,"exp_addFields"+source.getName()).getRowSets().get(0);
```

The Expression transformation contains all ports of the source table, the “target” source table, and additional ports. To get the handle of the RowSet, get the handle of the first group of the getRowSets() instance method call.

## Creating a Router Transformation

The next step is to create a Router transformation with two defined output groups, one for the update stream and one for the insert stream. Initialize an empty vector object instance and fill it with TransformGroup object instances. The insertTransformGroup() and updateTransformGroup() method create groups, and the mapping helper.router instance creates the Router transformation. Note that the returned OutputSet consists of all Output ports of all groups, including the two specified groups with names “INSERT” and “UPDATE” and the default group “DEFAULT1”.

The following sample code shows how to create a Router transformation with two output groups:

```
// Create a TransformGroup for the router (see TransformGroup creation in own method)
Vector vTransformGrp = new Vector();
vTransformGrp.add( insertTransformGroup() );
vTransformGrp.add( updateTransformGroup() );
// create a Router Transformation
OutputSet routerOutputSet = helper.router( expRS, vTransformGrp,
          "Router_transform" );
```

## Using the insertTransformGroup() Method

This auxiliary method creates the TransformationGroup instance for the insert stream and specifies the required attributes, the group name, and the Router Transformation condition as string. The condition string compares columns that are not null. If one of the ports has a null value, the other ports have to be checked for non-null values. The check for non-null values has to be repeated for each of the non-primary key columns.

In the application, the compare string is created and then embedded in a Groovy String. The method returns the created TransformGroup object instance.

The following sample code shows how the insert group of Router transformation is created:

```
/*
 * Creates Transformgroup for the router insert branch
 */
TransformGroup insertTransformGroup() {
    // PK fields does not exists in target
    // Any field was different in target and source

    // create the check snipplet for NULL in PK's
    def pkNull = ""
    dbTable.pkColNameList.each{ pkNull +="isNull(${it}) OR "}
    // ignore the trailing " AND" (5 chars)
    pkNull = pkNull[0..pkNull.size()-4]

    // Create the code snipplet to check for different fields for all nonPK fields
    // Note that NULL does not compare well, so we have more stuff to check!
    def fieldDiffer = ""
    dbTable.payloadColNameList.each{
fieldDiffer +=
"""
(
(isNull(IN_${it}) AND  NOT isNull(${it}))
OR
( NOT isNull(IN_${it}) AND isNull(${it}))
OR
(IN_${it} != ${it})
)
 OR """
    // ignore the trailing " AND" (5 chars)
    fieldDiffer = fieldDiffer[0..fieldDiffer.size()-4]
    // Create the group for insert branch
    TransformGroup transGrp = new TransformGroup( "INSERT",
"""
IIF(${pkNull},
    1,
    IIF(${fieldDiffer},1,0)
)
""")
return transGrp
}
```

A similar function creates the TransformationGroup instance for the update case. As both group conditions can intersect, both streams can be filled with data from the same input row.

## Creating the Update Strategy Transformation for the Insert Group

The OutputSet of the Router transformation instance contains at least two groups. You can use the OutputSet instance method getRowSet with the group name parameter to access the Rowset for a single group. To limit the ports to only those required, you can create an InputSet object instance from the RowSet either by referencing all required ports or by removing all unnecessary ports. The SlowChagingDimension example removes all unnecessary ports.

The application creates a String[] excludes list with the names of the unnecessary ports, including the name of the “target” source ports and auxiliary fields. The names of the output group ports are dynamically generated and suffixed with the group number. The first group is the INSERT group, the second group is the UPDATE group, and the third group is the DEFAULT group. The names of the ports are suffixed by the group number. For example, ports in the INSERT group has the suffix 1 and ports in the UPDATE group has the suffix 2.

The following sample code shows how the updateStrategy object is created with the helper.updateStrategy() instance method and the exclude context. All rows flagged with DD_INSERT will be inserted.

```
// get the rowset for the INSERT group
RowSet insertRS = routerOutputSet.getRowSet( "INSERT" );

// Now make a fitting inputset for this
// all IN_ ports have to be connected natively
// a valid_from connected with NOW
// a valid_to connected with DOOMSDAY
// all PM field

// initialize excludes list
def excludes = []
// exclude the parts for target source, as they are not needed on insert new
[dbTable.colNameList, "valid_from", "valid_to", "JUSTBEFORESESSIONSTART", "pmmapping",
"pmsession", "pmworkflow", "pmreposervice"]
.flatten().each{ it = it+"1"; excludes <<
it }

// make a new input set using the excludes
InputSet usInsertIS = new InputSet(insertRS,
PortPropagationContextFactory
.getContextForExcludeColsFromAll((String[])excludes))

// make a new update strategy transformation with helper
RowSet updateStrategyInsertRS = (RowSet) helper.updateStrategy( usInsertIS,
        "DD_INSERT", "usInsert" )
        .getRowSets().get( 0 );
```

The RowSet returned by updateStrategyInsertRS contains column names that cannot be used as target port names and cannot be used as parameters for a writeTarget call. If they are used as parameters, the columns of the new target instance will not have the correct names. To avoid this problem, the application uses a variant of the writeTarget method that creates the port assignments with a link map.

## Creating a Target and Connecting with a Link Map

To create a link map, the application uses an helper method linkMapForInsert. Based on the link map, you can create an InputSet object for the insertTarget instance that connects the output ports of the Update Strategy transformation to the input ports of the insertTarget target instance. The call to the writeTarget method connects the ports and adds the target to the folder.

The following sample code shows how the linkMap can be used to create the input set:

```
// now link the columns of the rowset to the insertTargets. The simplest approach
// is to use a linkmap which a specific mapping will provide for the target fields
InputSet ilIS = new InputSet(updateStrategyInsertRS,
    PortLinkContextFactory.getPortLinkContextByMap(

    linkMapForInsert(updateStrategyInsertRS,insertTarget,"1")
    )
)
// connect the update stategy transformation (write to target is not done as the target
was created to the mapping before)
mapping.writeTarget( ilIS, insertTarget);
```

The update data stream follows a similar algorithm as the insert stream. The update strategy flags the rows for update and selects other ports. The helper method linkMapForUpdate creates the link map.

## Using the linkMapForInsert Method

The helper method linkMapForInsert creates the link map for the insert target and handles ports with names that do not match. Use the target.getFields() method to get the field names of the target instance and iterate through all the fields. Create the link with a switch case or an if-else-if cascade on matching name rules. The link map assignment creates connections between individual ports.

The following sample code shows how to use the helper method linkMapForInsert:

```
/**
 * Makes a linkMap for fields between target and rowset in insert stream
 * the 4 pm fields are mapped
 * @param fromRS Rowset from the last transformation (usually update strategy)
 * @param target target instance objects with the fields
 * @param suffix Field suffix of fields, usually numbers generated by router group
 * @return linkMap<Field,Field>
 */
Map <Field, Field> linkMapForInsert (RowSet fromRS, Target target, String suffix) {
    // Make an empty linkMap for collecting the result
    Map<Field, Field> linkMap = new LinkedHashMap<Field, Field>();
    // Iterate over all target fields
    target.getFields().each {
// take field name
def fName = it.getName()
// check for the pm fields
if (    fName ==~ /pmmapping/ ||
fName ==~ /pmsession/ ||
fName ==~ /pmworkflow/ ||
fName ==~ /pmreposervice/)
{
    def f = fromRS.getField(fName+"_new"+suffix)
if (f)
linkMap.put(f,it)
}
// check for the valid_From field
else if (fName ==~ /valid_From/) {
        def f = fromRS.getField("NOW"+suffix)
        if (f)
linkMap.put(f,it)
}
// check for the valid_To field
else if (fName ==~ /valid_To/) {
        def f = fromRS.getField("DOOMSDAY"+suffix)
        if (f)
linkMap.put(f,it)
}
// all other fields
else {
       def f = fromRS.getField("IN_"+fName+suffix)
       if(f)
linkMap.put(f,it)
}
    }
    return linkMap
}
```

You can also use the Pattern Match Strategy to link the ports. You can use the PatternMatchStrategy class to use a regular expression to propagate and link ports. For example, the regex pattern _1$ propagates all ports with the suffix _1 and maps them to ports with the same name but without the suffix.

### Related Topics:

- “Sample Patterns for Regular Expressions for Port Propagation” on page 177

## Assigning the Connection Types

The application sets the connection properties for the source and target instances and sets the update target to UPDATE_AS_UPDATE. If the update target is not set, the default setting of UPDATE_AS_INSERT creates duplicate keys.

The following sample code shows the connection attributes for the target instance and the mapping object added to the folder:

```
// Now we have to assign some properties to the sources and targets
// set the connection properties
source.getConnInfo().getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME,s
ourceConnectionName)
source_tgt.getConnInfo().getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNA
ME,targetConnectionName)
insertTarget.getConnInfo().getConnProps().setProperty(ConnectionPropsConstants.CONNECTION
NAME,targetConnectionName)
updateTarget.getConnInfo().getConnProps().setProperty(ConnectionPropsConstants.CONNECTION
NAME,targetConnectionName)
// set the update target to "Update as Update" to avoid write failure
updateTarget.getConnInfo().getConnProps().setProperty(ConnectionPropsConstants.RELATIONAL
_UPDATE_AS_UPDATE ,"YES")
// add mapping to folger
folder.addMapping(mapping);
```

## Creating the Session

The createSession method creates a Session object and initializes it with a name derived from the mapping. The name is prefixed with s_. The application sets the value of the TREAT_SOURCE_ROWS_AS property to “Data driven” to enable the Update Strategy transformation to update based on the data processed. The setMapping() instance method assigns the mapping to the session.

The following sample code shows how to create the session:

```
session = new Session("s_"+mapping.getName(), "s_"+mapping.getName(),
"Session for mapping "+mapping.getName());
// set session to data driven to allow update startegy transformation to work
session.getProperties().setProperty( SessionPropsConstants.TREAT_SOURCE_ROWS_AS,
        "Data driven" );
session.setMapping(mapping);
```

You can use the setTaskInstanceProperty method of the session instance to set task properties. The Design API version 8.6.x supports reusable session task objects. It does not support the call `session.setTaskInstanceProperty(Task.REUSABLE,Task.NO_OPTION)`.

### Creating the Workflow

The constructor for the method to create a workflow requires the workflow name, which is derived from the mapping name. Use the addSession() instance method to assign the mapping to the session and add the session to the workflow. The workflow property SERVERNAME contains the name of the Integration Service to run the workflow. The addWorkflow method assigns the workflow to the folder.

The following sample code shows how the workflow is created:

```
workflow = new Workflow("wf_"+mapping.getName(), "wf_"+mapping.getName(),
"Workflow for session s_"+mapping.getName());
workflow.addSession(session);
// set a integration server name to run on
workflow.getProperties().setProperty("SERVERNAME",pcisServerName)
folder.addWorkFlow(workflow);
```

### Saving PowerCenter Objects to the Repository

After PowerCenter objects have been added to the repository, the JMFSlowCd instance calls generateOutput() to transfer the objects into a PowerCenter export file as specified in the *pmrep* connection properties. If the doImport variable is set to True, then the objects in the export file will be imported to the repository. The *pmrep* configuration file must be configured for the import.

The following sample code shows the generateOutput() method:

```
MapFwkOutputContext outputContext = new MapFwkOutputContext(
    MapFwkOutputContext.OUTPUT_FORMAT_XML,
    MapFwkOutputContext.OUTPUT_TARGET_FILE,
    mapFileName);
boolean doImport = true;
rep.save(outputContext, doImport);
```

The sample application does not send the output of the *pmrep* command to a log file. It sends the output to stdout. If the save method attempts to write a locked PowerCenter object to the repository, an exception can occur. A PowerCenter object may be locked if it is used during a Designer session.

# Installing and Running the Sample Application

The SlowChangingDimensions application was built as a web application running in a Groovy on Grails environment. It is shipped as a ZIP file containing all Groovy sources and compiled class files, properties files, and required jar files. The application runs on Windows with PowerCenter 8.6.x.

## Setting Up the Runtime Environment

To run an application that uses the Design API, you must have the following files:

- **Application libraries.** Includes all libraries and files required to run the application.
- **JDK or JRE 1.8.** You can use the JDK installed with PowerCenter.
- **pcconfig.properties.** Contains PowerCenter repository connection information required by any plug-in that calls methods in the Design API.
- **jmfdemo.properties.** Contains the configuration information required to run the SlowChangingDimensions sample application. This configuration file is required only for the SlowChangingDimensions sample application. You do not need this file for other applications or plug-ins.

## pcconfig.properties File

The following table describes the properties in the pcconfig.properties file:

| Property | Description |
|---|---|
| PC_CLIENT_INSTALL_PATH | Path where *pmrep* is located. |
| TARGET_FOLDER_NAME | Folder name in which to create mappings, sessions, and workflows. |
| TARGET_REPO_NAME | Name of the repository. |
| REPO_SERVER_HOST | Host for the Repository Service. |
| REPO_SERVER_PORT | Port for the Repository Service. |
| ADMIN_USERNAME | User name of the repository administrator. |
| ADMIN_PASSWORD | Password for the repository administrator. |
| SERVER_PORT | Not used. |
| DATABASETYPE | Not used. |
| PMREP_CACHE_FOLDER | Cache folder for intermediate files created by *pmrep*. |

## jmfdemo.properties File

The sample application reads the configuration information from the jmfdemo.properties file. Before you run SlowChangingDimension, modify the options in the jmfdemo.properties file to match your PowerCenter environment.

The following table describes the properties in the jmfdemo.properties file:

| Property | Description |
|---|---|
| url | JDBC URL. Connection string to the PowerCenter repository database.<br>Default is `jdbc:oracle:thin:@localhost:1521:ORCL`. |
| userName | User name for the user account to log in to the repository. |
| password | Password for the user account to log in to the repository. |
| driverName | Name of the JDBC driver. Default is `oracle.jdbc.OracleDriver`. |
| defaultSourceConnection | PowerCenter connection object for the source. |
| defaultTargetConnection | PowerCenter connection object for the target. |
| logLevel | Level of error messages to write to the log file. Default is INFO. |
| pcisServerName | Name of the Integration Service to run the workflow. |

| Property | Description |
|---|---|
| defaultTableName | Name of the source table. |
| cloneSuffix | Suffix to append to the name of a cloned table. Default is _SCL. |

When you initially run the sample application, the application reads the connection information from the PowerCenter repository configured in the jmfdemo.properties file. The application saves the connection information in a file named pmConnections.xml. When you subsequently run the sample application, it reads the connection information from the pmConnections.xml file instead of the repository.

To force the sample application to read the connection information from the PowerCenter repository, delete the pmConnections.xml file or run the application with the *-c* or *--getconnections* option. When you run the application with the -c option, the application reads the connection information from the PowerCenter repository and overwrites the pmConnections.xml file with the new connection information.

## Running the Sample Application

To start the sample application:

1. Log in to the database where you want to create the target table and create the CUSTOMER_DIM_SLC table.

   You can also use the Designer to run the SQL statements to create the tables based on the target object.
2. Extract the zipped example files into any directory.
3. Go to the directory where you extract the files.
4. In the /JMFLIB folder, open the jmfdemo.properties file and modify the properties to match your PowerCenter environment.
5. Run the SlowChangingDimension.bat file.

## Recompiling the Sample Application

To modify or extend the sample application, you can recompile the source files. To recompile, you need the JDK 1.8 installed with PowerCenter. You also need the Groovy software.

Groovy is an open source language that you can download from the following web site:

```
http://groovy.codehaus.org/Download
```

The sample application was created using Groovy version 1.5.6. You can use an IDE such as Eclipse or use the Groovy compiler groovyc to compile on the command line. After you install Groovy, you can set the GROOVY_HOME environment variable and use the compile.bat file included in the zip file to compile the source.

## Limitations of the Sample Application

The SlowChangingDimension sample application has the following limitations:

- The target table is not created by default. You must create the target table before you run the application.
- The application does not verify that the target folder exists. An exception occurs if the target does not exist and *pmrep* fails to import the XML file.

APPENDIX A

# Design API Sample Code

This appendix includes the following topics:

## Design API Sample Code Overview

This appendix provides sample code that you can use when you build plug-ins with the Design API. Modify the sample code to match your repository objects and include them in your application. Or, modify the code to extend the functionality or to accomplish a similar function.

This appendix also provides examples of regular expressions that you can use in your port propagation strategies.

## Sample Code for the Design API

This section provides sample code that shows how to use the Design API to create and work with PowerCenter objects.

### Using Mapplets

The following sample code shows how to create a mapplet and use it in a mapping:

```
//creating a mapplet object
        Mapplet mapplet = new Mapplet("MappletSample", "MappletSample",
                "This is a MappletSample mapplet");

        // create helper for mapplet
        TransformHelper helperMapplet = new TransformHelper(mapplet);

        //creating transformations for mapplet
        OutputSet outputSet = helperMapplet.inputTransform(getIdRs(),
                "InputIDTransform");
        RowSet inputIDRS = (RowSet) outputSet.getRowSets().get(0);

        RowSet FilterRS = (RowSet) helperMapplet.filter(inputIDRS, "TRUE",
                "FilereTrans").getRowSets().get(0);
```

```
 PortPropagationContext filterRSContext = PortPropagationContextFactory
         .getContextForAllIncludeCols();

 // create a lookup transformation
 outputSet = helperMapplet.lookup(FilterRS, idPostSrc, "ID = IN_ID",
         "Lookup_IdPost_Table");
 RowSet lookupRS = (RowSet) outputSet.getRowSets().get(0);

 PortPropagationContext lkpRSContext = PortPropagationContextFactory
         .getContextForIncludeCols(new String[] { "post" });

 List<InputSet> inputSets = new ArrayList<InputSet>();
 inputSets.add(new InputSet(FilterRS, filterRSContext));

 inputSets.add(new InputSet(lookupRS, lkpRSContext));

 helperMapplet.outputTransform(inputSets, "outputIdPost");

 //adding this mapplet to folder
 folder.addMapplet(mapplet);

 //creating mapping object that will use above created mapplet
 mapping = new Mapping("MappletSampleMapping", "MappletSampleMapping",
         "This is a sample for mapplet mapping");

 //setting mapping file name
 setMapFileName(mapping);

 // create helper for mapping..
 TransformHelper helperMapping = new TransformHelper(mapping);

 //creating source qualifier
 OutputSet outputSet = helperMapping.sourceQualifier(idSrc);
 RowSet dsqIdRS = (RowSet) outputSet.getRowSets().get(0);

 //creating List of InputSet that will be used for creating mapplet tranformation
 List<InputSet> inSets = new ArrayList<InputSet>();
 inSets.add(new InputSet(dsqIdRS));

 //creating mapplet transformation
outputSet = helperMapping.mapplet(mapplet, inSets,
         "myMapplet");

 List<RowSet> vMappRS = outputSet.getRowSets();


 //  write to target
 mapping.writeTarget((RowSet) vMappRS.get(0), idPostTrg);
```

## Creating and Using Shorcuts

Shortcuts are references to reusable objects from a shared folder.

The following sample code shows how to refer to a shortcut to a source in a mapping:

```
ShortCut scSrc=new ShortCut("sc_src_age","shortcut to
source","Repo_rjain","temp","age",RepositoryObjectConstants.OBJTYPE_SOURCE,ShortCut.LOCAL
);
folder.addShortCut(scSrc);
mapping.addShortCut(scSrc);
scSrc.setRefObject(mySource); // mySource is the source object fetched from repository.
OutputSet outSet = helper.sourceQualifier(scSrc);
RowSet dsqRS = (RowSet) outSet.getRowSets().get( 0 );
```

# Validating Objects

You can validate objects in the repository. For example, when you validate a mapping, you can find missing or incorrect links in the mapping.

The following sample code shows how to validate a mapping:

```
CachedRepositoryConnectionManager connMgr = new CachedRepositoryConnectionManager(new
PmrepRepositoryConnectionManager());
rep.setRepositoryConnectionManager(connMgr);

String Writer outputSummary = new StringWriter();
// validate mapping
try
{
    connMgr.validate(mapping.getName(), RepositoryObjectConstants.OBJTYPE_MAPPING,
folder.getName(), true,outputSummary);
} catch (RepoConnectionObjectOperationException e)
{
            e.printStackTrace();
} catch (RepoObjectValidationFailedException e)
{

    e.printStackTrace();
} catch (IOException e)
{
    e.printStackTrace();
}
```

# Creating Parameter Files

A session parameter file contains parameters for mappings and sessions and is used when the PowerCenter Integration Service runs a session.

The following sample code shows how to create a PowerCenter session parameter file:

```
List<MappingVariable> vMappingVars = new ArrayList<MappingVariable>();

// While creating mapping variables, we will use transformation data types as mapping
variables
// belong to mapping, not to source or target.

