# Table of Contents

Informatica Resources ........................................ vii
Informatica Customer Portal ................................ vii
Informatica Documentation ................................... vii
Informatica Web Site ........................................ vii
Informatica How-To Library ................................ vii
Informatica Knowledge Base ................................ viii
Informatica Global Customer Support ....................... viii

**Chapter 1: Introduction to PowerCenter Data Integration** ................. 1
Overview ......................................................... 1
Enterprise Application Components ............................... 2
Embedded Data Integration ....................................... 3
PowerCenter Data Integration ..................................... 4

**Chapter 2: PowerCenter Interfaces** .................................. 7
Overview .......................................................... 7
Informatica Development Platform ................................. 7
Command Line Programs ........................................ 8
Web Services ...................................................... 8
PowerExchange API ............................................... 9
Usage ............................................................... 9
Requirements ...................................................... 9
Transformation API ................................................ 10
Usage .............................................................. 10
Operations API ...................................................... 10
Usage .............................................................. 11
Design API ........................................................ 11
Usage .............................................................. 11
Custom Function API ............................................ 12
Usage .............................................................. 12

**Chapter 3: Developing a PowerExchange Adapter** ....................... 13
Overview .......................................................... 13
Step 1. Get the PowerCenter Repository ID Attributes ..................... 13
Step 2. Define the Metadata for the Plug-In ............................. 15
Step 3. Register the Plug-in Metadata .................................. 15
Step 4. Register the Client Plug-in in the Windows Registry ............... 16
Step 5. Set Up the Development Environment ........................... 16
  Defining the Path for the DLL .................................. 16
  Selecting a Compiler ........................................... 16
Step 6. Build Server and Client Plug-ins ................................ 17
Unregistering a PowerExchange Plug-in ............................. 19
<table>
<thead>
<tr>
<th>Chapter 4: Plug-in Metadata</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>21</td>
</tr>
<tr>
<td>Structure of the Plug-in Definition File</td>
<td>21</td>
</tr>
<tr>
<td>PLUGIN Element</td>
<td>22</td>
</tr>
<tr>
<td>DBTYPE Element</td>
<td>23</td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>EXTENSION Element</td>
<td>33</td>
</tr>
<tr>
<td>ATTRIBUTE Element</td>
<td>34</td>
</tr>
<tr>
<td>LIBRARY Element</td>
<td>36</td>
</tr>
<tr>
<td>CLASS Element</td>
<td>36</td>
</tr>
<tr>
<td>ALLOWEDDBTYPE Element</td>
<td>37</td>
</tr>
<tr>
<td>ALLOWEDTEMPLATE Element</td>
<td>37</td>
</tr>
<tr>
<td>CONNECTIONREFERENCE Element</td>
<td>37</td>
</tr>
<tr>
<td>CONNECTION Element</td>
<td>40</td>
</tr>
<tr>
<td>DBTYPETOEXTENSION Element</td>
<td>41</td>
</tr>
<tr>
<td>CONNECTIONTOEXTENSION Element</td>
<td>42</td>
</tr>
<tr>
<td>HIDDENEXTENSIONATTRIBUTETOCONNECTION Element</td>
<td>43</td>
</tr>
<tr>
<td>MEDOMAIN Element</td>
<td>43</td>
</tr>
<tr>
<td>MEDEFINITION Element</td>
<td>44</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 5: PowerExchange API Example</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>47</td>
</tr>
<tr>
<td>Plug-in Definition File</td>
<td>47</td>
</tr>
<tr>
<td>Server Plug-in</td>
<td>48</td>
</tr>
<tr>
<td>Plug-in Definition for the Java DB Adapter</td>
<td>48</td>
</tr>
<tr>
<td>PLUGIN</td>
<td>48</td>
</tr>
<tr>
<td>DBTYPE</td>
<td>48</td>
</tr>
<tr>
<td>EXTENSION</td>
<td>50</td>
</tr>
<tr>
<td>CONNECTION</td>
<td>53</td>
</tr>
<tr>
<td>CONNECTIONTOEXTENSION</td>
<td>54</td>
</tr>
<tr>
<td>MEDOMAIN</td>
<td>54</td>
</tr>
<tr>
<td>Objects and Methods in the Java DB adapter</td>
<td>55</td>
</tr>
<tr>
<td>Reader Session</td>
<td>56</td>
</tr>
<tr>
<td>Writer Session</td>
<td>57</td>
</tr>
<tr>
<td>Adapter Processes</td>
<td>58</td>
</tr>
<tr>
<td>Using the Java DB Adapter</td>
<td>60</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 6: Design API</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>63</td>
</tr>
<tr>
<td>Browsing Metadata in PowerCenter Repository</td>
<td>63</td>
</tr>
<tr>
<td>Chapter 7: Design API Example</td>
<td>75</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----</td>
</tr>
<tr>
<td>Overview</td>
<td>75</td>
</tr>
<tr>
<td>Overview of the Process Flow</td>
<td>75</td>
</tr>
<tr>
<td>Using Outer Joins to Load the Dimension Table</td>
<td>77</td>
</tr>
<tr>
<td>Mapping Result</td>
<td>78</td>
</tr>
<tr>
<td>Design API Methods</td>
<td>79</td>
</tr>
<tr>
<td>Retrieving Metadata</td>
<td>79</td>
</tr>
<tr>
<td>Calling the Design API Methods</td>
<td>80</td>
</tr>
<tr>
<td>Installing and Running the Sample Application</td>
<td>90</td>
</tr>
<tr>
<td>Setting Up the Runtime Environment</td>
<td>90</td>
</tr>
<tr>
<td>Running the Sample Application</td>
<td>92</td>
</tr>
<tr>
<td>Recompiling the Sample Application</td>
<td>92</td>
</tr>
<tr>
<td>Limitations of the Sample Application</td>
<td>92</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix A: Design API Sample Code</th>
<th>93</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>93</td>
</tr>
<tr>
<td>Sample Code Using the Design API</td>
<td>93</td>
</tr>
<tr>
<td>Using Multiple Instance of Sources and Targets</td>
<td>93</td>
</tr>
<tr>
<td>Using One Source Qualifier for Multiple Sources</td>
<td>94</td>
</tr>
<tr>
<td>Using Mapplets</td>
<td>94</td>
</tr>
<tr>
<td>Creating and Using Shorcuts</td>
<td>95</td>
</tr>
<tr>
<td>Creating and Using Reusable Transformations</td>
<td>95</td>
</tr>
<tr>
<td>Validating Objects</td>
<td>95</td>
</tr>
<tr>
<td>Creating Parameter Files</td>
<td>96</td>
</tr>
<tr>
<td>Creating Sources</td>
<td>96</td>
</tr>
<tr>
<td>Defining a PowerExchange Source</td>
<td>97</td>
</tr>
<tr>
<td>Creating Transformations</td>
<td>98</td>
</tr>
<tr>
<td>Sample Patterns for Regular Expressions for Port Propagation</td>
<td>101</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix B: Interface Limitations</th>
<th>103</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>103</td>
</tr>
<tr>
<td>Design API</td>
<td>103</td>
</tr>
<tr>
<td>PowerExchange API</td>
<td>104</td>
</tr>
</tbody>
</table>
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 Customer Portal