MappingVariable mpVar1 = new
MappingVariable("MAX",MappingVariableDataTypes.STRING,"default
value1","description",true,"$$var1","20","0",true);
MappingVariable mpVar2 = new
MappingVariable("MIN",MappingVariableDataTypes.INTEGER,"100","description",false,"$
$var2","10","0",true);
MappingVariable mpVar3 = new
MappingVariable("MAX",MappingVariableDataTypes.NSTRING,"default
value3","description",true,"$$var3","10","0",true);
MappingVariable mpVar4 = new
MappingVariable("MAX",MappingVariableDataTypes.INTEGER,"101","description",false,"$
$var4","10","0",true);
MappingVariable mpVar5 = new
MappingVariable("MIN",MappingVariableDataTypes.STRING,"default
value5","description",true,"$$var5","15","0",true);
MappingVariable mpVar6 = new
MappingVariable("MAX",MappingVariableDataTypes.NSTRING,"default
value6","description",false,"$$var6","20","0",true);

vMappingVars.add(mpVar1);
vMappingVars.add(mpVar2);
vMappingVars.add(mpVar3);
vMappingVars.add(mpVar4);
vMappingVars.add(mpVar5);
vMappingVars.add(mpVar6);

workflow = new Workflow("Workflow_for_parameter_file",
                "Workflow_for_parameter_file", "This workflow for parameter file");
workflow.addSession(session);
workflow.setParentFolder(folder);
workflow.addWorkflowVariables(vMappingVars);
folder.addWorkFlow(workflow);

List<String> listOfParams = workflow.getListOfParameters();

ParameterFile pmFile = new ParameterFile("C:\\param.save");

Iterator<String> listOfParamsIter = listOfParams.iterator();
int i=0;
while(listOfParamsIter.hasNext())
{
    pmFile.setParameterValue(listOfParamsIter.next(), new Integer(i).toString());
    i++;
}
pmFile.save();
```

## Using Partitions

The following sample code shows how to use partitions in a mapping:

```
protected void createMappings() throws Exception {
        // create a mapping
        mapping = new Mapping( "PartitioningExample", " PartitioningExample ", "This is
a partitioning example." );
        setMapFileName( mapping );
        TransformHelper helper = new TransformHelper( mapping );
        // creating DSQ Transformation
        OutputSet outSet = helper.sourceQualifier( employeeSrc );
        RowSet dsqRS = (RowSet) outSet.getRowSets().get( 0 );
        TransformationProperties props = new TransformationProperties();

        List<InputSet> rows = new ArrayList<InputSet>();
        rows.add( new InputSet(dsqRS) );

        // create a Filter transformation with partition point details as argument
        // filter out rows that don't belong to USA
        RowSet filterRS = (RowSet) helper.filter( rows, "Country = 'USA'", props,
"filter_transform",  createRR_PPD(3))
                .getRowSets().get( 0 );
        // write to target
        mapping.writeTarget( filterRS, outputTarget );
        folder.addMapping( mapping );
    }

    private PartitionPointDetails createRR_PPD(int n) {
        PartitionPointDetails ppd = new PartitionPointDetails();

        ppd.setPolicy(new RoundRobinPartitioningPolicy());  //RoundRobin Partition
Policy applied here
        List<Partition> parts = new ArrayList<Partition>();
        for (int i = 0; i < n; ++i) {
            parts.add(new Partition(""));
        }
        ppd.setPartitions(parts);

        return ppd;
    }
```

## Writing to an SAP Table

The following sample code shows how to use an SAP target in a mapping:

```
package com.informatica.powercenter.sdk.mapfwk.samples;

import java.util.ArrayList;
import java.util.List;

import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionPropsConstants;
import com.informatica.powercenter.sdk.mapfwk.connection.SourceTargetType;
import com.informatica.powercenter.sdk.mapfwk.core.Field;
import com.informatica.powercenter.sdk.mapfwk.core.FieldKeyType;
import com.informatica.powercenter.sdk.mapfwk.core.FieldType;
import com.informatica.powercenter.sdk.mapfwk.core.InputSet;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.NativeDataTypes;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.Source;
import com.informatica.powercenter.sdk.mapfwk.core.StringConstants;
import com.informatica.powercenter.sdk.mapfwk.core.Target;
import com.informatica.powercenter.sdk.mapfwk.core.Transformation;
import com.informatica.powercenter.sdk.mapfwk.core.TransformationConstants;
import com.informatica.powercenter.sdk.mapfwk.core.TransformationContext;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;
import com.informatica.powercenter.sdk.mapfwk.metaextension.MetaExtension;

public class SAPWriterSample extends Base{

    protected Target target;
    protected Source source;
    protected Transformation dsqTransform;

    public SAPWriterSample() {
        target = null;
        source = null;
        dsqTransform = null;
    }

    @Override
    protected void createSources() {
        // TODO Auto-generated method stub
        this.source = this.createOracleJobSource("OrclSRC_SAPWRTER");
        this.folder.addSource(this.source);
    }

    @Override
    protected void createTargets() {
        // TODO Auto-generated method stub
//        List<Field> fields = new ArrayList<Field>();
//
//        Field mandtField = new Field( "JOB_ID", "Client", "Client",
//                NativeDataTypes.SAP.CLNT, "3", "0",
//                    FieldKeyType.PRIMARY_KEY, FieldType.SOURCE, true );
//        fields.add( mandtField );
//
//        Field eblnField = new Field( "EBELN", "Purchasing document number",
"Purchasing document number",
//                NativeDataTypes.SAP.CHAR, "10", "0",
//                FieldKeyType.PRIMARY_FOREIGN_KEY, FieldType.SOURCE, true );
//        fields.add( eblnField );
//
//        Field bukrsField = new Field( "BUKRS", "Company Code", "Company Code",
//                NativeDataTypes.SAP.CHAR, "4", "0",
//                FieldKeyType.NOT_A_KEY, FieldType.SOURCE, true );
//        fields.add( bukrsField );
//
//        Field pincrField = new Field( "PINCR", "Item number interval", "Item number
interval",
//                NativeDataTypes.SAP.NUMC, "5", "0",
//                FieldKeyType.NOT_A_KEY, FieldType.SOURCE, false );
//        fields.add( pincrField );

        //ConnectionInfo info = getRelationalConnInfo( SourceTargetType.SAPTableWriter ,
"SAPTgt_SAPWRTER");
        this.target = this.createRelationalTarget(SourceTargetType.SAPTableWriter,
"SAPTgt_SAPWRTER");
        //target = new Target( "EKKO", "Purchasing Document Header11", "Purchasing
Document Header1", "EKKO", info );
//        target.setFields(fields);

        this.folder.addTarget(this.target);
        //

    }

    @Override
    protected void createMappings() throws Exception {
        // TODO Auto-generated method stub
        mapping = new Mapping("Mapping_SAP_Table_Writer", "Mapping_SAP_Table_Writer",
"This is Mapping_SAP_Table_Writer");
        setMapFileName(mapping);

        List<InputSet> inputSets = new ArrayList<InputSet>();
        InputSet tptSource = new InputSet(this.source);
        inputSets.add(tptSource);

        TransformationContext tc = new TransformationContext( inputSets );
        dsqTransform = tc.createTransform( TransformationConstants.DSQ, "TPT_DSQ" );
        RowSet dsqRS = (RowSet) dsqTransform.apply().getRowSets().get( 0 );
        mapping.addTransformation( dsqTransform );

        mapping.writeTarget(dsqRS, this.target);
        folder.addMapping(mapping);
    }

    @Override
    protected void createSession() throws Exception {
        // TODO Auto-generated method stub
        session = new Session( "session_For_SAP_Table_Writer",
"Session_For_SAP_Table_Writer",
        "This is session for SAP_Table_Writer" );
        ConnectionInfo src_connInfo = new ConnectionInfo(SourceTargetType.Oracle);
        src_connInfo.getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME,
"Oracle");
        session.addConnectionInfoObject(source, src_connInfo);

        ConnectionInfo tgt_connInfo = new
ConnectionInfo(SourceTargetType.SAPTableWriter);
        tgt_connInfo.getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME,
"SAP_R3");
        session.addConnectionInfoObject(target, tgt_connInfo);
        session.setMapping( this.mapping );
    }

    @Override
    protected void createWorkflow() throws Exception {
        // TODO Auto-generated method stub
        workflow = new Workflow( "Workflow_for_SAP_Table_Writer",
"Workflow_for_SAP_Table_Writer",
        "This workflow for SAP_Table_Writer" );
        workflow.addSession( session );
        folder.addWorkFlow( workflow );
    }

    public static void main(String[] args) {
        try {
            SAPWriterSample SAPWriter = new SAPWriterSample();
            if (args.length > 0) {
                if (SAPWriter.validateRunMode( args[0] )) {
                    SAPWriter.execute();
                }
            } else {
                SAPWriter.printUsage();
            }
        } catch (Exception e) {
            e.printStackTrace();
```

```
                System.err.println( "Exception is: " + e.getMessage() );
            }
        }

}
```

# Using Multiple Instances of Sources and Targets

Some mappings require multiple instances of the same source. For example, a source has two pipelines or you join a source with the same source.

The following sample code shows how you can use the same source for two different pipelines:

```
public void createMappings() throws Exception {
// create a mapping
mapping = new Mapping( "SourceCloneMapping", "mapping", "Testing SourceClone sample" );
setMapFileName( mapping );
TransformHelper helper = new TransformHelper( mapping );
// creating DSQ Transformation
OutputSet outSet = helper.sourceQualifier( employeeSrc );
RowSet dsqRS = (RowSet) outSet.getRowSets().get( 0 );
// write to target
mapping.writeTarget( dsqRS, outputTarget );
// clone the source and target
Source empSrcClone = (Source) employeeSrc.clone();
empSrcClone.setInstanceName( empSrcClone.getName() + "_clone" );
Target targetClone = (Target) outputTarget.clone();
targetClone.setInstanceName( outputTarget.getName() + "_clone" );
mapping.addTarget( targetClone );
// create DSQ and write to target
outSet = helper.sourceQualifier( empSrcClone );
dsqRS = (RowSet) outSet.getRowSets().get( 0 );
mapping.writeTarget( dsqRS, targetClone );
folder.addMapping( mapping );
}
```

# Sources

The following code examples show how to create and use different types of sources in a mapping.

## Creating a Relational Source

The following sample code shows how to create a relational source object:

```
protected Source createOracleJobSource( String dbName) {
Source jobSource = null;

List<Field> fields = new ArrayList<Field>();
Field jobIDField = new Field( "JOB_ID", "JOB_ID", "",
                NativeDataTypes.Oracle.VARCHAR2, "10", "0",
                    FieldKeyType.PRIMARY_KEY, FieldType.SOURCE, true );
fields.add( jobIDField );

Field jobTitleField = new Field( "JOB_TITLE", "JOB_TITLE", "",
                NativeDataTypes.Oracle.VARCHAR2, "35", "0",
                FieldKeyType.NOT_A_KEY, FieldType.SOURCE, false );
fields.add( jobTitleField );

Field minSalField = new Field( "MIN_SALARY", "MIN_SALARY", "",
                NativeDataTypes.Oracle.NUMBER_PS, "6", "0",
                FieldKeyType.NOT_A_KEY, FieldType.SOURCE, false );
fields.add( minSalField );

Field maxSalField = new Field( "MAX_SALARY", "MAX_SALARY", "",
                NativeDataTypes.Oracle.NUMBER_PS, "6", "0",
                FieldKeyType.NOT_A_KEY, FieldType.SOURCE, false );
fields.add( maxSalField );

ConnectionInfo info = getRelationalConnInfo( SourceTargetType.Oracle , dbName);
jobSource = new Source( "JOBS", "JOBS", "This is JOBS table", "JOBS", info );
jobSource.setFields(fields);
return jobSource;
}
protected ConnectionInfo getRelationalConnInfo( SourceTargetType dbType, String dbName )
{
ConnectionInfo connInfo = null;
connInfo = new ConnectionInfo( dbType );
connInfo.getConnProps().setProperty( ConnectionPropsConstants.DBNAME, dbName );
return connInfo;
}
```

## Connecting to a DB2 Data Source

The following sample code shows how to define a data source to connect to DB2 for Linux, UNIX, and Windows:

```
protected void createSources()
{
    PowerConnectSourceFactory sourceFactory = PowerConnectSourceFactory.getInstance();
    try
    {
        source = sourceFactory.getPowerConnectSourceInstance("PWX_DB2UDB_CDC",
"mySource", "mySourceDBD", "mySource", "mySource");
        SourceGroup srcGrp = new SourceGroup("ct_ALLDTYPES_SRC",(String)null);

source.createField("DTL__CAPXRESTART1",srcGrp,"","","PACKED","25","0",FieldKeyType.NOT_A_
KEY,FieldType.SOURCE, false);

source.createField("DTL__CAPXRESTART2",srcGrp,"","","string","10","0",FieldKeyType.NOT_A_
KEY, FieldType.SOURCE, false);
        source.setMetaExtensionValue("Access Method", "V");
        source.setMetaExtensionValue("Map Name", "ct_ALLDTYPES_SRC");
        source.setMetaExtensionValue("Original Name" , "ALLDTYPES_SRC");
        source.setMetaExtensionValue("Original Schema", "PWXUDB");
        List<ConnectionInfo> connInfos = source.getConnInfos();
        for (int i=0;i<connInfos.size();i++)
        {
            ConnectionInfo connInfo = (ConnectionInfo) connInfos.get(i);

connInfo.getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME,
"myTestConnection");
connInfo.getConnProps().setProperty( ConnectionPropsConstants.DBNAME,"myDBName");

        }
    }catch (RepoOperationException e)
    {// TODO Auto-generated catch block
        e.printStackTrace();
    }catch (MapFwkException e)
    {// TODO Auto-generated catch block
        e.printStackTrace();
    }
    folder.addSource(source);
    this.mapFileName = "PowerExchangeSource.xml";
}
```

## Connecting to a Netezza Data Source

The following sample code shows how to use a Netezza data source in a mapping:

```
package com.informatica.powercenter.sdk.mapfwk.samples;

import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionPropsConstants;
import com.informatica.powercenter.sdk.mapfwk.connection.SourceTargetType;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.OutputSet;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.Source;
import com.informatica.powercenter.sdk.mapfwk.core.Target;
import com.informatica.powercenter.sdk.mapfwk.core.TransformField;
import com.informatica.powercenter.sdk.mapfwk.core.TransformHelper;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;

/**
 * This example applies a simple expression transformation on the Employee table
 * and writes to a target.
 *
 */
public class NetezzaSample extends Base {
    // ////////////////////////////////////////////////////////////////
    // Instance variables
    // ////////////////////////////////////////////////////////////////
    protected Source employeeSrc;
    protected Target outputTarget;

    /**
     * Create sources
     */
    protected void createSources() {
        employeeSrc = this.createNetezzaSource();
        folder.addSource( employeeSrc );
    }

    /**
     * Create targets
     */
    protected void createTargets() {
        outputTarget = this.createRelationalTarget(SourceTargetType.Flat_File, "Target");
    }

    public void createMappings() throws Exception {
        // create a mapping
        mapping = new Mapping( "NetezzaMapping", "mapping", "Testing Netezza sample" );
        setMapFileName( mapping );
        TransformHelper helper = new TransformHelper( mapping );
        // creating DSQ Transformation
        OutputSet outSet = helper.sourceQualifier( employeeSrc );
        RowSet dsqRS = (RowSet) outSet.getRowSets().get( 0 );
        // create an expression Transformation
        // the fields LastName and FirstName are concataneted to produce a new
        // field fullName
        String expr = "string(80, 0) fullName= firstName || lastName";
        TransformField outField = new TransformField( expr );
        String expr2 = "integer(10,0) YEAR_out=IIF(EmployeeID=0,2000,2001)";
        TransformField outField2 = new TransformField( expr2 );
        List<TransformField> transFields = new ArrayList<TransformField>();
        transFields.add( outField );
        transFields.add( outField2 );
        RowSet expRS = (RowSet) helper.expression( dsqRS, transFields,
"exp_transform" ).getRowSets()
                .get( 0 );
        // write to target
        mapping.writeTarget( expRS, outputTarget );
        folder.addMapping( mapping );
    }

    public static void main( String args[] ) {
        try {
            NetezzaSample expressionTrans = new NetezzaSample();
            if (args.length > 0) {
                if (expressionTrans.validateRunMode( args[0] )) {
                    expressionTrans.execute();
                }
            } else {
                expressionTrans.printUsage();
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println( "Exception is: " + e.getMessage() );
        }
    }

    /*
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSession()
     */
    protected void createSession() throws Exception {
        session = new Session( "Session_For_netezza", "Session_For_netezza",
                "This is session for Netezza" );
        session.setMapping( this.mapping );

        /* Configure all the Netezza connection related properties for DSQ at Session
level */

        Properties props = new Properties();

        props.put(ConnectionPropsConstants.CONNECTIONNAME, "Netezza");
        props.put(ConnectionPropsConstants.USER_DEFINED_JOIN, "asdasd");
        props.put(ConnectionPropsConstants.NUMBER_OF_SORTED_PORTS, "10");
        props.put(ConnectionPropsConstants.TRACING_LEVEL, "Verbose Data");
        props.put(ConnectionPropsConstants.PRE_SQL, "sdada");
        props.put(ConnectionPropsConstants.POST_SQL, "TD_OPER_CLI");
        props.put(ConnectionPropsConstants.SOURCE_FILTER, "asdad");

        session.addSessionTransformInstanceProperties(employeeSrc,  props);
    }

    /*
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createWorkflow()
     */
    protected void createWorkflow() throws Exception {
        workflow = new Workflow( "Workflow_netezza", "Workflow_netezza",
                "This workflow for Netezza" );
        workflow.addSession( session );
        folder.addWorkFlow( workflow );
    }
}
```

## Connecting to an SAP Data Source

The following sample code shows how to use an SAP source and source qualifier in a mapping:

```
package com.informatica.powercenter.sdk.mapfwk.samples;

import java.util.ArrayList;
import java.util.List;

import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionPropsConstants;
import com.informatica.powercenter.sdk.mapfwk.connection.SourceTargetType;
import com.informatica.powercenter.sdk.mapfwk.core.DSQTransformation;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.NativeDataTypes;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.SAPASQTransformation;
import com.informatica.powercenter.sdk.mapfwk.core.SAPFunction;
import com.informatica.powercenter.sdk.mapfwk.core.SAPScalarInputValueType;
import com.informatica.powercenter.sdk.mapfwk.core.SAPStructure;
import com.informatica.powercenter.sdk.mapfwk.core.SAPStructureField;
import com.informatica.powercenter.sdk.mapfwk.core.SAPStructureInstance;
import com.informatica.powercenter.sdk.mapfwk.core.SAPVariable;
import com.informatica.powercenter.sdk.mapfwk.core.SAPVariableCategory;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.Source;
import com.informatica.powercenter.sdk.mapfwk.core.StringConstants;
import com.informatica.powercenter.sdk.mapfwk.core.Target;
import com.informatica.powercenter.sdk.mapfwk.core.TransformField;
import com.informatica.powercenter.sdk.mapfwk.core.TransformHelper;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;

/**
 * This is a sample program that demonstrates how to create a mapping
 * with an SAP source and an SAP application source qualifier.
 *
 * 1. Create a mapping with following structure:
 * SAP source --> ASQ --> Expr Transform --> Flatfile Target
 *
 * 2. Create a session for the mapping.
 *
 * 3. Create workflow using the session created in step 2.
 *
 */
public class SAPMappingExample extends Base {

    private Mapping mapping = null;
    private Source ekkoSAPSrc = null;
    private Target fileTgt = null;
    private SAPASQTransformation dsq;

    private SAPStructure sapStruc1;
    private SAPStructure sapStruc2;
    private SAPStructure sapStruc3;
    private SAPStructure sapStruc4;
    private SAPStructure sapStruc5;

    /*
     * Create an SAP source
     */
    protected void createSources() {
        ekkoSAPSrc = this.createSAPekkoSource("sophie");
        folder.addSource(ekkoSAPSrc);
    }

    /*
     * Create a Flatfile target
     */
    protected void createTargets() {
        fileTgt = this.createFlatFileTarget("Output1");
    }

    /*
     * Create a mapping with the following structure:
  *   SAP Source --> ASQ --> Flatfile Target
     */
    protected void createMappings() throws Exception {

        // create a mapping object
        mapping = new Mapping("M_SAP_EKKO_SAMPLE_MAPPING",
                "M_SAP_EKKO_SAMPLE_MAPPING",
                "Mapping to test mapping with an SAP Source");
        setMapFileName(mapping);

        // create transform helper
        TransformHelper helper = new TransformHelper(mapping);

        // create DSQ
        dsq = (SAPASQTransformation) ekkoSAPSrc.createASQTransform();
        mapping.addTransformation(dsq);

        //Create Structures
        createSAPStructures();

        SAPVariable var1 = new SAPVariable("var1", SAPVariableCategory.ABAPTYPE,
                NativeDataTypes.SAP.CHAR, "0", "10", "10", true);
        dsq.addSAPVariable(var1);
        SAPVariable var2 = new SAPVariable("var2", SAPVariableCategory.STRUCTURETYPE,
"Definition");
        dsq.addSAPVariable(var2);
        SAPVariable var3 = new SAPVariable("var3",
SAPVariableCategory.STRUCTUREFIELDTYPE, "initialValue", "def-def");
        dsq.addSAPVariable(var3);

        /**
         * SAPFunction with ScalarInput and Table
         */
        SAPFunction func2 = new SAPFunction("Z_PM_RFC_RUN_PROGRAM", "A RFC callable
function to run programs for staging data in files.");

        SAPStructureField fld1 = new SAPStructureField("/INFATRAN/ZPMDATATYPE-CHAR32",
NativeDataTypes.SAP.CHAR, "PROGRAM_NAME", "32", "0", false, "<None>",
SAPScalarInputValueType.NONE);
        func2.addSAPFunctionScalarInput(fld1);
        SAPStructureField fld2 = new SAPStructureField("/INFATRAN/ZPMDATATYPE-FILENAME",
NativeDataTypes.SAP.CHAR, "OUTPUT_FILENAME", "128", "0", false, "<None>",
SAPScalarInputValueType.NONE);
        func2.addSAPFunctionScalarInput(fld2);
        SAPStructureField fld3 = new SAPStructureField("/INFATRAN/ZPMDATATYPE-CHAR32",
NativeDataTypes.SAP.CHAR, "FORM_NAME", "32", "0", false, "<None>",
SAPScalarInputValueType.NONE);
        func2.addSAPFunctionScalarInput(fld3);
        SAPStructureField fld4 = new SAPStructureField("/INFATRAN/ZPRAMS-PARAMKEY",
NativeDataTypes.SAP.CHAR, "PARAMID", "10", "0", false, "<None>",
SAPScalarInputValueType.NONE);
        func2.addSAPFunctionScalarInput(fld4);
        SAPStructureField fld5 = new SAPStructureField("/INFATRAN/ZPMDATATYPE-CHAR1",
NativeDataTypes.SAP.CHAR, "MODE", "1", "0", true, "' '",
SAPScalarInputValueType.DEFAULT);
        func2.addSAPFunctionScalarInput(fld5);
        SAPStructureField fld6 = new SAPStructureField("/INFATRAN/ZPRAMS-STRF2",
NativeDataTypes.SAP.INT4, "PACKAGESIZE", "10", "0", true, "<None>",
SAPScalarInputValueType.NONE);
        func2.addSAPFunctionScalarInput(fld6);
        SAPStructureField fld7 = new SAPStructureField("/INFATRAN/ZPRAMS-STRF2",
NativeDataTypes.SAP.INT4, "NUMOFROWS", "10", "0", true, "<None>",
SAPScalarInputValueType.NONE);
        func2.addSAPFunctionScalarInput(fld7);

        SAPStructureInstance sapStrucinst1 = new SAPStructureInstance(sapStruc2,
"SELECTFIELDLIST");
        func2.addSAPFunctionScalarTable(sapStrucinst1);
        SAPStructureInstance sapStrucinst2 = new SAPStructureInstance(sapStruc3,
"SELECTFIELDDETAIL");
        func2.addSAPFunctionScalarTable(sapStrucinst2);
        SAPStructureInstance sapStrucinst3 = new SAPStructureInstance(sapStruc2,
"ORDERFIELDLIST");
        func2.addSAPFunctionScalarTable(sapStrucinst3);

        dsq.addSAPFunction(func2);

        /**
         * SAPFunction with Table and Changing
         */
        SAPFunction func4 = new SAPFunction("ZTEMP_TABPARAM", "TABPARAM");

        SAPStructureInstance sapStrucinst4 = new SAPStructureInstance(sapStruc4,
"ZTABONE");
        func4.addSAPFunctionScalarTable(sapStrucinst4);
        SAPStructureInstance sapStrucinst5 = new SAPStructureInstance(sapStruc4,
"TEMPCHANGE");
        func4.addSAPFunctionScalarChanging(sapStrucinst5);
        dsq.addSAPFunction(func4);

        /**
         * SAP Function containing scalar input and scalar output.
         */
        SAPFunction func5 = new SAPFunction("ZCHAR_UNISCALAR", "RFC for unicode
testing");
        SAPStructureField func5fld1 = new SAPStructureField("ZCHAR_UNI-FKEY",
NativeDataTypes.SAP.CHAR, "FKEY_OUT", "10", "0", false);
        func5.addSAPFunctionScalarOutput(func5fld1);
        SAPStructureField func5fld2 = new SAPStructureField("ZZCHAR_UNI-FCHAR",
NativeDataTypes.SAP.CHAR, "FCHAR_OUT", "255", "0", false);
        func5.addSAPFunctionScalarOutput(func5fld2);
        SAPStructureField func5fld3 = new SAPStructureField("ZCHAR_UNI-FKEY",
NativeDataTypes.SAP.CHAR, "FKEYTYPE_OUT", "10", "0", false);
        func5.addSAPFunctionScalarOutput(func5fld3);
        SAPStructureField func5fld4 = new SAPStructureField("ZCHAR_UNI-FCHAR",
NativeDataTypes.SAP.CHAR, "FCHARTYPE_OUT", "255", "0", false);
        func5.addSAPFunctionScalarOutput(func5fld4);

        SAPStructureField func5fld5 = new SAPStructureField("ZCHAR_UNI-FKEY",
NativeDataTypes.SAP.CHAR, "FKEY_IN", "10", "0", false, StringConstants.NONETAG,
SAPScalarInputValueType.NONE);
        func5.addSAPFunctionScalarInput(func5fld5);
        SAPStructureField func5fld6 = new SAPStructureField("ZCHAR_UNI-FCHAR",
NativeDataTypes.SAP.CHAR, "FCHAR_IN", "255", "0", false, StringConstants.NONETAG,
SAPScalarInputValueType.NONE);
        func5.addSAPFunctionScalarInput(func5fld6);
        SAPStructureField func5fld7 = new SAPStructureField("ZCHAR_UNI-FKEY",
NativeDataTypes.SAP.CHAR, "FKEYTYPE_IN", "10", "0", false, StringConstants.NONETAG,
SAPScalarInputValueType.NONE);
        func5.addSAPFunctionScalarInput(func5fld7);
        SAPStructureField func5fld8 = new SAPStructureField("ZCHAR_UNI-FCHAR",
NativeDataTypes.SAP.CHAR, "FCHARTYPE_IN", "255", "0", false, StringConstants.NONETAG,
SAPScalarInputValueType.NONE);
        func5.addSAPFunctionScalarInput(func5fld8);

        dsq.addSAPFunction(func5);

        RowSet dsqRowSet = dsq.apply().getRowSets().get(0);

        // create an expression transformation with a output port "AVG_SALARY".,
        // of type decimal, which provides average of MIN_SALARY and MAX_SALARY
        String expr = "String (10,0)AVG_SALARY = PINCR";
        TransformField outField = new TransformField(expr);
        //Create expression transformation
        List<TransformField> fields = new ArrayList<TransformField>();
        fields.add(outField);
        RowSet exprRS = (RowSet) helper.expression(dsqRowSet, fields,
                "exp_transform").getRowSets().get(0);

        // write to target
        mapping.writeTarget(exprRS, this.fileTgt);
        folder.addMapping(mapping);
    }

    protected void createSAPStructures()
    {
        sapStruc2 = new SAPStructure("/INFATRAN/ZPMSELFLDLIST");
        SAPStructureField sapStruc2fld1 = new SAPStructureField("CONTINUATION",
NativeDataTypes.SAP.CHAR, "1", "0");
        sapStruc2.addSAPStructureFields(sapStruc2fld1);
        SAPStructureField sapStruc2fld2 = new SAPStructureField("COLUMNLIST",
NativeDataTypes.SAP.CHAR, "1024", "0");
        sapStruc2.addSAPStructureFields(sapStruc2fld2);

        sapStruc3 = new SAPStructure("/INFATRAN/ZPMSELFLDDETAIL");
        SAPStructureField sapStruc3fld1 = new SAPStructureField("TABNAME",
NativeDataTypes.SAP.CHAR, "32", "0");
        sapStruc3.addSAPStructureFields(sapStruc3fld1);
        SAPStructureField sapStruc3fld2 = new SAPStructureField("FIELDNAME",
NativeDataTypes.SAP.CHAR, "32", "0");
        sapStruc3.addSAPStructureFields(sapStruc3fld2);
        SAPStructureField sapStruc3fld3 = new SAPStructureField("LENG",
NativeDataTypes.SAP.NUMC, "6", "0");
        sapStruc3.addSAPStructureFields(sapStruc3fld3);
        SAPStructureField sapStruc3fld4 = new SAPStructureField("INTLEN",
NativeDataTypes.SAP.NUMC, "6", "0");
        sapStruc3.addSAPStructureFields(sapStruc3fld4);
        SAPStructureField sapStruc3fld5 = new SAPStructureField("DECIMALS",
NativeDataTypes.SAP.NUMC, "6", "0");
        sapStruc3.addSAPStructureFields(sapStruc3fld5);
        SAPStructureField sapStruc3fld6 = new SAPStructureField("DATATYPE",
NativeDataTypes.SAP.CHAR, "4", "0");
        sapStruc3.addSAPStructureFields(sapStruc3fld6);
        SAPStructureField sapStruc3fld7 = new SAPStructureField("FLDOFFSET",
NativeDataTypes.SAP.NUMC, "6", "0");
        sapStruc3.addSAPStructureFields(sapStruc3fld7);
        SAPStructureField sapStruc3fld8 = new SAPStructureField("INTTYPE",
NativeDataTypes.SAP.CHAR, "1", "0");
        sapStruc3.addSAPStructureFields(sapStruc3fld8);
        SAPStructureField sapStruc3fld9 = new SAPStructureField("FLDALIAS",
NativeDataTypes.SAP.CHAR, "64", "0");
        sapStruc3.addSAPStructureFields(sapStruc3fld9);

        sapStruc4 = new SAPStructure("ZTABONE");
        SAPStructureField sapStruc4fld1 = new SAPStructureField("FIELD1",
NativeDataTypes.SAP.INT4, "10", "0");
        sapStruc4.addSAPStructureFields(sapStruc4fld1);
        SAPStructureField sapStruc4fld2 = new SAPStructureField("FIELD2",
NativeDataTypes.SAP.CHAR, "100", "0");
        sapStruc4.addSAPStructureFields(sapStruc4fld2);
        SAPStructureField sapStruc4fld3 = new SAPStructureField("FIELD3",
NativeDataTypes.SAP.CHAR, "100", "0");
        sapStruc4.addSAPStructureFields(sapStruc4fld3);
        SAPStructureField sapStruc4fld4 = new SAPStructureField("FIELD4",
NativeDataTypes.SAP.CHAR, "100", "0");
        sapStruc4.addSAPStructureFields(sapStruc4fld4);
    }

    /*
     * Create a session
     */
    protected void createSession() throws Exception {
        session = new Session("SampleSAPSession", "SampleSAPSession",
                "SAP session with sap source");
        session.setMapping(mapping);

        ConnectionInfo connInfo = getRelationalConnectionInfo(SourceTargetType.SAP_R3);


connInfo.getConnProps().setProperty( ConnectionPropsConstants.SESSION_EXTENSION_NAME,
StringConstants.SAP_STAGING_READER );

        //SAP ASQ Session level properties

connInfo.getConnProps().setProperty( ConnectionPropsConstants.STAGE_FILE_DIRECTORY,
"STAGE_FILE_DIRECTORY" );

connInfo.getConnProps().setProperty(ConnectionPropsConstants.SOURCE_FILE_DIRECTORY,"SOURC
E_FILE_DIRECTORY");
        connInfo.getConnProps().setProperty( ConnectionPropsConstants.STAGE_FILE_NAME,
"asq_ekko" );

connInfo.getConnProps().setProperty( ConnectionPropsConstants.REINITIALIZE_THE_STAGE_FILE
, "YES" );

connInfo.getConnProps().setProperty( ConnectionPropsConstants.PERSIST_THE_STAGE_FILE,
"YES" );

connInfo.getConnProps().setProperty( ConnectionPropsConstants.RUN_SESSION_IN_BACKGROUND,
"YES" );
```

```
        session.addConnectionInfoObject(dsq, connInfo);

    }

    /*
     * Create workflow using SAP session
     */
    protected void createWorkflow() throws Exception {
        workflow = new Workflow("WF_Sample_SAP_Workflow",
                "WF_Sample_SAP_Workflow", "Workflow for sap session");
        workflow.addSession(session);
        folder.addWorkFlow(workflow);
    }

    /*
     * Sample program
     */
    public static void main(String[] args) {
        try {
            SAPMappingExample sapMapping = new SAPMappingExample();
            if (args.length > 0) {
                if (sapMapping.validateRunMode(args[0])) {
                    sapMapping.execute();
                }
            } else {
                sapMapping.printUsage();
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Exception is: " + e.getMessage());
        }
    }
}
```

## Connecting to an SAP Source with SAP Functions

The following sample code shows how to use an SAP source with SAP functions in a mapping:

```
package com.informatica.powercenter.sdk.mapfwk.samples;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionPropsConstants;
import com.informatica.powercenter.sdk.mapfwk.connection.SourceTargetType;
import com.informatica.powercenter.sdk.mapfwk.core.ABAPProgram;
import com.informatica.powercenter.sdk.mapfwk.core.DSQTransformation;
import com.informatica.powercenter.sdk.mapfwk.core.MapFwkOutputContext;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.NativeDataTypes;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.SAPASQTransformation;
import com.informatica.powercenter.sdk.mapfwk.core.SAPFunction;
import com.informatica.powercenter.sdk.mapfwk.core.SAPProgramFlowCodeBlock;
import com.informatica.powercenter.sdk.mapfwk.core.SAPProgramFlowSource;
import com.informatica.powercenter.sdk.mapfwk.core.SAPScalarInputValueType;
import com.informatica.powercenter.sdk.mapfwk.core.SAPStructure;
import com.informatica.powercenter.sdk.mapfwk.core.SAPStructureField;
import com.informatica.powercenter.sdk.mapfwk.core.SAPStructureInstance;
import com.informatica.powercenter.sdk.mapfwk.core.SAPVariable;
import com.informatica.powercenter.sdk.mapfwk.core.SAPVariableCategory;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.Source;
import com.informatica.powercenter.sdk.mapfwk.core.StringConstants;
import com.informatica.powercenter.sdk.mapfwk.core.Target;
import com.informatica.powercenter.sdk.mapfwk.core.TransformField;
import com.informatica.powercenter.sdk.mapfwk.core.TransformHelper;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;

/**
 * This is a sample program that demonstrates how to create a mapping
 * with an SAP source and SAP application source qualifier.
 *
 * This sample program shows how to generate and install an ABAP program.
 * The SAP functions and variables that you create for the ABAP program must
 * match the functions and variables defined in the SAP server.
 * If the names and values for the SAP functions do not match
 * the names and values in the SAP server, the generated ABAP program
 * will have incorrect values and will not extract data from the SAP server
 * correctly. This will cause the workflow to fail.
 *
 * 1. Create a mapping with following structure:
 * SAP source --> ASQ --> Expr Transform --> Flatfile Target
 *
 * 2. Create SAP functions and structures.
 *
 * 3. Create a session for the mapping.
 *
 * 4. Create workflow using the session created in step 2.
 *
 */
public class SAPMappingWithFunctions extends Base {

    private Mapping mapping = null;
    private Source sapSrcA004 = null;
    private Target fileTgt = null;
    private SAPASQTransformation dsq;

    /*
     * Create a SAP source
     */
    protected void createSources() {
        sapSrcA004 = this.createSAPekkoSource("in23sapec6");
        folder.addSource(sapSrcA004);
    }

    /*
     * Create a Flatfile target
     */
    protected void createTargets() {
        fileTgt = this.createFlatFileTarget("Output1");
    }

    /*
     * Create a mapping with the following structure
  * SAP Source --> ASQ --> Flatfile
     * Target
     */
    protected void createMappings() throws Exception {

        // create a mapping object
        mapping = new Mapping("MAP_SAMPLE", "MAP_SAMPLE", "Mapping to test mapping with 
an SAP Source");
        setMapFileName(mapping);

        // create transform helper
        TransformHelper helper = new TransformHelper(mapping);

        // create DSQ
        dsq = (SAPASQTransformation) sapSrcA004.createASQTransform();
        mapping.addTransformation(dsq);

        // Create Structures
        // createSAPStructures();

        SAPVariable var1 = new SAPVariable("var1", SAPVariableCategory.ABAPTYPE, 
NativeDataTypes.SAP.CHAR, "0", "10", "10", true);
        dsq.addSAPVariable(var1);
        /**
         * SAPFunction with Table and Changing
         */
        SAPFunction function = new SAPFunction("ZAHLS_CHECK", "");

        SAPStructureField struc1 = new SAPStructureField("KNB1-ZAHLS123",
NativeDataTypes.SAP.CHAR, "I_ZAHLS", "10", "0", false);
        struc1.setValueType(SAPScalarInputValueType.CONSTANT);
        struc1.setValue("1");

        SAPStructureField struc2 = new SAPStructureField("CHAR",
NativeDataTypes.SAP.CHAR, "I_MSG_HANDLING", "10", "0", false);
        struc2.setValueType(SAPScalarInputValueType.DEFAULT);

        function.addSAPFunctionScalarInput(struc1);
        function.addSAPFunctionScalarInput(struc2);

        dsq.addSAPFunction(function);

        // SAPProgramFlowSource po = new SAPProgramFlowSource(sapSrcA004);
        // po.setDynamicFilter("ddd");

        SAPProgramFlowCodeBlock cb = new SAPProgramFlowCodeBlock("a", "");
        dsq.addProgramFlowObject(cb);

        RowSet dsqRowSet = dsq.apply().getRowSets().get(0);

        // create an expression transformation with a output port "AVG_SALARY".,
        // of type decimal, which provides average of MIN_SALARY and MAX_SALARY
        String expr = "String (10,0)AVG_SALARY = PINCR";
        TransformField outField = new TransformField(expr);
        // Create expression transformation
        List<TransformField> fields = new ArrayList<TransformField>();
        fields.add(outField);
        RowSet expRS = (RowSet) helper.expression(dsqRowSet, fields,
"exp_transform").getRowSets().get(0);

        // write to target
        mapping.writeTarget(expRS, this.fileTgt);
        folder.addMapping(mapping);
    }

    /*
     * Create a session
     */
    protected void createSession() throws Exception {
        session = new Session("SampleSAPSession12", "SampleSAPSession12", "SAP session
with sap source");
        session.setMapping(mapping);

        ConnectionInfo connInfo = getRelationalConnectionInfo(SourceTargetType.SAP_R3);

        ConnectionInfo ftpConnInfo = getFTPConnectionInfo();


connInfo.getConnProps().setProperty(ConnectionPropsConstants.SESSION_EXTENSION_NAME,
StringConstants.SAP_STAGING_READER);

        // SAP ASQ Session level properties

connInfo.getConnProps().setProperty(ConnectionPropsConstants.STAGE_FILE_DIRECTORY,
"STAGE_FILE_DIRECTORY");

connInfo.getConnProps().setProperty(ConnectionPropsConstants.SOURCE_FILE_DIRECTORY,
"SOURCE_FILE_DIRECTORY");
        connInfo.getConnProps().setProperty(ConnectionPropsConstants.STAGE_FILE_NAME,
"asq_ekko");

connInfo.getConnProps().setProperty(ConnectionPropsConstants.REINITIALIZE_THE_STAGE_FILE,
 "YES");
```

```
connInfo.getConnProps().setProperty(ConnectionPropsConstants.PERSIST_THE_STAGE_FILE,
"YES");

connInfo.getConnProps().setProperty(ConnectionPropsConstants.RUN_SESSION_IN_BACKGROUND,
"YES");
        session.addConnectionInfoObject(dsq, connInfo);
        session.addConnectionInfoObject(dsq, ftpConnInfo);
    }

    /*
     * Create workflow using SAP session
     */
    protected void createWorkflow() throws Exception {
        workflow = new Workflow("WF_Sample_SAP_Workflow2", "WF_Sample_SAP_Workflow2",
"Workflow for sap session2");

        workflow.addSession(session);
        folder.addWorkFlow(workflow);
    }

    /**
     * This method generates the output xml
     *
     * @throws Exception
     *             exception
     */
    public void generateOutput() throws Exception {
        MapFwkOutputContext outputContext = new
MapFwkOutputContext(MapFwkOutputContext.OUTPUT_FORMAT_XML,
MapFwkOutputContext.OUTPUT_TARGET_FILE, mapFileName);

        try {
            intializeLocalProps();
        } catch (IOException ioExcp) {
            System.err.println(ioExcp.getMessage());
            System.err.println("Error reading pcconfig.properties file.");
            System.err.println("pcconfig.properties file is present in \\javamappingsdk\
\samples directory");
            System.exit(0);
        }

        boolean doImport = false;
        if (runMode == 1)
            doImport = true;
        rep.save(outputContext, doImport);
        System.out.println("Mapping generated in " + mapFileName);

        if (doImport) { // Install ABAP only when xml is imported
            rep.installABAPProgram(new ABAPProgram(mapping.getName()),
StringConstants.SAP_STREAM_MODE);
            // rep.generateABAPProgram(new
            // ABAPProgram(mapping.getBusinessName()),
            // StringConstants.SAP_STREAM_MODE);
        }
    }

    /*
     * Sample program
     */
    public static void main(String[] args) {
        try {
            SAPMappingWithFunctions sapMapping = new SAPMappingWithFunctions();
            if (args.length > 0) {
                if (sapMapping.validateRunMode(args[0])) {
                    sapMapping.execute();
                }
            } else {
                sapMapping.printUsage();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

## Using One Source Qualifier for Multiple Sources

You can use a Source Qualifier transformation to join two sources. The two sources must have a primary key-foreign key relationship.