As an Informatica customer, you can access the Informatica Customer Portal site at http://my.informatica.com. The site contains product information, user group information, newsletters, access to the Informatica customer support case management system (ATLAS), the Informatica How-To Library, the Informatica Knowledge Base, Informatica Documentation Center, and access to the Informatica user community.

Informatica Documentation

The Informatica Documentation team takes every effort to create accurate, usable documentation. If you have questions, comments, or ideas about this documentation, contact the Informatica Documentation team through email at infa_documentation@informatica.com. We will use your feedback to improve our documentation. Let us know if we can contact you regarding your comments.

Informatica Web Site

You can access the Informatica corporate web site at http://www.informatica.com. The site contains information about Informatica, its background, upcoming events, and sales offices. You will also find product and partner information. The services area of the site includes important information about technical support, training and education, and implementation services.

Informatica How-To Library

As an Informatica customer, you can access the Informatica How-To Library at http://my.informatica.com. The How-To Library is a collection of resources to help you learn more about Informatica products and features. It
includes articles and interactive demonstrations that provide solutions to common problems, compare features and behaviors, and guide you through performing specific real-world tasks.

**Informatica Knowledge Base**

As an Informatica customer, you can access the Informatica Knowledge Base at http://my.informatica.com. Use the Knowledge Base to search for documented solutions to known technical issues about Informatica products. You can also find answers to frequently asked questions, technical white papers, and technical tips.

**Informatica Global Customer Support**

There are many ways to access Informatica Global Customer Support. You can contact a Customer Support Center through telephone, email, or the WebSupport Service.

Use the following email addresses to contact Informatica Global Customer Support:

- support@informatica.com for technical inquiries
- support_admin@informatica.com for general customer service requests

WebSupport requires a user name and password. You can request a user name and password at http://my.informatica.com.

Use the following telephone numbers to contact Informatica Global Customer Support:

<table>
<thead>
<tr>
<th>North America / South America</th>
<th>Europe / Middle East / Africa</th>
<th>Asia / Australia</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Informatica Corporation</strong></td>
<td><strong>Informatica Software Ltd.</strong></td>
<td><strong>Informatica Business</strong></td>
</tr>
<tr>
<td>Headquarters</td>
<td>6 Waltham Park</td>
<td>Solutions Pvt. Ltd.</td>
</tr>
<tr>
<td>100 Cardinal Way</td>
<td>Waltham Road, White Waltham</td>
<td>Diamond District</td>
</tr>
<tr>
<td>Redwood City, California</td>
<td>Maidenhead, Berkshire</td>
<td>Tower B, 3rd Floor</td>
</tr>
<tr>
<td>94063</td>
<td>SL6 3TN</td>
<td>150 Airport Road</td>
</tr>
<tr>
<td>United States</td>
<td>United Kingdom</td>
<td>Bangalore 560 008</td>
</tr>
<tr>
<td><strong>Toll Free</strong></td>
<td><strong>Toll Free</strong></td>
<td><strong>India</strong></td>
</tr>
<tr>
<td>+1 877 463 2435</td>
<td>00 800 4632 4357</td>
<td>Australia: 1 800 151 830</td>
</tr>
<tr>
<td><strong>Standard Rate</strong></td>
<td><strong>Standard Rate</strong></td>
<td>Singapore: 001 800 4632 4357</td>
</tr>
<tr>
<td>Brazil: +55 11 3523 7761</td>
<td>Belgium: +32 1 281 702</td>
<td><strong>Toll Free</strong></td>
</tr>
<tr>
<td>Mexico: +52 55 1168 9763</td>
<td>France: +33 1 41 38 92 26</td>
<td>Australia: 1 800 151 830</td>
</tr>
<tr>
<td>United States: +1 650 385 5800</td>
<td>Germany: +49 1805 702 702</td>
<td>Singapore: 001 800 4632 4357</td>
</tr>
<tr>
<td></td>
<td>Netherlands: +31 306 022 797</td>
<td><strong>Standard Rate</strong></td>
</tr>
<tr>
<td></td>
<td>Spain and Portugal: +34 93 480 3760</td>
<td>India: +91 80 4112 5738</td>
</tr>
<tr>
<td></td>
<td>United Kingdom: +44 1628 511 445</td>
<td></td>
</tr>
</tbody>
</table>