The following sample code shows how to use one source qualifier to read two related sources:

```
// Logic to create a DSQ Transformation using 2 sources. They
        // should satisfy PKFK constraint.
        List<InputSet> inputSets = new ArrayList<InputSet>();
        InputSet itemIS = new InputSet( itemSource );
        InputSet productIS = new InputSet( productSource );
        inputSets.add( itemIS );
        inputSets.add( productIS );
        TransformationContext tc = new TransformationContext( inputSets );
        Transformation dsqTransform = tc.createTransform( TransformationConstants.DSQ,
"CMN_DSQ" );
        // RowSet of combined transformation
        RowSet dsqRS = (RowSet) dsqTransform.apply().getRowSets().get( 0 );
        mapping.addTransformation( dsqTransform );
        // Create an Expression Transformation
        TransformHelper helper = new TransformHelper( mapping );
        TransformField orderCost = new TransformField(
                "decimal(24,0) OrderCost = (Price*Wholesale_cost)" );
        RowSet expRowSet = (RowSet) helper.expression( dsqRS, orderCost,
"comb_exp_transform" )
                .getRowSets().get( 0 ); // target
        // write to target
        mapping.writeTarget( expRowSet, outputTarget );
```

## Using One Source Qualifier for Multiple SAP Sources

The following sample code shows how to use one source qualifier to read two SAP sources:

```
package com.informatica.powercenter.sdk.mapfwk.samples;

import java.util.ArrayList;
import java.util.List;

import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.core.Field;
import com.informatica.powercenter.sdk.mapfwk.core.FieldKeyType;
import com.informatica.powercenter.sdk.mapfwk.core.InputSet;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.SAPASQTransformation;
import com.informatica.powercenter.sdk.mapfwk.core.SAPProgramFlowCodeBlock;
import com.informatica.powercenter.sdk.mapfwk.core.SAPProgramFlowSource;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.Source;
import com.informatica.powercenter.sdk.mapfwk.core.Target;
import com.informatica.powercenter.sdk.mapfwk.core.TransformationConstants;
import com.informatica.powercenter.sdk.mapfwk.core.TransformationContext;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;

/**
 * This is a sample program that demonstrates how to create a mapping
 * with two SAP sources and a common SAP application source qualifier.
 *
 * 1. Create a mapping with following structure:
 * (SAP source1 , SAP source2) --> ASQ --> Flatfile Target
 *
 * 2. Create a session for the mapping.
 *
 * 3. Create workflow using the session created in step 2.
 *
 */
public class SAPCommonASQExample extends Base {

    protected Target outputTarget;
    protected Source ekkoSource;
    protected Source ekpoSource;
    SAPASQTransformation asqTransform;

    /*
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSources()
     */
    protected void createSources() {
        ekpoSource = this.createSAPekpoSource("in23sapec6");
        folder.addSource(ekpoSource);
        ekkoSource = this.createSAPekkoSource("in23sapec6");
        folder.addSource(ekkoSource);
    }

    /*
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createTargets()
     */
    protected void createTargets() {
        outputTarget = this.createFlatFileTarget("CMN_DSQ_Output");
    }

    /*
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createMappings()
     */
    protected void createMappings() throws Exception {
        mapping = new Mapping("CMN_SAP_ASQ", "CMN_SAP_ASQ", "This is CMN_ASQ sample");
        setMapFileName(mapping);
        // Logic to create a ASQ Transformation using 2 sources. They
        // should satisfy PKFK constraint or a join override.
        List<InputSet> inputSets = new ArrayList<InputSet>();
        InputSet ekkoIS = new InputSet(ekkoSource);
        Field ebelnFld = ekkoSource.getField("BUKRS");
        ebelnFld.setFieldKeyType(FieldKeyType.FOREIGN_KEY);
        ebelnFld.setReferenceContraint(ekpoSource.getName(), ekpoSource.getDBName(),
"BUKRS");
        InputSet ekpoIS = new InputSet(ekpoSource);
        inputSets.add(ekkoIS);
        inputSets.add(ekpoIS);
        TransformationContext tc = new TransformationContext(inputSets);

        asqTransform = (SAPASQTransformation)
tc.createTransform(TransformationConstants.ASQ_SAP, "ASQ_CMN_SAP");

        // ASQ will create the default program flow objects for sources. To
        // apply any program flow object properties,
        // fetch the objects and apply. Sample shown below
        SAPProgramFlowSource pfSrc = (SAPProgramFlowSource)
asqTransform.getProgramFlowObject("EKKO");
        pfSrc.setDynamicFilter("EKKO.BUKRS = \"XYZ\"");

        // create program code blocks
        SAPProgramFlowCodeBlock scblock = new SAPProgramFlowCodeBlock("codeblock1",
"This is first code block");
        SAPProgramFlowCodeBlock scblock2 = new SAPProgramFlowCodeBlock("codeblock2",
"This is last code block");
        // You can order the program objects in the list
        asqTransform.getProgramFlowObjects().add(0, scblock);
        asqTransform.addProgramFlowObject(scblock2);

        // RowSet of combined transformation
        RowSet dsqRS = (RowSet) asqTransform.apply().getRowSets().get(0);
        mapping.addTransformation(asqTransform);
        // write to target
        mapping.writeTarget(dsqRS, outputTarget);
        // add mapping to folder
        folder.addMapping(mapping);
    }

    /*
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createWorkflow()
     */
    protected void createWorkflow() throws Exception {
        workflow = new Workflow("Workflow_for_CMN_ASQ", "Workflow_for_CMN_DSQ", "This
workflow for joiner");
        workflow.addSession(session);
        folder.addWorkFlow(workflow);
    }

    public static void main(String args[]) {
        try {
            SAPCommonASQExample commonASQExample = new SAPCommonASQExample();
            if (args.length > 0) {
                if (commonASQExample.validateRunMode(args[0])) {
                    commonASQExample.execute();
                }
            } else {
                commonASQExample.printUsage();
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Exception is: " + e.getMessage());
        }
    }

    /*
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSession()
     */
    protected void createSession() throws Exception {
        session = new Session("Session_For_CMN_ASQ", "Session_For_CMN_ASQ", "This is
session for CMN_ASQ");
        session.setMapping(this.mapping);
    }
}
```

# Transformations

The following code examples show how to create different types of transformations.

## Aggregator

```
// create an aggregator Transformation
// calculate cost per order using the formula
// SUM((UnitPrice * Quantity) * (100 - Discount1) / 100) grouped by
// OrderId
TransformField cost = new TransformField(
        "decimal(15,0) total_cost = (SUM((UnitPrice * Quantity) * (100 - Discount) /
100))");
RowSet aggRS = (RowSet) helper.aggregate(dsqRS, cost,
        new String[] { "OrderID" }, "agg_transform").getRowSets().get(0);
// write to target
mapping.writeTarget(aggRS, this.outputTarget);
```

## Data Masking

```
//creating a mapping with Data Masking transformation.
protected void createMappings() throws Exception {
    mapping = new Mapping("DMO003", "DataMaskTransformationTest", "Mapping for
DataMaskTransformationTest");
    setMapFileName(mapping);
    TransformHelper helper = new TransformHelper(mapping);
    RowSet dsqRowSet1 = (RowSet)
helper.sourceQualifier(this.jobSourceObj).getRowSets().get(0);
    InputSet ip = new InputSet(dsqRowSet1,
PortPropagationContextFactory.getContextForAllIncludeCols(),
            PortLinkContextFactory.getPortLinkContextByName());
    RowSet datamaskRS = (RowSet) helper.datamaskTransformation(ip, null, "DMO003", null,
            CodePageConstants.META_EXTENTION_CODE_PAGE_EN).getRowSets().get(0);
    dmo = (DataMaskTransformation) mapping.getTransformation("DMO003");
    setDataToTransformation();
    mapping.writeTarget(datamaskRS, this.targetObj);
    folder.addMapping(mapping);
}

//Setting various mask options.
protected void setDataToTransformation() throws Exception {
    dmo.addPortwithSeed(MaskConstants.DATA_TYPE_STRING, "Key", "JOB_ID", "10", "0",
"190", "", "3");
    dmo.addPortwithSeed(MaskConstants.DATA_TYPE_STRING, "SSN", "JOB_TITLE", "35", "0",
"", "", "10");
    dmo.addPort(MaskConstants.DATA_TYPE_DECIMAL, "Random", "MIN_SALARY", "6", "0", "",
"1");
    dmo.addKeyMaskForStringByPortName("JOB_ID", "FALSE", "Mask only", "FALSE", "Use
only", "FALSE", "", "", "");
    dmo.addSSNMaskForStringByPortName("JOB_TITLE", "FALSe", "FALSE");
    DataMask.MaskNumeric maskNumeric = new DataMask.MaskNumeric("Fixed", "FALSE",
"TRUE", "", "", "", "1000000.000000", "0");
    dmo.addRandomMaskForDecimalByPortName("MIN_SALARY", maskNumeric);

    dmo.setMetaExtensionValue(StringConstants.META_EXTENTION_MASKING_RULES,
dmo.getPortinfo().getXml());
}
```

## Email Data Masking

```
    //creating a mapping with Email Data Masking transformation.
    protected void createMappings() throws Exception {
        mapping = new Mapping("EmailDMOTx", "Email_DataMaskTransformationTest",
            "Mapping for Email DataMaskTransformationTest");
        setMapFileName(mapping);
        RowSet datamaskRS = createTransformation();
        mapping.writeTarget(datamaskRS, this.targetObj);
        folder.addMapping(mapping);
    }

    //setting various mask options.
    protected void setDataToTransformation() throws Exception {
        dmo.addPortwithSeed(MaskConstants.DATA_TYPE_STRING,
MaskConstants.MASKING_OPTION_EMAIL_ADDRESS,
        "JOB_TITLE", "35", "0", "$$Data", "", "7");
    /*
     * creating a mapping variable in the repository for this mapping
     */
        MappingVariable mappingVar = new MappingVariable( "MAX",
MappingVariableDataTypes.INTEGER,
            "0", "mapping variable example", true, "$$Data", "10", "0", true );
        mapping.addMappingVariable( mappingVar );
        List<String> columns = new ArrayList<String>();
        columns.add("FNAME");
        columns.add("LNAME");
        columns.add("SALARY");
        columns.add("DOMAINTYPE");
    /*
     * DMO Email transformation with flat file constant domain
     */
        EmailDictionary.FlatFileDomainDictionary flatfileDomainDictionary =
            new EmailDictionary.FlatFileDomainDictionary("2252", "firstname.dic",
"FNAME", columns);
        EmailDictionary.RelationalDomainDictionary relationalDomainDictionary =
            new EmailDictionary.RelationalDomainDictionary("FNAME", "sqlserver",
"Administrator",
            "admin", "admin", "firstname.dic", columns);
        EmailDictionary.FlatFilePortDictionary flatfilePortDictionary =
            new EmailDictionary.FlatFilePortDictionary("2252", "firstname.dic",
"ROUND(5)", columns);
        EmailDictionary.RelationalPortDictionary relationalPortDictionary =
            new EmailDictionary.RelationalPortDictionary("sqlserver", "Administrator",
            "admin", "admin", "firstname.dic", "ROUND(5)", columns);
    /*
     * From dictionary with relational port dictionary and relational domain dictionary
     */
        // dmo.addEMailMaskFromDictionary("JOB_TITLE", "", "1",
        // EmailDictionary.DomainType.RANDOM, "FNAME", "10", "TRUE", "FALSE", "FNAME",
        // "10",relationalPortDictionary,relationalDomainDictionary);
    /*
     * From dictionary with flat file port dictionary and relational domain dictionary
     */
        // dmo.addEMailMaskFromDictionary("JOB_TITLE", "", "1",
        // EmailDictionary.DomainType.RANDOM, "FNAME", "10", "TRUE", "FALSE", "FNAME",
        // "10",flatfilePortDictionary,relationalDomainDictionary);

    /*
     * From dictionary with flat file port dictionary and flat file domain dictionary
     */
        // dmo.addEMailMaskFromDictionary("JOB_TITLE", "", "1",
EmailDictionary.DomainType.RANDOM,
        // "FNAME", "10", "TRUE", "FALSE", "FNAME",
        // "10",flatfilePortDictionary,flatfileDomainDictionary);

    /*
     * From mapping with relational domain dictionary
     */
        // dmo.addEmailMaskFromMapping("JOB_TITLE", "", "1",
EmailDictionary.DomainType.RANDOM,
        // "out_JOB_ID", "10", "TRUE", "FALSE", "out_JOB_ID", "10",
relationalDomainDictionary);
    /*
     * From mapping with constant domain
     */
        // dmo.addEmailMaskFromMapping("JOB_TITLE", "gmail.com", "1",
        // EmailDictionary.DomainType.CONSTANT, "out_JOB_ID", "10",
        // "TRUE", "FALSE", "out_JOB_ID", "10", null);

    /*
     * Standard email masking
     */
        dmo.addStandardEmailMask("JOB_TITLE", "TRUE", "TRUE");
        dmo.setMetaExtensionValue(StringConstants.META_EXTENTION_MASKING_RULES,
dmo.getPortinfo().getXml());
        }
```

## Expression

```
// create an expression Transformation
        // the fields LastName and FirstName are concataneted to produce a new
        // field fullName
        String expr = "string(80, 0) fullName= firstName || lastName";
        TransformField outField = new TransformField( expr );
        String expr2 = "integer(10,0) YEAR_out=IIF(EmployeeID=0,2000,2001)";
        TransformField outField2 = new TransformField( expr2 );
        List<TransformField> transFields = new ArrayList<TransformField>();
        transFields.add( outField );
        transFields.add( outField2 );
        RowSet expRS = (RowSet) helper.expression( dsqRS, transFields,
"exp_transform" ).getRowSets()
                .get( 0 );
        // write to target
        mapping.writeTarget( expRS, outputTarget );
```

## Filter

```
// create a mapping
mapping = new Mapping( "SorterMapping", "mapping", "Testing Sorter sample" );
setMapFileName( mapping );
TransformHelper helper = new TransformHelper( mapping );
// create a Filter Transformation
// Filter out rows that don't belong to USA
RowSet filterRS = (RowSet) helper.filter( dsqRS, "Country = 'India'",
"filter_transform" ).getRowSets().get( 0 );
```

## Router

```
TransformHelper helper = new TransformHelper( mapping );
        // Create a TransformGroup
        List<TransformGroup> transformGrps = new ArrayList<TransformGroup>();
        TransformGroup transGrp = new TransformGroup( "LONDON_GROUP", "City =
'London'" );
        transformGrps.add( transGrp );
        transGrp = new TransformGroup( "SEATTLE_GROUP", "City = 'Seattle'" );
        transformGrps.add( transGrp );
        // creating DSQ Transformation
        OutputSet itemOSet = helper.sourceQualifier( employeeSrc );
        RowSet employeeRowSet = (RowSet) itemOSet.getRowSets().get( 0 );
        // create a Router Transformation
        OutputSet routerOutputSet = helper.router( employeeRowSet, transformGrps,
                "Router_transform" );
        // write to target
        RowSet outRS = routerOutputSet.getRowSet( "LONDON_GROUP" );
        if (outRS != null)
            mapping.writeTarget( outRS, londonOutputTarget );
        outRS = routerOutputSet.getRowSet( "SEATTLE_GROUP" );
        if (outRS != null)
            mapping.writeTarget( outRS, seattleOutputTarget );
        outRS = routerOutputSet.getRowSet( "DEFAULT" );
        if (outRS != null)
            mapping.writeTarget( outRS, defaultOutputTarget );
```

## Sequence Generator

```
// create a Sequence Generator Transformation
RowSet seqGenRS = (RowSet) helper.sequenceGenerator( "sequencegenerator_transform" )
                .getRowSets().get( 0 );
List<InputSet> vinSets = new ArrayList<InputSet>();
vinSets.add( new InputSet( dsqRS ) );
vinSets.add( new InputSet( seqGenRS ) );
// write to target
mapping.writeTarget( vinSets, outputTarget );
```

## Sorter

```
// create a sorter Transformation
RowSet sorterRS = (RowSet) helper.sorter( dsqRS, new String[] { "FirstName",
"LastName" },
new boolean[] { true, false }, "sorter_transform" ).getRowSets().get( 0 );
```

## SQL Transformation

```
List<TransformField> transformFields = new ArrayList<TransformField>();
Field field1 = new
Field("NewField1","NewField1","",TransformationDataTypes.STRING,"20","0",
FieldKeyType.NOT_A_KEY,FieldType.TRANSFORM,false);
field1.setAttributeValues(SQLTransformation.ATTR_SQLT_PORT_ATTRIBUTE, new
Integer(2));
```

```
field1.setAttributeValues(SQLTransformation.ATTR_SQLT_PORT_NATIVE_TYPE,"char");
TransformField tField1 = new TransformField(field1,PortType.OUTPUT);

Field field2 = new
Field("NewField2","NewField2","NewField2",TransformationDataTypes.STRING,"20","0",
FieldKeyType.NOT_A_KEY,FieldType.TRANSFORM,false);
field2.setAttributeValues(SQLTransformation.ATTR_SQLT_PORT_ATTRIBUTE, new
Integer(2));
field2.setAttributeValues(SQLTransformation.ATTR_SQLT_PORT_NATIVE_TYPE,"char");
TransformField tField2 = new TransformField(field2,PortType.INPUT);

Field field3 = new
Field("NewField3","NewField3","NewField3",TransformationDataTypes.STRING,"20","0",
FieldKeyType.NOT_A_KEY,FieldType.TRANSFORM,false);
field3.setAttributeValues(SQLTransformation.ATTR_SQLT_PORT_ATTRIBUTE, new
Integer(2));
field3.setAttributeValues(SQLTransformation.ATTR_SQLT_PORT_NATIVE_TYPE,"char");
TransformField tField3 = new TransformField(field3,PortType.OUTPUT);

transformFields.add(tField1);
transformFields.add(tField2);
transformFields.add(tField3);

InputSet ip = new InputSet
(dsqRS,PortPropagationContextFactory.getContextForAllIncludeCols(),PortLinkContextFactory
.getPortLinkContextByName());
RowSet sqlRS = (RowSet) helper.sqlTransformation(ip, transformFields, false,
SourceTargetType.Oracle, false, false, "SqlTransformation", null).getRowSets().get(0);

mapping.writeTarget(sqlRS, this.outputTarget);
```

## Stored Procedure

```
// create a stored procedure transformation
        List<TransformField> transformFields = new ArrayList<TransformField>();
        Field field1 = new Field( "RetValue", "RetValue", "This is return value",
        TransformationDataTypes.INTEGER, "10", "0", FieldKeyType.NOT_A_KEY,
                FieldType.TRANSFORM, false );
        TransformField tField1 = new TransformField( field1,PortType.RETURN_OUTPUT );
        transformFields.add( tField1 );
        Field field2 = new Field( "nID1", "nID1", "This is the ID field",
        TransformationDataTypes.INTEGER, "10", "0", FieldKeyType.NOT_A_KEY,
                FieldType.TRANSFORM, false );
        TransformField tField2 = new TransformField( field2, PortType.INPUT );
        // transformFields.add( tField2 );
        Field field3 = new Field( "outVar", "outVar", "This is the Output field",
        TransformationDataTypes.STRING, "20", "0", FieldKeyType.NOT_A_KEY,
                FieldType.TRANSFORM, false );
        TransformField tField3 = new TransformField( field3, PortType.INPUT_OUTPUT );
        transformFields.add( tField3 );
        java.util.Hashtable<Field,Object> link = new java.util.Hashtable<Field,Object>();
        link.put( dsqRS.getField( "ItemId" ), field2 );
        PortLinkContext linkContext =
PortLinkContextFactory.getPortLinkContextByMap( link );
        RowSet storedProcRS = (RowSet) helper.storedProc( new InputSet( dsqRS,
linkContext ),
                transformFields, "SampleStoredProc", "Sample Stored Procedure
Transformation" )
                .getRowSets().get( 0 );
        // write to target
        mapping.writeTarget( storedProcRS, this.outputTarget );
```

## Transaction Control

```
// create an Transaction Control Transformation
String condition = "IIF(EmployeeID>10,TC_COMMIT_AFTER,TC_CONTINUE_TRANSACTION)";
```

```
RowSet tcRS = (RowSet) helper.transactionControl( dsqRS, null, "tc_transform", condition,
null ).getRowSets().get( 0 );
```

## Unconnected Lookup

```
// create an unconnected lookup transformation
        // the fields LastName and FirstName are concataneted to produce a new
        // field fullName
        String expr = "string(80, 0) firstName1= IIF(ISNULL(:LKP.lkp(ItemId,
Item_Name)), "
                + "DD_UPDATE, DD_REJECT)";
        TransformField outField = new TransformField( expr );
        RowSet expRS = (RowSet) helper.expression( dsqRS, outField,
"exp_transform" ).getRowSets()
                .get( 0 );
        // create a unconnected lookup transformation
        // set the return port
        Field retField = new Field( "Item_No1", "Item_No", "",
TransformationDataTypes.INTEGER, "10",
                "0", FieldKeyType.PRIMARY_KEY, FieldType.TRANSFORM, false );
        // create input and output fields
        List<Field> input = new ArrayList<Field>();
        List<Field> output = new ArrayList<Field>();
        createUncLkpFields( input, output );
        // create an unconnected lookup
        String condition = "ItemId = EmployeeID";
        UnconnectedLookup uncLkp = helper.unconnectedLookup( "lkp", retField, input,
condition,
                employeeSrc );
        uncLkp.setPortType( "EmployeeID", PortType.LOOKUP );
        mapping.addTransformation(uncLkp);
        // write to target
        mapping.writeTarget( expRS, outputTarget );
```

## Union

```
 RowSet rsGroupFld = new RowSet();
        Field field1 = new Field( "ItemId", "ItemId", "",
TransformationDataTypes.INTEGER, "10", "0",
                FieldKeyType.NOT_A_KEY, FieldType.TRANSFORM, false );
        rsGroupFld.addField( field1 );
        Field field2 = new Field( "Item_Name", "Item_Name", "",
TransformationDataTypes.STRING, "72",
                "0", FieldKeyType.NOT_A_KEY,FieldType.TRANSFORM, false );
        rsGroupFld.addField( field2 );
        Field field3 = new Field( "Item_Price", "Item_Price", "",
TransformationDataTypes.DECIMAL, "10",
                "2", FieldKeyType.NOT_A_KEY, FieldType.TRANSFORM, false );
        rsGroupFld.addField( field3 );
        // creating DSQ Transformation
        OutputSet itemOSet = helper.sourceQualifier( itemsSrc );
        RowSet itemRowSet = (RowSet) itemOSet.getRowSets().get( 0 );
        // itemRowSet.setGroupName("ITEM_GROUP");
        OutputSet productOSet = helper.sourceQualifier( productSrc );
        RowSet productRowSet = (RowSet) productOSet.getRowSets().get( 0 );
        // productRowSet.setGroupName("PRODUCT_GROUP");
        // Port propogation for Items and products
        PortPropagationContext itemRSContext = PortPropagationContextFactory
                .getContextForIncludeCols( new String[] { "ItemId", "Item_Name",
"Price" } );
        PortPropagationContext productRSContext = PortPropagationContextFactory
                .getContextForIncludeCols( new String[] { "Item_No", "Item_Name",
"Cust_Price" } );
        List<InputSet> inputSets = new ArrayList<InputSet>();
        inputSets.add( new InputSet( itemRowSet, itemRSContext ) );
        inputSets.add( new InputSet( productRowSet, productRSContext ) );
        // create a Union Transformation
        RowSet unionRS = (RowSet) helper.union( inputSets, rsGroupFld,
"Union_transform" )
                .getRowSets().get( 0 );
        // write to target
        mapping.writeTarget( unionRS, outputTarget );
```

## Update Strategy

```
// create an update strategy transformation
// Insert only if the city is 'Seattle' else reject it
RowSet filterRS = (RowSet) helper.updateStrategy( dsqRS,
"IIF(City = 'Seattle', DD_INSERT, DD_REJECT )", "updateStrategy_transform" )
.getRowSets().get( 0 );
```

## XML Generator

```
/*
 * Copyright Informatica Corporation.
 */
package com.informatica.powercenter.sdk.mapfwk.samples;

import java.util.ArrayList;
import java.util.List;

import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.connection.SourceTargetType;
import com.informatica.powercenter.sdk.mapfwk.core.Field;
import com.informatica.powercenter.sdk.mapfwk.core.FieldKeyType;
import com.informatica.powercenter.sdk.mapfwk.core.FieldType;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.NativeDataTypes;
import com.informatica.powercenter.sdk.mapfwk.core.OutputSet;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.Source;
import com.informatica.powercenter.sdk.mapfwk.core.Target;
import com.informatica.powercenter.sdk.mapfwk.core.TransformHelper;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;

/**
 * Uses the XML Generator transformation in a mapping.
 *
 */
public class XMLParserGeneratorSample extends Base {
    // ////////////////////////////////////////////////////////////////
    // instance variables
    // ////////////////////////////////////////////////////////////////
    protected Mapping mapping = null;
    protected Source jobSourceObj = null;
    protected Target targetObj = null;

    /*
     * Creates a source.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSources()
     */
    protected void createSources() {
        jobSourceObj = createOracleJobSource("Oracle_Source");
        folder.addSource(jobSourceObj);
    }

    /*
     * Creates a target.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createTargets()
     */
    protected void createTargets() {
        targetObj = this.createRelationalTarget(SourceTargetType.Oracle,
"Oracle_target");
    }

    /*
     * Creates a mapping.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createMappings()
     */
    protected void createMappings() throws Exception {
        // create a mapping object
        mapping = new Mapping("XMLParserGeneratorTransformation",
"XMLParserGeneratorTransformation", "Mapping for XMLParserGeneratorTransformation");
        setMapFileName(mapping);
        TransformHelper helper = new TransformHelper(mapping);
        // create DSQ
        RowSet dsqRowSet = (RowSet)
helper.sourceQualifier(this.jobSourceObj).getRowSets().get(0);      //only one field is
given as input.

        OutputSet outf = (OutputSet) helper.xmlParser(dsqRowSet,
"XML_Parser_Transformation","xsd_samples\\FIN_STRUCT-ZTT_ALL_STRUCT.xsd");

        RowSet outputTarget = (RowSet) outf.getRowSets().get(0);

        OutputSet outGen = (OutputSet) helper.xmlGenerator(outputTarget,
"XML_Generator_Transformation", "xsd_samples\\FIN_STRUCT-ZTT_ALL_STRUCT.xsd");

        RowSet outputGenTarget = (RowSet) outGen.getRowSets().get(0);

        mapping.writeTarget(outputGenTarget, this.targetObj);

        folder.addMapping(mapping);
    }

    /*
     * Creates a session to run the mapping.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSession()
     */
    protected void createSession() throws Exception {
        session = new Session("Session_For_CustomACTIVE", "Session_For_CustomACTIVE",
"This is session for Custom Transformation");
        session.setMapping(mapping);
    }

    /*
     * Creates a workflow.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createWorkflow()
     */
    protected void createWorkflow() throws Exception {
        workflow = new Workflow("Workflow_for_CustomTransformation",
"Workflow_for_CustomTransformation", "This workflow for Custom Transformation");
        workflow.addSession(session);
        folder.addWorkFlow(workflow);
    }

    protected Source createOracleJobSource(String dbName) {
        Source jobSource = null;

        List<Field> fields = new ArrayList<Field>();
        Field jobIDField = new Field("JOB_ID", "JOB_ID", "",
NativeDataTypes.Oracle.VARCHAR2, "10", "0", FieldKeyType.PRIMARY_KEY, FieldType.SOURCE,
true);
        fields.add(jobIDField);

        ConnectionInfo info = getRelationalConnInfo(SourceTargetType.Oracle, dbName);
        jobSource = new Source("JOBS", "JOBS", "This is JOBS table", "JOBS", info);
        jobSource.setFields(fields);
        return jobSource;
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        try {
            XMLParserGeneratorSample customActive = new XMLParserGeneratorSample();
            if (args.length > 0) {
                if (customActive.validateRunMode(args[0])) {
                    customActive.execute();
                }
            } else {
                customActive.printUsage();
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Exception is: " + e.getMessage());
        }
    }
}
```

## XML Parser

```
/*
 * Copyright Informatica Corporation.
 */
package com.informatica.powercenter.sdk.mapfwk.samples;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.connection.SourceTargetType;
import com.informatica.powercenter.sdk.mapfwk.core.Field;
import com.informatica.powercenter.sdk.mapfwk.core.FieldKeyType;
import com.informatica.powercenter.sdk.mapfwk.core.FieldType;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.NativeDataTypes;
import com.informatica.powercenter.sdk.mapfwk.core.OutputSet;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.Source;
import com.informatica.powercenter.sdk.mapfwk.core.Target;
import com.informatica.powercenter.sdk.mapfwk.core.TransformHelper;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;

/**
 *  Uses the XML Parser transformation in a mapping.
 *
 */
public class XMLParserTransformationSample extends Base {
    // /////////////////////////////////////////////////////////////////
    // instance variables
    // /////////////////////////////////////////////////////////////////
    protected Mapping mapping = null;
    protected Source jobSourceObj = null;
    protected Target targetObj = null;

    /*
     * Creates a source.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSources()
     */
    protected void createSources() {
        jobSourceObj = createOracleJobSource("Oracle_Source");
        folder.addSource(jobSourceObj);
    }
    /*
     * Creates a target.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createTargets()
     */
    protected void createTargets() {
        targetObj = this.createRelationalTarget(SourceTargetType.Oracle,
"Oracle_target");
    }

    /*
     * Creates a mapping.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createMappings()
     */
    protected void createMappings() throws Exception {
        // create a mapping object
        mapping = new Mapping("XMLParserTransformation", "XMLParserTransformation",
"Mapping for FDA on JOBS table");
        setMapFileName(mapping);
        TransformHelper helper = new TransformHelper(mapping);
        // create DSQ
        RowSet dsqRowSet = (RowSet)
helper.sourceQualifier(this.jobSourceObj).getRowSets().get(0);      //only one field is
given as input.

        OutputSet outf = (OutputSet) helper.xmlParser(dsqRowSet, "XML_Parser_Transform",
"xsd_samples\\FIN_STRUCT-ZTT_ALL_STRUCT.xsd");

        RowSet outputTarget = (RowSet) outf.getRowSets().get(0);

        mapping.writeTarget(outputTarget, this.targetObj);

        folder.addMapping(mapping);
    }

    /*
     * Creates a session to run the mapping.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSession()
     */
    protected void createSession() throws Exception {
        session = new Session("Session_For_CustomACTIVE", "Session_For_CustomACTIVE",
"This is session for Custom Transformation");
        session.setMapping(mapping);
    }

    /*
     * Creates a workflow.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createWorkflow()
     */
    protected void createWorkflow() throws Exception {
        workflow = new Workflow("Workflow_for_CustomTransformation",
"Workflow_for_CustomTransformation", "This workflow for Custom Transformation");
        workflow.addSession(session);
        folder.addWorkFlow(workflow);
    }

    protected Source createOracleJobSource(String dbName) {
        Source jobSource = null;

        List<Field> fields = new ArrayList<Field>();
        Field jobIDField = new Field("JOB_ID", "JOB_ID", "",
NativeDataTypes.Oracle.VARCHAR2, "10", "0", FieldKeyType.PRIMARY_KEY, FieldType.SOURCE,
true);
        fields.add(jobIDField);

        ConnectionInfo info = getRelationalConnInfo(SourceTargetType.Oracle, dbName);
        jobSource = new Source("JOBS", "JOBS", "This is JOBS table", "JOBS", info);
        jobSource.setFields(fields);
        return jobSource;
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        try {
            XMLParserTransformationSample customActive = new
XMLParserTransformationSample();
            if (args.length > 0) {
                if (customActive.validateRunMode(args[0])) {
                    customActive.execute();
                }
            } else {
                customActive.printUsage();
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Exception is: " + e.getMessage());
        }
    }
}
```

## Creating and Using Reusable Transformations

The following sample code shows how to create a reusable transformation and use it in a mapping. When you import the reusable transformation into the repository, the reusable transformation is saved in the folder. The mapping refers to the transformation in the folder:

```
//create a filter transformation creating this way because we need to set it reusable
FilterTransformation trans=new
FilterTransformation("filter_transform","","","firstFilter",mapping,vInputSet,"TRUE",null
);
trans.setReusable(true);
//creating another instance of reusable transformation not that in both filter
transformation
we are having
//same filtertransformation name we are passsing different instance name
FilterTransformation trans1=new
FilterTransformation("filter_transform","","","secondFilter",mapping,vInputSet1,"TRUE",nu
ll);
trans1.setReusable(true);
// write to target
RowSet filterRS1= (RowSet)trans.apply().getRowSets().get(0);
RowSet filterRS2= (RowSet)trans1.apply().getRowSets().get(0);
/*filterRS1.setName("firstFilter");
filterRS2.setName("secondFilter");*/
mapping.writeTarget( filterRS1, outputTarget );
mapping.writeTarget( filterRS2, outputTarget1 );
```

# XML Sources and Targets

You can use the Design API to create an XML source or XML target definition from an XML schema definition with an entity view or a hierarchical normalized view. You can also access an XML source or target from the PowerCenter repository.

## Creating an XML Definition with an Entity View

The following sample code shows how to import XML definition from an XML schema definition with an entity view:

```
package com.informatica.powercenter.sdk.mapfwk.samples;
```

```
import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.connection.SourceTargetType;
import com.informatica.powercenter.sdk.mapfwk.core.FieldType;
import com.informatica.powercenter.sdk.mapfwk.core.GroupType;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.TransformHelper;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;
import com.informatica.powercenter.sdk.mapfwk.core.xml.EntityView;
import com.informatica.powercenter.sdk.mapfwk.core.xml.XMLSource;
import com.informatica.powercenter.sdk.mapfwk.core.xml.XMLTarget;
import com.informatica.powercenter.sdk.mapfwk.core.xml.XMLView;

public class XMLSourceTargetEntityViewSample extends Base{

        private XMLSource xmlSrc;
        private XMLTarget xmlTgt;
        protected XMLView entityViewSrc = null;
        protected XMLView entityViewTgt = null;

        @Override
        protected void createSources() {
        entityViewSrc = new EntityView("C:\\Venkat\\JMF\\JMF\\eBiz\\main\\javamappingsdk\
\xsd_samples\\FIN_STRUCT-ZTT_ALL_STRUCT.xsd", GroupType.OUTPUT,FieldType.SOURCE);
        xmlSrc = new XMLSource(entityViewSrc,"XMLSourceEntity", "XMLSourceEntity",
            "XMLSourceEntity", "XMLSourceEntity", new
ConnectionInfo(SourceTargetType.XML));
        xmlSrc.createSourceFields();
        folder.addSource(xmlSrc);
        }

        @Override
        protected void createTargets() {
        entityViewTgt = new EntityView("C:\\Venkat\\JMF\\JMF\\eBiz\\main\\javamappingsdk\
\xsd_samples\\FIN_STRUCT-ZTT_ALL_STRUCT.xsd", GroupType.INPUT,FieldType.TARGET);
        xmlTgt = new XMLTarget(entityViewTgt, "XMLTargetEntity", "XMLTargetEntity",
            "XMLTargetEntity", "XMLTargetEntity", new
ConnectionInfo(SourceTargetType.XML));
        xmlTgt.createTargetFields();
        folder.addTarget(xmlTgt);
        }

        @Override
        protected void createMappings() throws Exception {
        // create a mapping object
        mapping = new Mapping("XMLSourceTargetEntityViewMapping",
"XMLSourceTargetEntityViewMapping",
            "XML Source Target Entity Mapping");
        setMapFileName(mapping);
        TransformHelper helper = new TransformHelper(mapping);

        // create DSQ
        RowSet dsqRowSet = (RowSet) helper.sourceQualifier(xmlSrc).getRowSets().get(0);
        mapping.writeTarget(dsqRowSet, this.xmlTgt);
        folder.addMapping(mapping);
        }

        @Override
        protected void createSession() throws Exception {
        session = new Session("Session_For_XMLSourceTargetEntityView",
        "Session_For_XMLSourceTargetEntityView", This is session for
        XML Source / Target Entity");
        session.setMapping(mapping);
        }

        @Override
        protected void createWorkflow() throws Exception {
        workflow = new Workflow("Workflow_for_XMLSourceTargetEntityView",
        "Workflow_for_XMLSourceTargetEntityView", "This workflow for
        XML Source / Target Entity");
        workflow.addSession(session);
        folder.addWorkFlow(workflow);
        }

public static void main(String[] args) {
        try {
                XMLSourceTargetEntityViewSample xmlSourceSample = new
XMLSourceTargetEntityViewSample();
                if (args.length > 0) {
                    if (xmlSourceSample.validateRunMode(args[0])) {
                        xmlSourceSample.execute();
                    }
                } else {
                    xmlSourceSample.printUsage();
                }
            } catch (Exception e) {
                e.printStackTrace();
                System.err.println("Exception is: " + e.getMessage());
            }
        }
    }
```

## Creating an XML Definition with a Hierarchical Normalized View

The following sample code shows how to import XML definition from an XML schema definition with a hierarchical normalized view:

```
package com.informatica.powercenter.sdk.mapfwk.samples;

import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.connection.SourceTargetType;
import com.informatica.powercenter.sdk.mapfwk.core.Field;
import com.informatica.powercenter.sdk.mapfwk.core.FieldType;
import com.informatica.powercenter.sdk.mapfwk.core.GroupType;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.OutputSet;
import com.informatica.powercenter.sdk.mapfwk.core.PortType;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.Source;
import com.informatica.powercenter.sdk.mapfwk.core.SourceGroup;
import com.informatica.powercenter.sdk.mapfwk.core.Target;
import com.informatica.powercenter.sdk.mapfwk.core.TargetGroup;
import com.informatica.powercenter.sdk.mapfwk.core.TransformHelper;
import com.informatica.powercenter.sdk.mapfwk.core.Transformation;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;
import com.informatica.powercenter.sdk.mapfwk.core.xml.NormalizedView;
import com.informatica.powercenter.sdk.mapfwk.core.xml.XMLSource;
import com.informatica.powercenter.sdk.mapfwk.core.xml.XMLTarget;
import com.informatica.powercenter.sdk.mapfwk.core.xml.XMLView;

public class XMLSourceSample extends Base{

        private XMLSource xmlSrc;
        private XMLTarget xmlTgt;
        protected XMLView normalizedViewSrc = null;
        protected XMLView normalizedViewTgt = null;

        @Override
        protected void createSources() {
        normalizedViewSrc = new NormalizedView("C:\\Users\\vthenapp\\Desktop\\Tasks\\XML
Sources\\uw_original.xsd", GroupType.OUTPUT,FieldType.SOURCE);
        xmlSrc = new XMLSource(normalizedViewSrc,"XMLSource", "XMLSource", "XMLSource",
            "XMLSource", new ConnectionInfo(SourceTargetType.XML));
        xmlSrc.createSourceFields();
        folder.addSource(xmlSrc);
        }

        @Override
        protected void createTargets() {
        normalizedViewTgt = new NormalizedView("C:\\Users\\vthenapp\\Desktop\\Tasks\\XML
Sources\\uw_original.xsd", GroupType.INPUT,FieldType.TARGET);
        xmlTgt = new XMLTarget(normalizedViewTgt, "XMLTarget", "XMLTarget", "XMLTarget",
            "XMLTarget", new ConnectionInfo(SourceTargetType.XML));
        xmlTgt.createTargetFields();
        folder.addTarget(xmlTgt);
        }

        @Override
        protected void createMappings() throws Exception {
        // create a mapping object
        mapping = new Mapping("XMLSourceTargetMapping", "XMLSourceTargetMapping",
        "XML Source Target Mapping");
        setMapFileName(mapping);
        TransformHelper helper = new TransformHelper(mapping);

        // create DSQ
        RowSet dsqRowSet = (RowSet) helper.sourceQualifier(xmlSrc).getRowSets().get(0);
        mapping.writeTarget(dsqRowSet, this.xmlTgt);
        folder.addMapping(mapping);
        }

        @Override
        protected void createSession() throws Exception {
        session = new Session("Session_For_XMLSourceTarget",
"Session_For_XMLSourceTarget",
            "This is session for XML Source / Target");
        session.setMapping(mapping);
        }

        @Override
        protected void createWorkflow() throws Exception {
        workflow = new Workflow("Workflow_for_XMLSourceTarget",
"Workflow_for_XMLSourceTarget",
            "This workflow for XML Source / Target");
        workflow.addSession(session);
        folder.addWorkFlow(workflow);
        }

    public static void main(String[] args) {
            try {
                XMLSourceSample xmlSourceSample = new XMLSourceSample();
                if (args.length > 0) {
                    if (xmlSourceSample.validateRunMode(args[0])) {
                        xmlSourceSample.execute();
                    }
                } else {
                    xmlSourceSample.printUsage();
                }
            } catch (Exception e) {
                e.printStackTrace();
                System.err.println("Exception is: " + e.getMessage());
            }
        }
    }
```