viii  Preface
Chapter 1

Introduction to PowerCenter Data Integration

This chapter includes the following topics:

♦ Overview, 1
♦ Enterprise Application Components, 2
♦ Embedded Data Integration, 3
♦ PowerCenter Data Integration, 4

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

Figure 1-1 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.
Figure 1-2 shows the services that a data integration platform should provide through a data integration interface to control the application components.

**Figure 1-2. Data Integration Platform Interface to Application**

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:

<table>
<thead>
<tr>
<th>Capability</th>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extensible data sources</td>
<td>PowerExchange API</td>
<td>Provides connectivity to custom data sources and data formats. Available in Java and C++.</td>
</tr>
<tr>
<td>Extensible transformation processing</td>
<td>Transformation API</td>
<td>Allows you to invoke application specific APIs to extend transformation processing. Available in Java and C.</td>
</tr>
<tr>
<td>Application driven data integration logic</td>
<td>Design API</td>
<td>Allows you to control data integration logic in the application and to create metadata for PowerCenter objects without a user interface.</td>
</tr>
<tr>
<td>Execution control and monitoring</td>
<td>- Operations API</td>
<td>Allows you to drive execution and monitoring of PowerCenter integration processes through an API, the command line, or web services.</td>
</tr>
<tr>
<td></td>
<td>- Command line interface (pmcmd)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Batch Web Services</td>
<td></td>
</tr>
<tr>
<td>Capability</td>
<td>Interface</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Security and access control</td>
<td>Command line interface (pmrep)</td>
<td>Allows you to administer PowerCenter user accounts and manage application connections.</td>
</tr>
<tr>
<td>Administration</td>
<td>Command line interface (pmrep)</td>
<td>Enables you to administer the data integration metadata, perform backups, and migrate data integration components across environments.</td>
</tr>
</tbody>
</table>
This chapter includes the following topics:

♦ Overview, 7
♦ PowerExchange API, 9
♦ Transformation API, 10
♦ Operations API, 10
♦ Design API, 11
♦ Custom Function API, 12

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.
♦ Mapping 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 *PowerCenter Installation 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.

For more information, see the *PowerCenter Command Reference*.

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.

For more information, see the *PowerCenter Web Services Provider Guide*. 
PowerExchange API

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++.

For more information, see “PowerExchange API” on page 15.

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 C and Java.

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.

For more information, see “Design API” on page 63.

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 Java.

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.
Overview

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 Table 3-1. If you develop multiple plug-ins, each plug-in must have unique repository ID attribute values.

Table 3-1 describes the repository ID attributes that define a plug-in:

Table 3-1. Repository ID Attributes

<table>
<thead>
<tr>
<th>Repository ID Attribute</th>
<th>Description</th>
<th>Test Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vendor ID</td>
<td>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. For more information, see “PLUGIN Element” on page 17.</td>
<td>If you do not have a vendor ID, use 2001.</td>
</tr>
<tr>
<td>dbType</td>
<td>Identifies the database type for the application. This value corresponds to the ID attribute for the DBTYPE element in the plug-in definition file. For more information, see “DBTYPE Element” on page 17.</td>
<td>Use any value from 200,000 to 299,900, in increments of 100.</td>
</tr>
<tr>
<td>Datatype range</td>
<td>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. For more information, see “DATATYPE Element” on page 23.</td>
<td>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.</td>
</tr>
<tr>
<td>Extension subtype</td>
<td>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. For more information, see “EXTENSION Element” on page 25.</td>
<td>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.</td>
</tr>
<tr>
<td>Connection subtype</td>
<td>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. For more information, see “CONNECTION Element” on page 32.</td>
<td>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.</td>
</tr>
<tr>
<td>Metadata extension</td>
<td>Groups metadata extensions into one domain. This value corresponds to the ID attribute for the MEDOMAIN element in the plug-in definition file. For more information, see “MEDOMAIN Element” on page 36.</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

For more information about the plugin.dtd file, see “Plug-in Metadata” on page 21.

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.

For more information, see the *PowerCenter Administrator Guide*. 
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:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Environment Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>PATH</td>
</tr>
<tr>
<td>Solaris, Linux</td>
<td>LD_LIBRARY_PATH</td>
</tr>
<tr>
<td>HP-UX</td>
<td>SHLIB_PATH</td>
</tr>
<tr>
<td>AIX</td>
<td>LIBPATH</td>
</tr>
</tbody>
</table>

Selecting a Compiler

When you build plug-ins for PowerCenter, select a compiler that is compatible with your development platform. For example, to build the client plug-in for the PowerCenter Client, use the Microsoft Visual Studio (VC++) .Net 2003 compiler on Windows.
When you develop a plug-in in Java, use the compiler for the version of Java installed with PowerCenter. For PowerCenter 8.6, use the compiler for Java 1.5.0_11.