## Accessing an XML Source or Target from the PowerCenter Repository

The following sample code shows how to access an XML source or target from the PowerCenter repository:

```
package com.informatica.powercenter.sdk.mapfwk.samples;

import java.io.IOException;
import java.io.InputStream;
import java.util.List;
import java.util.Properties;

import com.informatica.powercenter.sdk.mapfwk.core.FieldType;
import com.informatica.powercenter.sdk.mapfwk.core.Folder;
import com.informatica.powercenter.sdk.mapfwk.core.Group;
import com.informatica.powercenter.sdk.mapfwk.core.GroupType;
import com.informatica.powercenter.sdk.mapfwk.core.INameFilter;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.Source;
import com.informatica.powercenter.sdk.mapfwk.core.Target;
import com.informatica.powercenter.sdk.mapfwk.core.TransformHelper;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;
import com.informatica.powercenter.sdk.mapfwk.core.xml.EntityView;
import com.informatica.powercenter.sdk.mapfwk.core.xml.NormalizedView;
import com.informatica.powercenter.sdk.mapfwk.core.xml.XMLSource;
import com.informatica.powercenter.sdk.mapfwk.core.xml.XMLTarget;
import com.informatica.powercenter.sdk.mapfwk.core.xml.XMLView;
import com.informatica.powercenter.sdk.mapfwk.exception.MapFwkReaderException;
import com.informatica.powercenter.sdk.mapfwk.exception.RepoOperationException;
import
com.informatica.powercenter.sdk.mapfwk.repository.PmrepRepositoryConnectionManager;
import com.informatica.powercenter.sdk.mapfwk.repository.RepoPropsConstants;
import com.informatica.powercenter.sdk.mapfwk.repository.RepositoryConnectionManager;

public class XMLSourceTargetFetchSample extends Base{

    private XMLSource xmlSrc;
    private XMLTarget xmlTgt;
    protected XMLView entityViewSrc  = null;
    protected XMLView entityViewTgt  = null;
    private int folderSize = 0;
    private List<Folder> folders = null;

    public XMLSourceTargetFetchSample() throws Exception{
        initialize();
        RepositoryConnectionManager repmgr = new PmrepRepositoryConnectionManager();
        rep.setRepositoryConnectionManager(repmgr);
        try {
                folders = rep.getFolders(new INameFilter() {
                public boolean accept(String name) {
                    return name.equals("Temp");
                }
            });
            folderSize = folders.size();
        } catch (RepoOperationException e) {
            e.printStackTrace();
        } catch (MapFwkReaderException e) {
            e.printStackTrace();
        }
    }

    protected void initialize() throws IOException {
        createRepository();
        Properties properties = new Properties();
        String filename = "pcconfig.properties";
        InputStream propStream =
getClass().getClassLoader().getResourceAsStream( filename);

        if ( propStream != null ) {
            properties.load( propStream );

rep.getRepoConnectionInfo().setPcClientInstallPath(properties.getProperty(RepoPropsConsta
nts.PC_CLIENT_INSTALL_PATH));

rep.getRepoConnectionInfo().setTargetFolderName(properties.getProperty(RepoPropsConstants
.TARGET_FOLDER_NAME));

rep.getRepoConnectionInfo().setTargetRepoName(properties.getProperty(RepoPropsConstants.T
ARGET_REPO_NAME));

rep.getRepoConnectionInfo().setRepoServerHost(properties.getProperty(RepoPropsConstants.R
EPO_SERVER_HOST));
```

```
rep.getRepoConnectionInfo().setAdminPassword(properties.getProperty(RepoPropsConstants.AD
MIN_PASSWORD));

rep.getRepoConnectionInfo().setAdminUsername(properties.getProperty(RepoPropsConstants.AD
MIN_USERNAME));

rep.getRepoConnectionInfo().setRepoServerPort(properties.getProperty(RepoPropsConstants.R
EPO_SERVER_PORT));

rep.getRepoConnectionInfo().setServerPort(properties.getProperty(RepoPropsConstants.SERVE
R_PORT));

rep.getRepoConnectionInfo().setDatabaseType(properties.getProperty(RepoPropsConstants.DAT
ABASETYPE));

            if(properties.getProperty(RepoPropsConstants.PMREP_CACHE_FOLDER) != null)

rep.getRepoConnectionInfo().setPmrepCacheFolder(properties.getProperty(RepoPropsConstants
.PMREP_CACHE_FOLDER));
        }
        else {
            throw new IOException( "pcconfig.properties file not found.");
        }
    }

    @Override
    protected void createSources() {
        List<Source> listOfSources = null;
        try {
            for (int i = 0; i < folderSize; i++) {
        //fetch XML source
                            listOfSources = ((Folder) folders.get(i))
                        .fetchSourcesFromRepository(new INameFilter() {
                            @Override
                            public boolean accept(String name) {
                                return name.equals("XMLSourceEntity");
                            }
                        });
            }
            xmlSrc = (XMLSource) listOfSources.get(0);
            xmlSrc.setName("FetchedXMLSource");
            xmlSrc.setBusinessName("FetchedXMLSource");
            xmlSrc.setInstanceName("FetchedXMLSource");
            xmlSrc.setDescription("FetchedXMLSource");
            xmlSrc.setModified(true);
            folder.addSource(xmlSrc);
        } catch (RepoOperationException e) {
            e.printStackTrace();
        } catch (MapFwkReaderException e) {
            e.printStackTrace();
        }
    }

    @Override
    protected void createTargets() {
        List<Target> listOfTargets = null;
        try {
            for (int i = 0; i < folderSize; i++) {
        //fetch XML target
listOfTargets = ((Folder) folders.get(i))
                        .fetchTargetsFromRepository(new INameFilter() {
                            @Override
                            public boolean accept(String name) {
                                return name.equals("XMLTargetEntity");
                            }
                        });
            }
            xmlTgt = (XMLTarget) listOfTargets.get(0);
            xmlTgt.setName("FetchedXMLTarget");
            xmlTgt.setBusinessName("FetchedXMLTarget");
            xmlTgt.setInstanceName("FetchedXMLTarget");
```

```
                xmlTgt.setDescription("FetchedXMLTarget");
                xmlTgt.setModified(true);
                folder.addTarget(xmlTgt);
            } catch (RepoOperationException e) {
                e.printStackTrace();
            } catch (MapFwkReaderException e) {
                e.printStackTrace();
            }
        }

        @Override
        protected void createMappings() throws Exception {
            // create a mapping object
            mapping = new Mapping("FetchedXMLSourceTargetMapping",
    "FetchedXMLSourceTargetMapping", "Fetched XML Source Target Mapping");
            setMapFileName(mapping);

            TransformHelper helper = new TransformHelper(mapping);

            // create DSQ
            RowSet dsqRowSet = (RowSet) helper.sourceQualifier(xmlSrc).getRowSets().get(0);

            mapping.writeTarget(dsqRowSet, this.xmlTgt);

            folder.addMapping(mapping);

        }

        @Override
        protected void createSession() throws Exception {
            session = new Session("Session_For_FetchedXMLSourceTargetMapping",
    "Session_For_FetchedXMLSourceTargetMapping",
            "This is session for Fetched XML Source / Target");
            session.setMapping(mapping);
        }

        @Override
        protected void createWorkflow() throws Exception {
            workflow = new Workflow("Workflow_for_FetchedXMLSourceTarget",
    "Workflow_for_FetchedXMLSourceTarget",
            "This workflow for Fetched XML Source / Target ");
            workflow.addSession(session);
            folder.addWorkFlow(workflow);
        }

        public static void main(String[] args) {
            try {
                XMLSourceTargetFetchSample xmlSourceTargetFetch = new
    XMLSourceTargetFetchSample();
                if (args.length > 0) {
                    if (xmlSourceTargetFetch.validateRunMode(args[0])) {
                        xmlSourceTargetFetch.execute();
                    }
                } else {
                    xmlSourceTargetFetch.printUsage();
                }
            } catch (Exception e) {
                e.printStackTrace();
                System.err.println("Exception is: " + e.getMessage());
            }
        }
    }
```

# Nonrelational Data Sources

The following code examples show how to connect to and use nonrelational data sources and targets through PowerExchange.

## Adabas

The following sample code shows how to connect to an Adabas data source:

```
package com.informatica.powercenter.sdk.mapfwk.samples;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Properties;

import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionPropsConstants;
import com.informatica.powercenter.sdk.mapfwk.core.ASQTransformation;
import com.informatica.powercenter.sdk.mapfwk.core.Field;
import com.informatica.powercenter.sdk.mapfwk.core.FieldKeyType;
import com.informatica.powercenter.sdk.mapfwk.core.FieldType;
import com.informatica.powercenter.sdk.mapfwk.core.InputSet;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.MultiGroupASQTransformation;
import com.informatica.powercenter.sdk.mapfwk.core.OutputSet;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.SourceGroup;
import com.informatica.powercenter.sdk.mapfwk.core.TargetGroup;
import com.informatica.powercenter.sdk.mapfwk.core.TransformGroup;
import com.informatica.powercenter.sdk.mapfwk.core.TransformHelper;
import com.informatica.powercenter.sdk.mapfwk.core.Transformation;
import com.informatica.powercenter.sdk.mapfwk.core.TransformationConstants;
import com.informatica.powercenter.sdk.mapfwk.core.TransformationContext;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;
import com.informatica.powercenter.sdk.mapfwk.exception.InvalidTransformationException;
import com.informatica.powercenter.sdk.mapfwk.exception.MapFwkException;
import com.informatica.powercenter.sdk.mapfwk.exception.RepoOperationException;
import com.informatica.powercenter.sdk.mapfwk.plugin.MainframeConPropsConstants;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectAccessMethodConstants;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectConInfo;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectMetaExtentionConstants;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectSource;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectSourceFactory;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectSourceTargetType;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectTarget;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectTargetFactory;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerExchangeStringConstants;
import 
com.informatica.powercenter.sdk.mapfwk.powercentercompatibility.PowerCenterCompatibilityF
actory;

/**
 *
 * When you run this sample code, set the mapFileName object to absolute location.
 * Otherwise, the example will fail to import an object into the repository.
 *
 *
 */
public class PWXAdabasSample extends Base{

    PowerConnectSource source;
    PowerConnectTarget outputTarget;
    TransformHelper helper;
    SourceGroup srcGrp;
    TargetGroup trgGrp;
    ASQTransformation ASQ_TRANS;
    Transformation filter;

    /* Creates a mapping.
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createMappings()
     */

    protected void createMappings() throws Exception
    {
        mapping = new Mapping("MAPPING_PWX_ADABAS", "MAPPING_PWX_ADABAS",
                "Testing sample with myMapping");
        /*
         *  Method 1 : Creates a source qualifier using TransformationHelper.
         */

        helper = new TransformHelper(mapping);
        RowSet asqRS = (RowSet) helper.sourceQualifier(source).getRowSets().get(0);
        InputSet asqIS = new InputSet(asqRS);
        List<InputSet> inSets = new ArrayList<InputSet>();
        inSets.add(asqIS);
        mapping.writeTarget(inSets, outputTarget);

        /*
         *  Creates a filter transformation on a PowerExchange target.
         */
        TransformationContext trans = new TransformationContext(inSets);
        filter = trans.createTransform(TransformationConstants.FILTER_PROC,
"PWX_FILTER_TRANS");
        RowSet filterRS = (RowSet) filter.apply().getRowSets().get(0);
        mapping.addTransformation(filter);
        mapping.writeTarget(filterRS, outputTarget);

        folder.addMapping(mapping);

        /*
         * Method 2 :    This creates a source qualifier without using the
TransformationHelper.
         *     If you use this method to create a source qualifier, enable this code
   *     and comment out the code in Method 1.
         */

        /*InputSet inSet = new InputSet( source );
        String name;
        RowSet rs =inSet.getOutRowSet();
        List<Field> fields = rs.getFields();
        Iterator<Field> fieldsIter = fields.iterator();
        while(fieldsIter.hasNext())
        {
             Field field = fieldsIter.next();
             field.setDataType(source.getTransformationDataType(field.getDataType()));
              field.setFieldType(FieldType.TRANSFORM);
        }

        TransformationContext tc = new TransformationContext(inSet);

        ASQ_TRANS =
(MultiGroupASQTransformation)tc.createTransform(TransformationConstants.MULTI_GROUP_ASQ,
"AMGDSQ_r3kperf_R3KROOT");
        RowSet asqRS = (RowSet) ASQ_TRANS.apply().getRowSets().get(0);

        TransformGroup transGroup = new
TransformGroup(srcGrp.getGroupName(),srcGrp.getPortType(),srcGrp.getFilterCondition());

        List<TransformGroup> listTransGroup = new ArrayList<TransformGroup>();
        listTransGroup.add(transGroup);

        ASQ_TRANS.setTransformGroups(listTransGroup);


((MultiGroupASQTransformation)ASQ_TRANS).setRefDBDName(source.getConnInfo().getConnProps(
).getProperty(ConnectionPropsConstants.DBNAME));
        ((MultiGroupASQTransformation)ASQ_TRANS).setRefSourceName(source.getName());

        mapping.addTransformation(ASQ_TRANS);
        mapping.writeTarget(asqRS, outputTarget);
        folder.addMapping(mapping); */

    }

    /* Creates a session to run the mapping.
```

```
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSession()
     */
    protected void createSession() throws Exception
    {
        session = new Session("Session_For_PWX_ADABAS",
                "Session_For_PWX_ADABAS", "This is session for PowerExcange Adabas");

        /*
         * Sets the session transformation instance properties PowerExchange sources.
         */
        Properties props = new Properties();
        props.setProperty(MainframeConPropsConstants.ADABAS.TRACING_LEVEL,
PowerExchangeStringConstants.TERSE);
        props.setProperty(MainframeConPropsConstants.ADABAS.OUTPUT_IS_DETERMINISTIC,
PowerExchangeStringConstants.NO);

        /*
         *  If you use method 1 in createMappings(), use the following code.
         */

session.addSessionTransformInstanceProperties(mapping.getTransformations().get(0),
props);

        /*
         *  If you use method 2 in createMappings(), enable the following code to set
session
         *  attributes and comment the previous code.
         */
        //session.addSessionTransformInstanceProperties(ASQ_TRANS, props);

        session.addSessionTransformInstanceProperties(filter, null);
        session.setMapping(this.mapping);
    }

    /* Creates a source.
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSources()
     */
    protected void createSources()
    {
        PowerConnectSourceFactory sourceFactory =
PowerConnectSourceFactory.getInstance();
        try
        {
            source =
sourceFactory.getPowerConnectSourceInstance(PowerConnectSourceTargetType.ADABAS,
"PWX_SRC_ADABAS", "mySourceDBD", "mySource", "r3kperf_R3KROOT");
            srcGrp = new SourceGroup("r3kperf_R3KROOT",(String)null);

source.createField("CCK_R3KROOT_COUNTER",srcGrp,"","","NUM32","10","0",FieldKeyType.PRIMA
RY_KEY, FieldType.SOURCE, true);

source.createField("COUNTER",srcGrp,"","","NUM32","10","0",FieldKeyType.PRIMARY_KEY,
FieldType.SOURCE, true);


source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.ACCESS_METHOD,
PowerConnectAccessMethodConstants.ADABAS);
            //source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.DBD_NAME,
"R3KPERFS");

source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.MAP_OR_TABLE_NAME ,
"r3kperf_R3KROOT");

source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.MAP_OR_TABLE_PREFIX,
"perform");

source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.COMMENT_TEXT, "");

            List<ConnectionInfo> connInfos = source.getConnInfos();
            for (int i=0;i<connInfos.size();i++)
            {
                PowerConnectConInfo connInfo = (PowerConnectConInfo) connInfos.get(i);

connInfo.getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME, "jmf_conn");

connInfo.getConnProps().setProperty( ConnectionPropsConstants.DBNAME,"IMS_BGQALS91_JMF");

                /*
                 *  Sets the session extension properties for a PowerExchange source.
                 */
                Properties connAttributes = new Properties();

connAttributes.setProperty(MainframeConPropsConstants.ADABAS.ADABAS_PASSWORD, "pass");
                connInfo.setCustSessionExtAttr(connAttributes);
            }

        } catch (RepoOperationException e){
            e.printStackTrace();
        } catch (MapFwkException e){
            e.printStackTrace();
        }
        folder.addSource(source);
        this.mapFileName = "C:\\Venkat\\JMF\\JMF\\eBiz\\main\\javamappingsdk\
\PowerExchangeSource_IMS.xml";
    }

    /* (non-Javadoc)
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createTargets()
     */
    protected void createTargets()
    {
        PowerConnectTargetFactory targetFactory =
PowerConnectTargetFactory.getInstance();
        try
        {
            outputTarget =
targetFactory.getPowerConnectTargetInstance(PowerConnectSourceTargetType.ADABAS,
"PWX_TGT_ADABAS",  "myTargetDBD",  "myTargetDBD",  "r3kperf_R3KROOT");
            trgGrp = new TargetGroup("perform",(String)null);

outputTarget.createField("CCK_R3KROOT_COUNTER",trgGrp,"","","NUM32","10","0",FieldKeyType
.PRIMARY_KEY, FieldType.TARGET, true);

outputTarget.createField("COUNTER",trgGrp,"","","NUM32","10","0",FieldKeyType.PRIMARY_KEY
, FieldType.TARGET, true);


outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.ACCESS_METHOD,
PowerConnectAccessMethodConstants.ADABAS);
            //
outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.DBD_NAME,
"R3KPERFS");

outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.MAP_OR_TABLE_NAME ,
 "r3kperf_R3KROOT");

outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.MAP_OR_TABLE_PREFIX
, "perform");

outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.COMMENT_TEXT, "");
            List<ConnectionInfo> connInfos = outputTarget.getConnInfos();
            for (int i=0;i<connInfos.size();i++)
            {
                PowerConnectConInfo connInfo = (PowerConnectConInfo) connInfos.get(i);

connInfo.getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME, "jmf_conn");

connInfo.getConnProps().setProperty( ConnectionPropsConstants.DBNAME,"IMS_BGQALS91_JMF");

                /*
                             *  Sets the session extension properties for a
PowerExchange target.
```

```
                 */
                Properties connAttributes = new Properties();

connAttributes.setProperty(MainframeConPropsConstants.ADABAS.ADABAS_PASSWORD,
"dfdsfsdf");
                connInfo.setCustSessionExtAttr(connAttributes);
            }
        } catch (RepoOperationException e){
            e.printStackTrace();
        } catch (MapFwkException e){
            e.printStackTrace();
        }
        folder.addTarget(outputTarget);
    }

    /* Creates a workflow.
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createWorkflow()
     */
    protected void createWorkflow() throws Exception
    {
        workflow = new Workflow("Workflow_For_PWX_ADABAS",
                "Workflow_For_PWX_ADABAS", "This workflow for PowerExcange Adabas");

        /*
         *  Sets the repository and domain information for the Integration Service.
         */
        workflow.assignIntegrationService("repo_IS", "Domain_IN173082");

        workflow.addSession(session);
        folder.addWorkFlow(workflow);
    }

    public static void main(String args[]) {
        try {
            PWXAdabasSample sample = new PWXAdabasSample();
            if (args.length > 0) {
                if (sample.validateRunMode(args[0])) {
                    PowerCenterCompatibilityFactory compFactory =
PowerCenterCompatibilityFactory.getInstance();
                    compFactory.setCompatibilityVersion(8, 5, 0);
                    sample.execute();
                }
            } else {
                sample.printUsage();
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Exception is: " + e.getMessage());
        }
    }

}
```