When you develop a plug-in in C++, use one of the following compilers based on your development platform:

<table>
<thead>
<tr>
<th>Operating System*</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Microsoft Visual C++ 6.0 or compatible.</td>
</tr>
<tr>
<td>Solaris</td>
<td>For PowerCenter 6.1 or later, Sun Workshop 4.2 or compatible. For PowerCenter 6.0 or earlier, Sun Workshop 5.3 or compatible.</td>
</tr>
<tr>
<td>HP-UX</td>
<td>A.03.25 or compatible.</td>
</tr>
<tr>
<td>AIX</td>
<td>AIX 5 or compatible.</td>
</tr>
</tbody>
</table>

*Operating system of the machine on which PowerCenter is installed

---

### 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:**

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:
   - `<PowerExchangeAPIInstallDir>\Include`
   - `<PowerExchangeAPIInstallDir>\Include\sdkclient`
   - `<PowerExchangeAPIInstallDir>\Include\sdkcli`
   - `<PowerExchangeAPIInstallDir>\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.
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:

<table>
<thead>
<tr>
<th>UNIX platform</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solaris</td>
<td>make -f makefile.sun</td>
</tr>
<tr>
<td>HP-UX (32-bit)</td>
<td>make -f makefile.hp</td>
</tr>
<tr>
<td>HP-UX (64-bit)</td>
<td>make -f makefile.hp64</td>
</tr>
<tr>
<td>AIX (32-bit)</td>
<td>make -f makefile.aix</td>
</tr>
<tr>
<td>AIX (64-bit)</td>
<td>make -f makefile.aix64</td>
</tr>
</tbody>
</table>

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.

**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.

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.

For more information, see the *PowerCenter Administrator Guide*. 
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 provide 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
        -DBTYPEEOWIDGETATTR
        -KEYTYPE
        -DATATYPE
        -FIELDATTR
        -DBTYPEEOWIDGETATTR
        -MULTIVALUEATTRIBUTE
        -LIBRARY
        -AUXFILE
      -EXTENSION
        -ATTRIBUTE
        -MULTIVALUEATTRIBUTE
        -ATTRIBUTECATEGORY
        -LIBRARY
        -AUXFILE
        -CLASS
        -ALLOWEDDBTYPE
        -ALLOWEDTEMPLATE
        -CONNECTIONREFERENCE
        -ALLOWEDCONNECTION
        -HIDDENCONNECTIONATTIBUTETOEXTENSION
      -CONNECTION
        -ATTRIBUTE
        -MULTIVALUEATTRIBUTE
        -ATTRIBUTECATEGORY
        -LIBRARY
        -AUXFILE
        -DBTYPEEEXTENSION
        -CONNECTIONTOEXTENSION
        -HIDDENCONNECTIONATTIBUTETOEXTENSION
        -HIDDENEXTENSIONATTIBUTETOCONNECTION
      -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.
Table 4-1 describes the attributes of the PLUGIN element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Required</td>
<td>Name of the plug-in.</td>
</tr>
<tr>
<td>ID</td>
<td>Required</td>
<td>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.</td>
</tr>
<tr>
<td>VENDORNAME</td>
<td>Required</td>
<td>Name of the vendor.</td>
</tr>
<tr>
<td>VENDORID</td>
<td>Required</td>
<td>Identifier for the vendor obtained from Informatica.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Optional</td>
<td>Description for the plug-in.</td>
</tr>
<tr>
<td>VERSION</td>
<td>Required</td>
<td>Version of the plug-in. Use this attribute to keep track of updates to the plug-in.</td>
</tr>
</tbody>
</table>

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.

Table 4-2 describes the attributes of the DBTYPE element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Required</td>
<td>Name of the third-party database that you want to define for the plug-in.</td>
</tr>
<tr>
<td>ID</td>
<td>Required</td>
<td>Identifier for the database type obtained from Informatica. This attribute identifies this DBTYPE.</td>
</tr>
<tr>
<td>BASEID</td>
<td>Required</td>
<td>Base ID for the datatypes that can be used with this DBTYPE. Use the lowest value from the datatype range obtained from Informatica.</td>
</tr>
<tr>
<td>DEFAULTDBSUBTYPE</td>
<td>Required</td>
<td>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.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Required/Optional</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FIELDFIELDSEPARATOR</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>INVALIDCHARS</td>
<td>Optional</td>
<td>Use this attribute to specify characters that cannot be used in table, field, transformation, or port names. For example, if the $ and &amp; characters are invalid, set the value of this attribute to “$&amp;”. The PowerExchange API framework uses this attribute to perform validation.</td>
</tr>
<tr>
<td>INVALIDFIRSTCHARS</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>TYPE</td>
<td>Required</td>
<td>Type of PowerCenter object to associate with this DBTYPE. You can set this attribute to one of the following values: - SOURCE - TARGET - BOTH</td>
</tr>
<tr>
<td>COMPONENTVERSION</td>
<td>Required</td>
<td>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. Update this attribute only when the DBTYPE has changed. This attribute does not depend on the version of the plug-in.</td>
</tr>
<tr>
<td>DATETIMEFORMAT</td>
<td>Optional</td>
<td>Date and time format to use with this DBTYPE.</td>
</tr>
<tr>
<td>HASGROUPS</td>
<td>Optional</td>
<td>Indicates whether fields for this DBTYPE can be grouped. Set to YES to enable groups for fields in an object with this DBTYPE. Set to NO to disable groups.</td>
</tr>
<tr>
<td>HASFIELDATTRS</td>
<td>Optional</td>
<td>Indicates whether fields of this DBTYPE can have attributes. Set to YES to enable attributes for fields in an object with this DBTYPE. Set to NO to disable attributes. If you set this attribute to NO, you cannot include a FIELDATTR child element for this DBTYPE.</td>
</tr>
<tr>
<td>HASKEYTYPE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>HASNULLTYPE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>HASBUSINESSNAME</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>HASFLATFILE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITGROUPS</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITFIELDATTRS</td>
<td>Optional</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 4-2. Attributes for the DBTYPE Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDITFIELDNAME</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITDATATYPE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITPRECISION</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITSCALE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITKEYTYPE</td>
<td>Optional</td>
<td>Indicates whether key types for this DBTYPE can be edited. 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.</td>
</tr>
<tr>
<td>EDITNULLTYPE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITBUSINESSNAME</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITFLATFILE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>ISRELATIONAL</td>
<td>Required</td>
<td>Indicates whether this DBTYPE is relational. Set to YES if the DBTYPE is relational. Set to NO to specify the DBTYPE as non-relational.</td>
</tr>
<tr>
<td>CANPREVIEWDATA</td>
<td>Optional</td>
<td>Set to YES to enable data preview for this DBTYPE. Set to NO to disable data preview.</td>
</tr>
<tr>
<td>FIXEDFIELDS</td>
<td>Optional</td>
<td>Set to YES to prevent editing or adding of fields to this DBTYPE. Set to NO to allow editing or adding fields.</td>
</tr>
<tr>
<td>CANCHANGEDBTYPETO</td>
<td>Optional</td>
<td>Set to YES to enable other DBTYPES to change into this DBTYPE. Set to NO to disable other DBTYPES from changing into this DBTYPE.</td>
</tr>
<tr>
<td>CANCHANGEDBTYPEFROM</td>
<td>Optional</td>
<td>Set to YES to enable this DBTYPE to change into other DBTYPES. Set to NO to disable this DBTYPE from changing into other DBTYPES.</td>
</tr>
<tr>
<td>CANCOPYFIELDSTO</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>CANCOPYFIELDSFROM</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>CANLINKFIELDSFROM</td>
<td>Optional</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

Table 4-3 describes the attributes of the DBSUBTYPE element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Required</td>
<td>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.</td>
</tr>
<tr>
<td>NAME</td>
<td>Required</td>
<td>Name of the DBSUBTYPE.</td>
</tr>
<tr>
<td>TYPE</td>
<td>Required</td>
<td>Type of PowerCenter object to associate with this DBSUBTYPE. You can set this attribute to one of the following values: - SOURCE - TARGET - BOTH</td>
</tr>
<tr>
<td>HASGROUPS</td>
<td>Optional</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Table 4-3. Attributes for the DBSUBTYPE Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBTYPE.Element</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>HASFIELDATTRS</td>
<td>Optional</td>
<td>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. If you set this attribute to NO, you cannot include a FIELDATTR child element for this DBSUBTYPE.</td>
</tr>
<tr>
<td>HASKEYTYPE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>HASNULLTYPE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>HASBUSINESSNAME</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>HASFLATFILE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITGROUPS</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITFIELDATTRS</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITFILENAME</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITDATATYPE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITPRECISION</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITSOURCE</td>
<td>Optional</td>
<td>Indicates whether datatypes 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.</td>
</tr>
<tr>
<td>EDITKEYTYPE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITNULLTYPE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Required/Optional</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EDITBUSINESSNAME</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>EDITFLATFILE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>ISRELATIONAL</td>
<td>Required</td>
<td>Indicates whether this DBSUBTYPE is relational. Set to YES if the DBSUBTYPE is relational. Set to NO to specify the DBSUBTYPE as non-relational.</td>
</tr>
<tr>
<td>CANPREVIEWDATA</td>
<td>Optional</td>
<td>Set to YES to enable data preview for this DBSUBTYPE. Set to NO to disable data preview.</td>
</tr>
<tr>
<td>FIXEDFIELDS</td>
<td>Optional</td>
<td>Set to YES to prevent editing or adding of fields to this DBSUBTYPE. Set to NO to allow editing or adding fields.</td>
</tr>
<tr>
<td>CANCHANGEDEBTYPETO</td>
<td>Optional</td>
<td>Set to YES to enable other DBSUBTYPEs to change into this DBSUBTYPE. Set to NO to disable other DBSUBTYPEs from changing into this DBSUBTYPE.</td>
</tr>
<tr>
<td>CANCHANGEDEBTYPETFROM</td>
<td>Optional</td>
<td>Set to YES to enable this DBSUBTYPE to change into other DBSUBTYPEs. Set to NO to disable this DBSUBTYPE from changing into other DBSUBTYPEs.</td>
</tr>
<tr>
<td>CANCOPYFIELDSTO</td>
<td>Optional</td>
<td>Set to YES to enable copying fields to a source or target of this DBSUBTYPE. Set to NO to disable copying fields to a source or target of this DBSUBTYPE.</td>
</tr>
<tr>
<td>CANCOPYFIELDSFROM</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>CANLINKFIELDSTO</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>CANLINKFIELDSFROM</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>CANBECREATED</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>CANADDNEWSOURCEFIELD</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>CANADDNEWTARGETFIELD</td>
<td>Optional</td>
<td>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</td>
</tr>
</tbody>
</table>

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.