## Datacom

The following sample code shows how to connect to a Datacom data source:

```
package com.informatica.powercenter.sdk.mapfwk.samples;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Properties;

import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionPropsConstants;
import com.informatica.powercenter.sdk.mapfwk.core.ASQTransformation;
import com.informatica.powercenter.sdk.mapfwk.core.Field;
import com.informatica.powercenter.sdk.mapfwk.core.FieldKeyType;
import com.informatica.powercenter.sdk.mapfwk.core.FieldType;
import com.informatica.powercenter.sdk.mapfwk.core.InputSet;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.MultiGroupASQTransformation;
import com.informatica.powercenter.sdk.mapfwk.core.OutputSet;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.SourceGroup;
import com.informatica.powercenter.sdk.mapfwk.core.TargetGroup;
import com.informatica.powercenter.sdk.mapfwk.core.TransformGroup;
import com.informatica.powercenter.sdk.mapfwk.core.TransformHelper;
import com.informatica.powercenter.sdk.mapfwk.core.Transformation;
import com.informatica.powercenter.sdk.mapfwk.core.TransformationConstants;
import com.informatica.powercenter.sdk.mapfwk.core.TransformationContext;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;
import com.informatica.powercenter.sdk.mapfwk.exception.InvalidTransformationException;
import com.informatica.powercenter.sdk.mapfwk.exception.MapFwkException;
import com.informatica.powercenter.sdk.mapfwk.exception.RepoOperationException;
import com.informatica.powercenter.sdk.mapfwk.plugin.MainframeConPropsConstants;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectAccessMethodConstants;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectConInfo;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectMetaExtentionConstants;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectSource;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectSourceFactory;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectSourceTargetType;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectTarget;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectTargetFactory;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerExchangeStringConstants;
import
com.informatica.powercenter.sdk.mapfwk.powercentercompatibility.PowerCenterCompatibilityF
actory;

/**
 *
 * When you run this sample code, set the mapFileName object to absolute location.
 * Otherwise, the example will fail to import an object into the repository.
 *
 *
 */
public class PWXDatacomSample extends Base{

    PowerConnectSource source;
    PowerConnectTarget outputTarget;
    TransformHelper helper;
    SourceGroup srcGrp;
    TargetGroup trgGrp;
    ASQTransformation ASQ_TRANS;
    Transformation filter;

    /* Creates mappings.
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createMappings()
     */

    protected void createMappings() throws Exception
    {
        mapping = new Mapping("MAPPING_PWX_DATACOM_SRC", "MAPPING_PWX_DATACOM_SRC",
                "Testing sample with myMapping");
        /*
         *  Method 1 : This creates a source qualifier using TransformationHelper
         */

        helper = new TransformHelper(mapping);
        RowSet asqRS = (RowSet) helper.sourceQualifier(source).getRowSets().get(0);
        InputSet asqIS = new InputSet(asqRS);
        List<InputSet> inSets = new ArrayList<InputSet>();
        inSets.add(asqIS);
        mapping.writeTarget(inSets, outputTarget);

        /*
         *  Creates a filter transformation on a PowerExchange target.
         */
        TransformationContext trans = new TransformationContext(inSets);
        filter = trans.createTransform(TransformationConstants.FILTER_PROC, 
"PWX_FILTER_TRANS");
        RowSet filterRS = (RowSet) filter.apply().getRowSets().get(0);
        mapping.addTransformation(filter);
        mapping.writeTarget(filterRS, outputTarget);

        folder.addMapping(mapping);

        /*
         * Method 2 :    This creates a source qualifier without using the 
TransformationHelper.
         *     If you use this method to create a source qualifier, enable this code
   *     and comment out the code in Method 1.
         */

        /*InputSet inSet = new InputSet( source );
        String name;
        RowSet rs =inSet.getOutRowSet();
        List<Field> fields = rs.getFields();
        Iterator<Field> fieldsIter = fields.iterator();
        while(fieldsIter.hasNext())
        {
             Field field = fieldsIter.next();
             field.setDataType(source.getTransformationDataType(field.getDataType()));
              field.setFieldType(FieldType.TRANSFORM);
        }

        TransformationContext tc = new TransformationContext(inSet);

        ASQ_TRANS = 
(MultiGroupASQTransformation)tc.createTransform(TransformationConstants.MULTI_GROUP_ASQ,
"AMGDSQ_r3kperf_R3KROOT");
        RowSet asqRS = (RowSet) ASQ_TRANS.apply().getRowSets().get(0);

        TransformGroup transGroup = new 
TransformGroup(srcGrp.getGroupName(),srcGrp.getPortType(),srcGrp.getFilterCondition());

        List<TransformGroup> listTransGroup = new ArrayList<TransformGroup>();
        listTransGroup.add(transGroup);

        ASQ_TRANS.setTransformGroups(listTransGroup);


((MultiGroupASQTransformation)ASQ_TRANS).setRefDBDName(source.getConnInfo().getConnProps(
).getProperty(ConnectionPropsConstants.DBNAME));
        ((MultiGroupASQTransformation)ASQ_TRANS).setRefSourceName(source.getName());

        mapping.addTransformation(ASQ_TRANS);
        mapping.writeTarget(asqRS, outputTarget);
        folder.addMapping(mapping); */

    }

    /* Creates a session to run the mapping.
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSession()
     */
    protected void createSession() throws Exception
    {
        session = new Session("Session_For_PWX_DATACOM_SRC",
                "Session_For_PWX_DATACOM_SRC", "This is session for PowerExcange DATACOM 
Src");

        /*
         * Sets the session transformation instance properties for PowerExchange 
sources.
         */
        Properties props = new Properties();
        props.setProperty(MainframeConPropsConstants.IDMS.TRACING_LEVEL, 
PowerExchangeStringConstants.TERSE);
        props.setProperty(MainframeConPropsConstants.IDMS.OUTPUT_IS_DETERMINISTIC, 
PowerExchangeStringConstants.NO);
```

```
        /*
         *  If you use method 1 in createMappings(), use the following code.
         */

session.addSessionTransformInstanceProperties(mapping.getTransformations().get(0),
props);

        /*
         *  If you use method 2 in createMappings(), enable the following code to set
session
         *  attributes and comment the previous code.
         */
        //session.addSessionTransformInstanceProperties(ASQ_TRANS, props);

        session.addSessionTransformInstanceProperties(filter, null);
        session.setMapping(this.mapping);
    }

    /* Creates a source.
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSources()
     */
    protected void createSources()
    {
        PowerConnectSourceFactory sourceFactory =
PowerConnectSourceFactory.getInstance();
        try
        {
            source =
sourceFactory.getPowerConnectSourceInstance(PowerConnectSourceTargetType.DATACOM,
"PWX_SRC_DATACOM", "DATACOM_mySourceDBD", "DATACOM_mySource",
"DATACOM_r3kperf_R3KROOT_DC");
            srcGrp = new SourceGroup("r3kperf_R3KROOT",(String)null);

source.createField("CCK_R3KROOT_COUNTER",srcGrp,"","","NUM32","10","0",FieldKeyType.PRIMA
RY_KEY, FieldType.SOURCE, true);

source.createField("COUNTER",srcGrp,"","","NUM32","10","0",FieldKeyType.PRIMARY_KEY,
FieldType.SOURCE, true);


source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.ACCESS_METHOD,
PowerConnectAccessMethodConstants.DATACOM);
            //source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.DBD_NAME,
"R3KPERFS");

source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.MAP_OR_TABLE_NAME ,
"r3kperf_R3KROOT");

source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.MAP_OR_TABLE_PREFIX,
"perform");

source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.COMMENT_TEXT, "");

            List<ConnectionInfo> connInfos = source.getConnInfos();
            for (int i=0;i<connInfos.size();i++)
            {
                PowerConnectConInfo connInfo = (PowerConnectConInfo) connInfos.get(i);

connInfo.getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME, "jmf_conn");

connInfo.getConnProps().setProperty( ConnectionPropsConstants.DBNAME,"IMS_BGQALS91_JMF");

                /*
                 *  Sets the session extension properties for a PowerExchange source.
                 */
                Properties connAttributes = new Properties();
                //
connAttributes.setProperty(MainframeConPropsConstants.ADABAS.ADABAS_PASSWORD, "pass");
                connInfo.setCustSessionExtAttr(connAttributes);
            }
        } catch (RepoOperationException e){
            e.printStackTrace();
        } catch (MapFwkException e){
            e.printStackTrace();
        }
        folder.addSource(source);
        this.mapFileName = "C:\\Venkat\\JMF\\JMF\\eBiz\\main\\javamappingsdk\
\PowerExchangeSource_DATACOM.xml";
    }

    /* Creates a target.
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createTargets()
     */
    protected void createTargets()
    {
        PowerConnectTargetFactory targetFactory =
PowerConnectTargetFactory.getInstance();
        try
        {
            outputTarget =
targetFactory.getPowerConnectTargetInstance(PowerConnectSourceTargetType.ADABAS,
"PWX_TGT_ADABAS",  "myTargetDBD",  "myTargetDBD",  "r3kperf_R3KROOT");
            trgGrp = new TargetGroup("perform",(String)null);

outputTarget.createField("CCK_R3KROOT_COUNTER",trgGrp,"","","NUM32","10","0",FieldKeyType
.PRIMARY_KEY, FieldType.TARGET, true);

outputTarget.createField("COUNTER",trgGrp,"","","NUM32","10","0",FieldKeyType.PRIMARY_KEY
, FieldType.TARGET, true);


outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.ACCESS_METHOD,
PowerConnectAccessMethodConstants.ADABAS);
            //
outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.DBD_NAME,
"R3KPERFS");

outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.MAP_OR_TABLE_NAME ,
 "r3kperf_R3KROOT");

outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.MAP_OR_TABLE_PREFIX
, "perform");

outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.COMMENT_TEXT, "");
            List<ConnectionInfo> connInfos = outputTarget.getConnInfos();
            for (int i=0;i<connInfos.size();i++)
            {
                PowerConnectConInfo connInfo = (PowerConnectConInfo) connInfos.get(i);

connInfo.getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME, "jmf_conn");

connInfo.getConnProps().setProperty( ConnectionPropsConstants.DBNAME,"IMS_BGQALS91_JMF");

                /*
                             *  Sets the session extension properties for a
PowerExchange target.
                 */
                Properties connAttributes = new Properties();
                //
connAttributes.setProperty(MainframeConPropsConstants.ADABAS.ADABAS_PASSWORD,
"dfdsfsdf");
                connInfo.setCustSessionExtAttr(connAttributes);
            }
        } catch (RepoOperationException e){
            e.printStackTrace();
        } catch (MapFwkException e){
            e.printStackTrace();
        }
        folder.addTarget(outputTarget);
    }
    /* Creates a workflow.
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createWorkflow()
     */
    protected void createWorkflow() throws Exception
    {
        workflow = new Workflow("Workflow_For_PWX_DATACOM_SRC",
                "Workflow_For_PWX_DATACOM_SRC", "This workflow for PowerExcange IDMS");

        /*
         *  Sets the repository and domain information for the Integration Service.
         */
        workflow.assignIntegrationService("repo_IS", "Domain_IN173082");

        workflow.addSession(session);
        folder.addWorkFlow(workflow);
    }

    public static void main(String args[]) {
        try {
            PWXDatacomSample sample = new PWXDatacomSample();
            if (args.length > 0) {
                if (sample.validateRunMode(args[0])) {
                    PowerCenterCompatibilityFactory compFactory =
PowerCenterCompatibilityFactory.getInstance();
                    compFactory.setCompatibilityVersion(8, 5, 0);
                    sample.execute();
                }
            } else {
                sample.printUsage();
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Exception is: " + e.getMessage());
        }
    }

}
```

## IDMS

The following sample code shows how to connect to an IDMS data source:

```
package com.informatica.powercenter.sdk.mapfwk.samples;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Properties;

import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionPropsConstants;
import com.informatica.powercenter.sdk.mapfwk.core.ASQTransformation;
import com.informatica.powercenter.sdk.mapfwk.core.Field;
import com.informatica.powercenter.sdk.mapfwk.core.FieldKeyType;
import com.informatica.powercenter.sdk.mapfwk.core.FieldType;
import com.informatica.powercenter.sdk.mapfwk.core.InputSet;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.MultiGroupASQTransformation;
import com.informatica.powercenter.sdk.mapfwk.core.OutputSet;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.SourceGroup;
import com.informatica.powercenter.sdk.mapfwk.core.TargetGroup;
import com.informatica.powercenter.sdk.mapfwk.core.TransformGroup;
import com.informatica.powercenter.sdk.mapfwk.core.TransformHelper;
import com.informatica.powercenter.sdk.mapfwk.core.Transformation;
import com.informatica.powercenter.sdk.mapfwk.core.TransformationConstants;
import com.informatica.powercenter.sdk.mapfwk.core.TransformationContext;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;
import com.informatica.powercenter.sdk.mapfwk.exception.InvalidTransformationException;
import com.informatica.powercenter.sdk.mapfwk.exception.MapFwkException;
import com.informatica.powercenter.sdk.mapfwk.exception.RepoOperationException;
import com.informatica.powercenter.sdk.mapfwk.plugin.MainframeConPropsConstants;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectAccessMethodConstants;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectConInfo;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectMetaExtentionConstants;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectSource;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectSourceFactory;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectSourceTargetType;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectTarget;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectTargetFactory;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerExchangeStringConstants;
import
com.informatica.powercenter.sdk.mapfwk.powercentercompatibility.PowerCenterCompatibilityF
actory;

/**
 *
 * When you run this sample code, set the mapFileName object to absolute location.
 * Otherwise, the example will fail to import an object into the repository.
 *
 */
public class PWXIDMSSample extends Base{

    PowerConnectSource source;
    PowerConnectTarget outputTarget;
    TransformHelper helper;
    SourceGroup srcGrp;
    TargetGroup trgGrp;
    ASQTransformation ASQ_TRANS;
    Transformation filter;

    /* Creates a mapping.
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createMappings()
     */

    protected void createMappings() throws Exception
    {
        mapping = new Mapping("MAPPING_PWX_IDMS_SRC", "MAPPING_PWX_IDMS_SRC",
                "Testing sample with myMapping");
        /*
         *  Method 1 : Creates a source qualifier using TransformationHelper.
         */

        helper = new TransformHelper(mapping);
        RowSet asqRS = (RowSet) helper.sourceQualifier(source).getRowSets().get(0);
        InputSet asqIS = new InputSet(asqRS);
        List<InputSet> inSets = new ArrayList<InputSet>();
        inSets.add(asqIS);
        mapping.writeTarget(inSets, outputTarget);

        /*
         *  Creates a filter transformation on a PowerExchange target.
         */
        TransformationContext trans = new TransformationContext(inSets);
        filter = trans.createTransform(TransformationConstants.FILTER_PROC,
"PWX_FILTER_TRANS");
        RowSet filterRS = (RowSet) filter.apply().getRowSets().get(0);
        mapping.addTransformation(filter);
        mapping.writeTarget(filterRS, outputTarget);

        folder.addMapping(mapping);

        /*
         * Method 2 :    This creates a source qualifier without using the
TransformationHelper.
         *     If you use this method to create a source qualifier, enable this code
   *     and comment out the code in Method 1.
         */
        /*InputSet inSet = new InputSet( source );
        String name;
        RowSet rs =inSet.getOutRowSet();
        List<Field> fields = rs.getFields();
        Iterator<Field> fieldsIter = fields.iterator();
        while(fieldsIter.hasNext())
        {
             Field field = fieldsIter.next();
             field.setDataType(source.getTransformationDataType(field.getDataType()));
              field.setFieldType(FieldType.TRANSFORM);
        }

        TransformationContext tc = new TransformationContext(inSet);

        ASQ_TRANS =
(MultiGroupASQTransformation)tc.createTransform(TransformationConstants.MULTI_GROUP_ASQ,
"AMGDSQ_r3kperf_R3KROOT");
        RowSet asqRS = (RowSet) ASQ_TRANS.apply().getRowSets().get(0);

        TransformGroup transGroup = new
TransformGroup(srcGrp.getGroupName(),srcGrp.getPortType(),srcGrp.getFilterCondition());

        List<TransformGroup> listTransGroup = new ArrayList<TransformGroup>();
        listTransGroup.add(transGroup);

        ASQ_TRANS.setTransformGroups(listTransGroup);


((MultiGroupASQTransformation)ASQ_TRANS).setRefDBDName(source.getConnInfo().getConnProps(
).getProperty(ConnectionPropsConstants.DBNAME));
        ((MultiGroupASQTransformation)ASQ_TRANS).setRefSourceName(source.getName());

        mapping.addTransformation(ASQ_TRANS);
        mapping.writeTarget(asqRS, outputTarget);
        folder.addMapping(mapping); */

    }

    /* Creates a session to run the mapping.
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSession()
     */
    protected void createSession() throws Exception
    {
        session = new Session("Session_For_PWX_IDMS_SRC",
                "Session_For_PWX_IDMS_SRC", "This is session for PowerExcange IDMS Src");

        /*
         * Sets the session transformation instance properties for PowerExchange
sources.
         */
        Properties props = new Properties();
        props.setProperty(MainframeConPropsConstants.IDMS.TRACING_LEVEL,
PowerExchangeStringConstants.TERSE);
        props.setProperty(MainframeConPropsConstants.IDMS.OUTPUT_IS_DETERMINISTIC,
PowerExchangeStringConstants.NO);

        /*
         *  If you use method 1 in createMappings(), use the following code.
         */

session.addSessionTransformInstanceProperties(mapping.getTransformations().get(0),
props);

        /*
         *  If you use method 2 in createMappings(), enable the following code to set
session
         *  attributes and comment out the previous code.
         */
        //session.addSessionTransformInstanceProperties(ASQ_TRANS, props);

        session.addSessionTransformInstanceProperties(filter, null);
        session.setMapping(this.mapping);
    }

    /* Creates a source.
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSources()
     */
    protected void createSources()
    {
        PowerConnectSourceFactory sourceFactory = 
PowerConnectSourceFactory.getInstance();
        try
        {
            source = 
sourceFactory.getPowerConnectSourceInstance(PowerConnectSourceTargetType.IDMS, 
"PWX_SRC_IDMS", "IDMS_mySourceDBD", "IDMS_mySource", "IDMS_r3kperf_R3KROOT_IDMS");
            srcGrp = new SourceGroup("r3kperf_R3KROOT",(String)null);

source.createField("CCK_R3KROOT_COUNTER",srcGrp,"","","NUM32","10","0",FieldKeyType.PRIMA
RY_KEY, FieldType.SOURCE, true);

source.createField("COUNTER",srcGrp,"","","NUM32","10","0",FieldKeyType.PRIMARY_KEY, 
FieldType.SOURCE, true);


source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.ACCESS_METHOD, 
PowerConnectAccessMethodConstants.IDMS);
            //source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.DBD_NAME, 
"R3KPERFS");

source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.MAP_OR_TABLE_NAME , 
"r3kperf_R3KROOT");

source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.MAP_OR_TABLE_PREFIX, 
"perform");

source.setMetaExtensionValue(PowerConnectMetaExtentionConstants.COMMENT_TEXT, "");

            List<ConnectionInfo> connInfos = source.getConnInfos();
            for (int i=0;i<connInfos.size();i++)
            {
                PowerConnectConInfo connInfo = (PowerConnectConInfo) connInfos.get(i);

connInfo.getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME, "jmf_conn");

connInfo.getConnProps().setProperty( ConnectionPropsConstants.DBNAME,"IMS_BGQALS91_JMF");

                /*
                 *  Sets the session extension properties for a PowerExchange source.
                 */
                Properties connAttributes = new Properties();
                // 
connAttributes.setProperty(MainframeConPropsConstants.ADABAS.ADABAS_PASSWORD, "pass");
                connInfo.setCustSessionExtAttr(connAttributes);
            }

        } catch (RepoOperationException e){
            e.printStackTrace();
        } catch (MapFwkException e){
            e.printStackTrace();
        }
        folder.addSource(source);
        this.mapFileName = "C:\\Venkat\\JMF\\JMF\\eBiz\\main\\javamappingsdk\
\PowerExchangeSource_IDMS.xml";
    }

    /* Creates a target.
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createTargets()
     */
    protected void createTargets()
    {
        PowerConnectTargetFactory targetFactory = 
PowerConnectTargetFactory.getInstance();
        try
        {
            outputTarget =
targetFactory.getPowerConnectTargetInstance(PowerConnectSourceTargetType.ADABAS,
"PWX_TGT_ADABAS",  "myTargetDBD",  "myTargetDBD",  "r3kperf_R3KROOT");
            trgGrp = new TargetGroup("perform",(String)null);

outputTarget.createField("CCK_R3KROOT_COUNTER",trgGrp,"","","NUM32","10","0",FieldKeyType
.PRIMARY_KEY, FieldType.TARGET, true);

outputTarget.createField("COUNTER",trgGrp,"","","NUM32","10","0",FieldKeyType.PRIMARY_KEY
, FieldType.TARGET, true);


outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.ACCESS_METHOD,
PowerConnectAccessMethodConstants.ADABAS);
            //
outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.DBD_NAME,
"R3KPERFS");

outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.MAP_OR_TABLE_NAME ,
 "r3kperf_R3KROOT");

outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.MAP_OR_TABLE_PREFIX
, "perform");

outputTarget.setMetaExtensionValue(PowerConnectMetaExtentionConstants.COMMENT_TEXT, "");
            List<ConnectionInfo> connInfos = outputTarget.getConnInfos();
            for (int i=0;i<connInfos.size();i++)
            {
                PowerConnectConInfo connInfo = (PowerConnectConInfo) connInfos.get(i);

connInfo.getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME, "jmf_conn");

connInfo.getConnProps().setProperty( ConnectionPropsConstants.DBNAME,"IMS_BGQALS91_JMF");

                /*
                             *  Sets the session extension properties for a
PowerExchange target.
                 */
                Properties connAttributes = new Properties();
                //
connAttributes.setProperty(MainframeConPropsConstants.ADABAS.ADABAS_PASSWORD,
"dfdsfsdf");
                connInfo.setCustSessionExtAttr(connAttributes);
            }
        } catch (RepoOperationException e){
            e.printStackTrace();
        } catch (MapFwkException e){
            e.printStackTrace();
        }
        folder.addTarget(outputTarget);
    }

    /* Creates a workflow.
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createWorkflow()
     */
    protected void createWorkflow() throws Exception
    {
        workflow = new Workflow("Workflow_For_PWX_IDMS",
                "Workflow_For_PWX_IDMS", "This workflow for PowerExcange IDMS");

        /*
         *  This is how integration service repo and domain information can be set.
         */
        workflow.assignIntegrationService("repo_IS", "Domain_IN173082");

        workflow.addSession(session);
        folder.addWorkFlow(workflow);
    }
    public static void main(String args[]) {
        try {
            PWXIDMSSample sample = new PWXIDMSSample();
            if (args.length > 0) {
                if (sample.validateRunMode(args[0])) {
                    PowerCenterCompatibilityFactory compFactory =
PowerCenterCompatibilityFactory.getInstance();
                    compFactory.setCompatibilityVersion(8, 5, 0);
                    sample.execute();
                }
            } else {
                sample.printUsage();
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Exception is: " + e.getMessage());
        }
    }

}
```

## SEQ Access Method

You can use the SEQ access method to connect to flat files and sequential data sets.