Table 4-4 describes the attributes of the KEYTYPE element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Required</td>
<td>Name of the key type.</td>
</tr>
<tr>
<td>TABLETYPE</td>
<td>Required</td>
<td>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.</td>
</tr>
<tr>
<td>KEYTYPE</td>
<td>Required</td>
<td>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.</td>
</tr>
<tr>
<td>KEYTYPEBIT</td>
<td>Optional</td>
<td>Decimal value of the key type bits for the key type. 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. Set this attribute only for custom key types.</td>
</tr>
</tbody>
</table>

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'/>`

Table 4-5 describes the attributes of the DATATYPE element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Required</td>
<td>Name of the datatype.</td>
</tr>
<tr>
<td>ID</td>
<td>Required</td>
<td>Identifier for the datatype. The ID must be within the range of DATATYPE IDs provided by Informatica.</td>
</tr>
<tr>
<td>ODBCTYPE</td>
<td>Required</td>
<td>ODBC type of this datatype. Define a separate DATATYPE element for each ODBC type.</td>
</tr>
<tr>
<td>READONLYPRECISION</td>
<td>Optional</td>
<td>Indicates the default precision for this datatype. If the CANEDITPRECISION attribute for this DATATYPE is set to YES, set this attribute to &quot;0&quot;.</td>
</tr>
<tr>
<td>READONLYSCALE</td>
<td>Optional</td>
<td>Indicates the default scale for this datatype. If the CANEDITSCALE attribute for this DATATYPE is set to YES, set this attribute to &quot;0&quot;.</td>
</tr>
</tbody>
</table>
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
- SQLVARCHAR

### 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:

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

Table 4-6 describes the attributes of the FIELDATTR element:

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

**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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Associated Object</th>
<th>Description</th>
<th>Pre-defined Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Scope</td>
<td>Target</td>
<td>Determines when the writer plug-in loads processed rows into the external application.</td>
<td>This property can have one of the following values: - Row. Loads a row after it is processed. - Transaction. Loads all rows processed in a transaction on commit. - All Input. Loads all rows at end of file. Default is All Input.</td>
</tr>
<tr>
<td>Partial Load Recovery</td>
<td>Target</td>
<td>Specifies how the target handles a previous partial load during recovery.</td>
<td>This property can have one of the following values: - None - Append - Truncate Default is None.</td>
</tr>
</tbody>
</table>
Table 4-7 describes the attributes of the DBTYPETOWIDGETATTR element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECTTYPE</td>
<td>Required</td>
<td>Type of PowerCenter object to associate with this DBTYPETOWIDGETATTR. You can set this attribute to one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SOURCE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TARGET</td>
</tr>
<tr>
<td>WIDGETATTRIBUTENAME</td>
<td>Required</td>
<td>Name of the pre-defined property.</td>
</tr>
<tr>
<td>ISREADONLY</td>
<td>Optional</td>
<td>Set to YES to make the property read-only. Set to NO if the user can edit the value.</td>
</tr>
<tr>
<td>ISDISABLED</td>
<td>Optional</td>
<td>Set to YES to disable the property. Set to NO enable the property.</td>
</tr>
<tr>
<td>ISHIDDEN</td>
<td>Optional</td>
<td>Set to YES to hide the property in the Designer. Set to NO to display the property in the Designer.</td>
</tr>
<tr>
<td>ISEXPORTED</td>
<td>Optional</td>
<td>Set to YES if this attribute can be exported. Set to NO if this attribute cannot exported.</td>
</tr>
<tr>
<td>ALLOWVALUES</td>
<td>Optional</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 ALLOWVALUES 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. For more information, see “ATTRIBUTE Element” on page 34.

Table 4-8 describes the attributes of the MULTIVALUEATTRIBUTE element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Required</td>
<td>Enter one of the values of an attribute or property with multiple possible values.</td>
</tr>
</tbody>
</table>

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:

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

Table 4-9 describes the attributes of the LIBRARY element:

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

The LIBRARY element has the following child element:

♦ 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:

```xml
<EXTENSION NAME= "HTML_WRITER" EXTENSIONTYPE= "WRITER" EXTENSIONSUBTYPE = "500001" DESCRIPTION= "HTML WRITER" COMPONENTVERSION = "1.0.0">
```

Table 4-10 describes the attributes of the EXTENSION element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Required</td>
<td>Name of the session extension.</td>
</tr>
<tr>
<td>EXTENSIONTYPE</td>
<td>Required</td>
<td>Type of extension. Set to READER for a reader extension. Set to WRITER for a writer extension.</td>
</tr>
<tr>
<td>EXTENSIONSUBTYPE</td>
<td>Required</td>
<td>Extension subtype. You can obtain the value for this attribute from Informatica.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Optional</td>
<td>Description of the session extension.</td>
</tr>
</tbody>
</table>
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:

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

Table 4-11 describes the attributes of the ATTRIBUTE element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Required</td>
<td>Name of the attribute.</td>
</tr>
<tr>
<td>ID</td>
<td>Required</td>
<td>Identifier for the attribute.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Required/Optional</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TYPE</td>
<td>Required</td>
<td>Type for the attribute. Set to one of the following values: - SQL - PROPERTY - BOOLEAN - MULTIVALUED - PASSWORD - FILENAME. 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 Table 4-12 on page 36.</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>Required</td>
<td>Set to NUMBER or STRING based on the value of the TYPE attribute. For more information, see Table 4-12 on page 36.</td>
</tr>
<tr>
<td>REFERENCELEVEL</td>
<td>Required</td>
<td>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.</td>
</tr>
<tr>
<td>ISINSTANCEOVERRIDABLE</td>
<td>Optional</td>
<td>Set to YES to enable overriding reusable session instances. Set to NO to disable instance overrides.</td>
</tr>
<tr>
<td>ISSESSIONOVERRIDABLE</td>
<td>Optional</td>
<td>Set to YES to enable session overrides. Set to NO to disable session overrides.</td>
</tr>
<tr>
<td>ISPARTITIONOVERRIDABLE</td>
<td>Optional</td>
<td>Set to YES to enable partition overrides. Set to NO to disable partition overrides.</td>
</tr>
<tr>
<td>ISSESSIONVARSALLOWED</td>
<td>Optional</td>
<td>Set to YES to allow session variables in the attribute. Set to NO to disable session variables.</td>
</tr>
<tr>
<td>ISSERVERVARSALLOWED</td>
<td>Optional</td>
<td>Set to YES to allow server variables in the attribute. Set to NO to disable server variables.</td>
</tr>
<tr>
<td>ISREQUIRED</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>ISVARPREFIXALLOWED</td>
<td>Optional</td>
<td>Set to YES to enable variable prefixes for the attribute. Set to NO to disable variable prefixes for the attribute.</td>
</tr>
<tr>
<td>ISVARFULLNAMEALLOWED</td>
<td>Optional</td>
<td>Set to YES to enable variable full names for the attribute. Set to NO to disable variable full names for the attribute.</td>
</tr>
<tr>
<td>VARIABLEPREFIX</td>
<td>Optional</td>
<td>Variable prefix for the extension or connection attribute.</td>
</tr>
<tr>
<td>MINVALUE</td>
<td>Optional</td>
<td>Minimum value for the extension or connection attribute.</td>
</tr>
<tr>
<td>MAXVALUE</td>
<td>Optional</td>
<td>Maximum value for the extension or connection attribute.</td>
</tr>
<tr>
<td>GROUPID</td>
<td>Optional</td>
<td>Identifier for the group to which the extension or connection attribute belongs. You can assign a number from 1 to 16 as group ID.</td>
</tr>
<tr>
<td>GROUPPOLICY</td>
<td>Optional</td>
<td>Defines the number of attributes that can be in one group. Set to one of the following values: - NONE - EXACTLYONE - ATMOSTONE.</td>
</tr>
<tr>
<td>ISDEPRECATED</td>
<td>Optional</td>
<td>Set to YES if this attribute is deprecated. Default is NO.</td>
</tr>
</tbody>
</table>
Table 4-12 shows the possible values for the TYPE attribute and the corresponding DATATYPE values:

<table>
<thead>
<tr>
<th>TYPE</th>
<th>DATATYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOLEAN</td>
<td>NUMBER</td>
</tr>
<tr>
<td>SQL</td>
<td>STRING</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>STRING</td>
</tr>
<tr>
<td>FILENAME</td>
<td>STRING</td>
</tr>
<tr>
<td>PROPERTY</td>
<td>NUMBER or STRING</td>
</tr>
</tbody>
</table>

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

For more information about the MULTIVALUEATTRIBUTE element, see “MULTIVALUEATTRIBUTE Element” on page 32.

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 32.

CLASS Element

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

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

Table 4-13 describes the attributes of the CLASS element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Required</td>
<td>Name of the Java library class.</td>
</tr>
</tbody>
</table>
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.

Table 4-14 describes the attributes of the ALLOWEDDBTYPE element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBTYPE</td>
<td>Required</td>
<td>ID of the DBTYPE you want to use with the session extension. Informatica</td>
</tr>
<tr>
<td></td>
<td></td>
<td>provides a list of predefined DBTYPE IDs for known databases. For a list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of the DBTYPE IDs for databases, see Table 4-15.</td>
</tr>
<tr>
<td>ISDEFAULT</td>
<td>Optional</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 4-15 shows the DBTYPE IDs for databases:

<table>
<thead>
<tr>
<th>Database</th>
<th>DBTYPE ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sybase</td>
<td>2</td>
</tr>
<tr>
<td>Oracle</td>
<td>3</td>
</tr>
<tr>
<td>Informix</td>
<td>4</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>5</td>
</tr>
<tr>
<td>IBM DB2</td>
<td>6</td>
</tr>
<tr>
<td>Flatfile</td>
<td>7</td>
</tr>
<tr>
<td>ODBC</td>
<td>8</td>
</tr>
<tr>
<td>XML</td>
<td>12</td>
</tr>
<tr>
<td>Teradata</td>
<td>15</td>
</tr>
</tbody>
</table>