The following sample code shows how to use the PowerExchange SEQ access method:

```
package com.informatica.powercenter.sdk.mapfwk.samples;

import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionPropsConstants;
import com.informatica.powercenter.sdk.mapfwk.connection.SourceTargetType;
import com.informatica.powercenter.sdk.mapfwk.core.ASQTransformation;
import com.informatica.powercenter.sdk.mapfwk.core.Field;
import com.informatica.powercenter.sdk.mapfwk.core.FieldKeyType;
import com.informatica.powercenter.sdk.mapfwk.core.FieldType;
import com.informatica.powercenter.sdk.mapfwk.core.InputSet;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.SourceGroup;
import com.informatica.powercenter.sdk.mapfwk.core.Target;
import com.informatica.powercenter.sdk.mapfwk.core.TargetGroup;
import com.informatica.powercenter.sdk.mapfwk.core.TransformHelper;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;
import com.informatica.powercenter.sdk.mapfwk.exception.MapFwkException;
import com.informatica.powercenter.sdk.mapfwk.exception.RepoOperationException;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectAccessMethodConstants;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectConInfo;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectMetaExtentionConstants;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectSource;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectSourceFactory;
import com.informatica.powercenter.sdk.mapfwk.plugin.PowerConnectSourceTargetType;
import
com.informatica.powercenter.sdk.mapfwk.portpropagation.PortPropagationContextFactory;
import
com.informatica.powercenter.sdk.mapfwk.powercentercompatibility.PowerCenterCompatibilityF
actory;

/**
 *
 * When you run this sample code, set the mapFileName object to absolute location.
 * Otherwise, the sample will fail to import an object into the repository.
 *
 *
 */
public class PWXSeqSample extends Base {

    PowerConnectSource source;
    Target outputTarget;
    TransformHelper helper;
    SourceGroup srcGrp;
    TargetGroup trgGrp;
    ASQTransformation ASQ_TRANS;

    /*
     * Creates a mapping.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createMappings()
     */

    protected void createMappings() throws Exception {
        mapping = new Mapping("MAPPING_PWX_SEQ", "MAPPING_PWX_SEQ",
                "Testing sample with myMapping");
        /*
         * Method 1 : Creates a source qualifier using TransformationHelper.
         */

        helper = new TransformHelper(mapping);
        RowSet asqRS = (RowSet) helper.sourceQualifier(source).getRowSets()
                .get(0);
        InputSet asqIS = new InputSet(asqRS,
                PortPropagationContextFactory
                        .getContextForIncludeCols(new String[] { "NAME" }));
        List<InputSet> inSets = new ArrayList<InputSet>();
        inSets.add(asqIS);
        mapping.writeTarget(inSets, outputTarget);

        folder.addMapping(mapping);
    }

    /*
     *  Creates a session to run the mapping.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSession()
     */
    protected void createSession() throws Exception {
        session = new Session("Session_For_PWX_SEQ", "Session_For_PWX_SEQ",
                "This is session for PowerExcange SEQ Files");

        Properties props = new Properties();

        /*
         * If you use method 1 in createMappings(), use the following code.
         */
        session.addSessionTransformInstanceProperties(mapping
                .getTransformations().get(0), props);

        /*
         * When you follow method 2 in createMappings(), enable the following code
         * to set session attributes and comment the above code.
         */
        // session.addSessionTransformInstanceProperties(ASQ_TRANS, props);

        session.setMapping(this.mapping);
    }

    /*
     * Creates a source.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createSources()
     */
    protected void createSources() {
        PowerConnectSourceFactory sourceFactory = PowerConnectSourceFactory
                .getInstance();
        try {
            source = sourceFactory.getPowerConnectSourceInstance(
                    PowerConnectSourceTargetType.SEQ, "datamap_NAME_REC",
                    "mySourceDBD", "mySource", "r3kperf_R3KROOT");
            srcGrp = new SourceGroup("datamap_NAME_REC", (String) null);
            source.createField("ACCOUNT", srcGrp, "", "", "UZONED", "3", "0",
                    FieldKeyType.NOT_A_KEY, FieldType.SOURCE, true);
            source.createField("RECTYPE", srcGrp, "", "", "UZONED", "2", "0",
                    FieldKeyType.NOT_A_KEY, FieldType.SOURCE, true);
            source.createField("NAME", srcGrp, "", "", "CHAR", "20", "0",
                    FieldKeyType.NOT_A_KEY, FieldType.SOURCE, true);
            source.createField("SEX", srcGrp, "", "", "CHAR", "1", "0",
                    FieldKeyType.NOT_A_KEY, FieldType.SOURCE, true);
            source.createField("ITEMCT", srcGrp, "", "", "UZONED", "1", "0",
                    FieldKeyType.NOT_A_KEY, FieldType.SOURCE, true);
            source.createField("ITEMS_1", srcGrp, "", "", "CHAR", "10", "0",
                    FieldKeyType.NOT_A_KEY, FieldType.SOURCE, true);
            source.createField("ITEMS_2", srcGrp, "", "", "CHAR", "10", "0",
                    FieldKeyType.NOT_A_KEY, FieldType.SOURCE, true);
            source.createField("ITEMS_3", srcGrp, "", "", "CHAR", "10", "0",
                    FieldKeyType.NOT_A_KEY, FieldType.SOURCE, true);

            source.setMetaExtensionValue(
                    PowerConnectMetaExtentionConstants.ACCESS_METHOD,
                    PowerConnectAccessMethodConstants.SEQ);
            source.setMetaExtensionValue(
                    PowerConnectMetaExtentionConstants.FILE_NAME,
                    "C:\\Program Files\\Informatica\\Informatica PowerExchange\\examples\
\train3.dat");
            source.setMetaExtensionValue(
                    PowerConnectMetaExtentionConstants.MAP_NAME,
                    "datamap_NAME_REC");
            source.setMetaExtensionValue(
                    PowerConnectMetaExtentionConstants.SCHEMA_NAME, "traiu3");

            List<ConnectionInfo> connInfos = source.getConnInfos();
            for (int i = 0; i < connInfos.size(); i++) {
                PowerConnectConInfo connInfo = (PowerConnectConInfo) connInfos
                        .get(i);
                connInfo.getConnProps().setProperty(
                        ConnectionPropsConstants.CONNECTIONNAME,
                        "PWX_NRDB_Batch");
                connInfo.getConnProps().setProperty(
                        ConnectionPropsConstants.DBNAME, "PWX_test");
                connInfo.getConnProps().setProperty(
                        ConnectionPropsConstants.OWNERNAME, "traiu3");
            }

        } catch (RepoOperationException e) {
            e.printStackTrace();
        } catch (MapFwkException e) {
            e.printStackTrace();
        }
        folder.addSource(source);
        this.mapFileName = "C:\\PowerExchangeSource_SEQ.xml";
    }

    /*
     * Creates a target.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createTargets()
     */
    protected void createTargets() {
        ConnectionInfo conInfo = new ConnectionInfo(SourceTargetType.Flat_File);
        outputTarget = new Target("rc_file", "", "", "rec_file", conInfo);
        outputTarget.addField(new Field("NAME", "", "", "string", "20", "0",
                FieldKeyType.NOT_A_KEY, FieldType.TARGET, false));
        folder.addTarget(outputTarget);
    }

    /*
     * Creates a workflow.
     *
     * @see com.informatica.powercenter.sdk.mapfwk.samples.Base#createWorkflow()
     */
    protected void createWorkflow() throws Exception {
        workflow = new Workflow("Workflow_For_PWX_SEQ", "Workflow_For_PWX_SEQ",
                "This workflow for PowerExcange SEQ Files");

        /*
         * Sets the Integration Service and domain information in the workflow.
         */
        workflow.assignIntegrationService("Integration_Service", "Domain_vm");

        workflow.addSession(session);
        folder.addWorkFlow(workflow);
    }

    public static void main(String args[]) {
        try {
            PWXSeqSample sample = new PWXSeqSample();
            if (args.length > 0) {
                if (sample.validateRunMode(args[0])) {
                    PowerCenterCompatibilityFactory compFactory =
PowerCenterCompatibilityFactory
                            .getInstance();
                    compFactory.setCompatibilityVersion(8, 5, 0);
                    sample.execute();
                }
            } else {
                sample.printUsage();
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println("Exception is: " + e.getMessage());
        }
    }
}
```

## Teradata MultiLoad Connection

The following sample code shows how to create a Teradata MultiLoad connection object:

```
package com.informatica.powercenter.sdk.mapfwk.samples;

import java.util.ArrayList;
import java.util.List;

import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionInfo;
import com.informatica.powercenter.sdk.mapfwk.connection.ConnectionPropsConstants;
import com.informatica.powercenter.sdk.mapfwk.connection.SourceTargetType;
import com.informatica.powercenter.sdk.mapfwk.core.Field;
import com.informatica.powercenter.sdk.mapfwk.core.FieldKeyType;
import com.informatica.powercenter.sdk.mapfwk.core.FieldType;
import com.informatica.powercenter.sdk.mapfwk.core.InputSet;
import com.informatica.powercenter.sdk.mapfwk.core.Mapping;
import com.informatica.powercenter.sdk.mapfwk.core.NativeDataTypes;
import com.informatica.powercenter.sdk.mapfwk.core.RowSet;
import com.informatica.powercenter.sdk.mapfwk.core.Session;
import com.informatica.powercenter.sdk.mapfwk.core.Source;
import com.informatica.powercenter.sdk.mapfwk.core.Target;
import com.informatica.powercenter.sdk.mapfwk.core.Transformation;
import com.informatica.powercenter.sdk.mapfwk.core.TransformationConstants;
import com.informatica.powercenter.sdk.mapfwk.core.TransformationContext;
import com.informatica.powercenter.sdk.mapfwk.core.Workflow;

public class TeradataMLoadSample extends Base{

    protected Target target;
    protected Source source;
    protected Transformation dsqTransform;

    public TeradataMLoadSample(){
        target = null;
        source = null;
        dsqTransform = null;
    }

    @Override
    protected void createSources() {
        this.source = this.createOracleJobSource("mloadSource");

this.source.setSessionTransformInstanceProperty(ConnectionPropsConstants.OWNER_NAME,
"DSFds");
        this.folder.addSource(this.source);
    }

    @Override
    protected void createTargets() {
        List<Field> fields = new ArrayList<Field>();
        Field jobIDField = new Field( "JOB_ID1", "JOB_ID", "",
            NativeDataTypes.Teradata.VARCHAR, "10", "0",FieldKeyType.PRIMARY_KEY,
FieldType.SOURCE, true );
        fields.add( jobIDField );

        Field jobTitleField = new Field( "JOB_TITLE1", "JOB_TITLE", "",
            NativeDataTypes.Teradata.VARCHAR, "35", "0",
            FieldKeyType.NOT_A_KEY, FieldType.SOURCE, false );
        fields.add( jobTitleField );

        Field minSalField = new Field( "MIN_SALARY1", "MIN_SALARY", "",
            NativeDataTypes.Teradata.DECIMAL, "6", "0",
            FieldKeyType.NOT_A_KEY, FieldType.SOURCE, false );
        fields.add( minSalField );

        Field maxSalField = new Field( "MAX_SALARY1", "MAX_SALARY", "",
            NativeDataTypes.Teradata.DECIMAL, "6", "0",
            FieldKeyType.NOT_A_KEY, FieldType.SOURCE, false );
        fields.add( maxSalField );

        ConnectionInfo info = getRelationalConnInfo( SourceTargetType.Teradata ,
"tera_tgt");
        target = new Target( "mloadTarget", "mloadTarget", "This is mloadTarget table",
"JOBS", info );
        target.setFields(fields);


target.setSessionTransformInstanceProperty(ConnectionPropsConstants.TARGET_TABLE_NAME,
"fgfdg");
        this.folder.addTarget(this.target);
    }

    protected Target createRelationalTarget( SourceTargetType DBType, String name ) {
        ConnectionInfo con = new ConnectionInfo( DBType );
        con.getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME,"tera");
        Target target = new Target(name, name, name, name, con ) ;
        return target;
    }

    @Override
    protected void createMappings() throws Exception {
        mapping = new Mapping("Teradata_mload", "Teradata_mload", "This is
Teradata_mload");
        setMapFileName(mapping);
        List<InputSet> inputSets = new ArrayList<InputSet>();
        InputSet tptSource = new InputSet(this.source);
        inputSets.add(tptSource);
        TransformationContext tc = new TransformationContext( inputSets );
        dsqTransform = tc.createTransform( TransformationConstants.DSQ, "TPT_DSQ" );

this.dsqTransform.setSessionTransformInstanceProperty(ConnectionPropsConstants.USER_DEFIN
ED_JOIN, "1join");
        RowSet dsqRS = (RowSet) dsqTransform.apply().getRowSets().get( 0 );
        mapping.addTransformation( dsqTransform );
        mapping.writeTarget(dsqRS, this.target);
        folder.addMapping(mapping);
    }

    @Override
    protected void createSession() throws Exception {
        session = new Session( "session_For_Teradata_mload",
"Session_For_Teradata_mload",
            "This is session for Teradata_mload" );
        session.setMapping( this.mapping );

        /* Overridden target connectionInfo*/
        ConnectionInfo tgtconInfo = new
ConnectionInfo(SourceTargetType.Teradata_Mload_External_Loader);

tgtconInfo.getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME,"mload");

tgtconInfo.getConnProps().setProperty(ConnectionPropsConstants.FOOTER_COMMAND,"Footer
Command");

tgtconInfo.getConnProps().setProperty(ConnectionPropsConstants.FLATFILE_CODEPAGE,"MS1252"
);
        session.addConnectionInfoObject(target, tgtconInfo);

  /* Overridden source connectionInfo*/
        ConnectionInfo conInfo =
this.getRelationalConnectionInfo(SourceTargetType.Oracle);

conInfo.getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME,"Oracle");

conInfo.getConnProps().setProperty(ConnectionPropsConstants.DRIVER_TRACING_LEVEL,"TD_OPER
_NOTIFY");

conInfo.getConnProps().setProperty(ConnectionPropsConstants.INFRASTRUCTURE_TRACING_LEVEL,
"TD_OPER_NOTIFY");

conInfo.getConnProps().setProperty(ConnectionPropsConstants.TRACE_FILE_NAME,"trace
file");

conInfo.getConnProps().setProperty(ConnectionPropsConstants.QUERY_BAND_EXPRESSION,"quary
band exp");
        session.addConnectionInfoObject(dsqTransform, conInfo);
    }

    @Override
    protected void createWorkflow() throws Exception {
            workflow = new Workflow( "Workflow_for_Teradata_mload",
"Workflow_for_Teradata_mload",
            "This workflow for Teradata_mload" );
        workflow.addSession( session );

        folder.addWorkFlow( workflow );
    }

    public static void main(String[] args) {
        try {
            TeradataMLoadSample teradataMload = new TeradataMLoadSample();
            if (args.length > 0) {
                if (teradataMload.validateRunMode( args[0] )) {
                    teradataMload.execute();
                }
            } else {
                teradataMload.printUsage();
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.err.println( "Exception is: " + e.getMessage() );
        }
        }
    }
```

# Sample Patterns for Regular Expressions for Port Propagation

You can use regular expressions to propagate and link ports. Use expressions when port names follow patterns that include prefixes or suffixes. You can use regular expressions with the PatternMatchStrategy class.

The regular expression pattern specification has the following format:

```
<From Port Pattern> (TO) <To Port Pattern>
```

The following table shows examples of regular expression patterns:

| Pattern | Description |
|---|---|
| [A-Za-z_][A-Za-z_0-9]* (TO) x_$0 | Prefix all port names with *x_*. Examples:<br>EMPLOYEE_ID => x_EMPLOYEE_ID<br>EMPLOYEE_NAME => x_EMPLOYEE_NAME<br>GENDER => x_GENDER<br>ETHIC_GROUP => x_ETHIC_GROUP |
| pattern = ^EMP.*_.*$ (TO) $0_IN | Select port names that start with *EMP* and append *_IN* to the end of the port name. Examples:<br>EMPLOYEE_ID => EMPLOYEE_ID_IN<br>EMPLOYEE_NAME => EMPLOYEE_NAME_IN<br>GENDER is not selected.<br>ETHIC_GROUP is not selected. |
| pattern = [\d]$ (TO) $0 | Select port names that end with a digit and keep the name as is. Examples:<br>EMPLOYEE_ID is not selected.<br>EMPLOYEE_NAME1 => EMPLOYEE_NAME1<br>GENDER35 => GENDER35<br>ETHIC_GROUP is not selected. |
| _IN$ | Select port names with the suffix *_IN* and remove the suffix from the port names. Examples:<br>EMPLOYEE_ID is not selected.<br>EMPLOYEE_NAME is not selected.<br>GENDER_IN => GENDER<br>ETHIC_GROUP_IN => ETHIC_GROUP |
| ^IN_ | Select port names that with the prefix *IN_* and remove the prefix from the port names. Examples:<br>IN_EMPLOYEE_ID => EMPLOYEE_ID<br>IN_EMPLOYEE_NAME => EMPLOYEE_NAME<br>GENDER is not selected.<br>IN_ETHIC_GROUP => ETHIC_GROUP |

APPENDIX B

# Interface Limitations

This appendix includes the following topics:

## Interface Limitations Overview

This appendix describes the limitations of the APIs in the Informatica Development Platform.

## Design API

The Design API has the following limitations:

- **Data source support.** The following are the Design API data source limitations:
  - You cannot create XML metadata from the following files types: XML, DTD, relational tables, and flat files.
  - You cannot create an XML source or XML target definition from an XML schema definition with a hierarchical denormalized view.
- **Transformations.** You cannot use the Design API to generate transformation metadata for the following transformations:
  - Unstructured Data
  - Data Quality
  - Java

## PowerExchange API

The PowerExchange API has the following limitations:

- You can create pass-through partitions but not key-range partitions.

- You can generate queries for reader session attributes such as SQL Query, User Defined Join, Pre SQL, Post SQL, and Source Filter. However, you cannot validate the queries.
- There are no setter methods for the Date and Time/TimeStamp datatypes in the IOutputBuffer interface. To work with these datatypes, convert them to long.
- There are no setter methods for the CLOB, BLOB, and XML datatypes in the IOutputBuffer interface. To work with the CLOB datatype, convert it to string. To work with the BLOB datatype, convert it to byte[].
- By default, the PowerExchange API readers are also created as application type readers for relational data sources. When you run a session, you cannot switch a JDBC source or target to a relational reader or writer.
- The PowerExchange API does not expose the Source Rows As session property. The Java DB adapter works around this limitation by getting the default value for the Source Rows As session property from the IUtilSrv. This displays the Source Rows A property and the values Insert, Delete, Update, and Data Driven in the Workflow Manager. You can also use an Update strategy transformation in a mapping to work around this limitation.
- The PowerExchange API does not support built-in mapping-level attributes such as SQL overrides and SQL filter. If you configure mapping-level attributes for the Source Qualifier in the Designer, the values of the attributes are not visible in the session but are evaluated when a query is prepared.
- You cannot use a PowerExchange API adapter to connect to a Lookup transformation.
- The PowerExchange API does not support data recovery or resiliency.
- The PowerExchange API cannot access the reject file to load rejected rows.
- The PowerExchange API for Java does not implement the ITargetField.getRefPKInfo() method.
- The isDiscardOutput interface for the Debugger is not available in PowerExchange API for Java.

# INDEX