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.
Table 4-16 describes the attributes of the CONNECTIONREFERENCE element:

**Table 4-16. Attributes for the CONNECTIONREFERENCE Element**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Required</td>
<td>Name of the connection.</td>
</tr>
<tr>
<td>CONNECTIONNUMBER</td>
<td>Required</td>
<td>Priority of the connection. 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.</td>
</tr>
<tr>
<td>ISSESSIONOVERRIDABLE</td>
<td>Optional</td>
<td>Set to YES to enable session overrides. Set to NO to disable session overrides.</td>
</tr>
<tr>
<td>ISPARTITIONOVERRIDABLE</td>
<td>Optional</td>
<td>Set to YES to enable partition overrides. Set to NO to disable partition overrides.</td>
</tr>
<tr>
<td>REFERENCELEVEL</td>
<td>Required</td>
<td>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.</td>
</tr>
<tr>
<td>ISREQUIRED</td>
<td>Optional</td>
<td>Set to YES to make CONNECTIONREFERENCE a requirement for the extension. Set to NO to make it optional.</td>
</tr>
<tr>
<td>ISSESSIONVARSALLOWED</td>
<td>Optional</td>
<td>Set to YES to allow session variables for the extension. Set to NO to disable session variables.</td>
</tr>
<tr>
<td>ISVARPREFIXALLOWED</td>
<td>Optional</td>
<td>Set to YES to enable variable prefixes for the extension. Set to NO to disable variable prefixes for the extension.</td>
</tr>
<tr>
<td>ISVARFULLNAMEALLOWED</td>
<td>Optional</td>
<td>Set to YES to enable variable full names for the extension. Set to NO to disable variable prefixes for the extension.</td>
</tr>
<tr>
<td>VARIABLEPREFIX</td>
<td>Optional</td>
<td>Variable prefix for the extension.</td>
</tr>
<tr>
<td>ISDATACONNECTION</td>
<td>Optional</td>
<td>Set to YES to indicate that the extension is a connection to a data source or target.</td>
</tr>
</tbody>
</table>

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.
Table 4-17 describes the attributes of the ALLOWEDCONNECTION element:

### Table 4-17. Attributes for the Element ALLOWEDCONNECTION

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECTIONTYPE</td>
<td>Required</td>
<td>Type of connection. Set to one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- RELATIONAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- APPLICATION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FTP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- EXTERNALLOADER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- QUEUE.</td>
</tr>
<tr>
<td>CONNECTIONSUBTYPE</td>
<td>Required</td>
<td>Identifier for the connection subtype. For a list of predefined connection subtype IDs, see Table 4-18.</td>
</tr>
<tr>
<td>ISDEFAULT</td>
<td>Optional</td>
<td>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. If CONNECTIONNUMBER attribute for the parent CONNECTIONREFERENCE element is set to 1, set this attribute to YES.</td>
</tr>
<tr>
<td>SUPPORTPARTITIONS</td>
<td>Optional</td>
<td>Set to YES if partitions can use the connection. Set to NO if partitions cannot use the connection. If you set this attribute to NO, do not set the ISFORALLPARTITIONS attribute to YES.</td>
</tr>
<tr>
<td>ISFORALLPARTITIONS</td>
<td>Optional</td>
<td>Set to YES if all partitions can use the connection. Set to NO if not all partitions can use the connection.</td>
</tr>
</tbody>
</table>

Table 4-18 shows the predefined IDs for the CONNECTIONSUBTYPE attribute:

### Table 4-18. Predefined IDs for the CONNECTIONSUBTYPE Attribute

<table>
<thead>
<tr>
<th>Connection Subtype</th>
<th>Type of Connection</th>
<th>Connection Subtype ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td>Relational/External Loader</td>
<td>101</td>
</tr>
<tr>
<td>Sybase</td>
<td>Relational/External Loader</td>
<td>102</td>
</tr>
<tr>
<td>Informix</td>
<td>Relational</td>
<td>103</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>Relational</td>
<td>104</td>
</tr>
<tr>
<td>DB2</td>
<td>Relational</td>
<td>105</td>
</tr>
<tr>
<td>ODBC</td>
<td>Relational</td>
<td>106</td>
</tr>
<tr>
<td>Teradata</td>
<td>Relational</td>
<td>107</td>
</tr>
<tr>
<td>SAP BW</td>
<td>Application</td>
<td>100001</td>
</tr>
<tr>
<td>SAP R/3</td>
<td>Application</td>
<td>100101</td>
</tr>
<tr>
<td>PeopleSoft on Oracle</td>
<td>Application</td>
<td>100201</td>
</tr>
<tr>
<td>PeopleSoft on Sybase</td>
<td>Application</td>
<td>100202</td>
</tr>
<tr>
<td>PeopleSoft on Informix</td>
<td>Application</td>
<td>100203</td>
</tr>
<tr>
<td>PeopleSoft on Microsoft SQL Server</td>
<td>Application</td>
<td>100204</td>
</tr>
<tr>
<td>PeopleSoft on DB2</td>
<td>Application</td>
<td>100205</td>
</tr>
<tr>
<td>Siebel on Oracle</td>
<td>Application</td>
<td>100301</td>
</tr>
<tr>
<td>Siebel on Sybase</td>
<td>Application</td>
<td>100302</td>
</tr>
<tr>
<td>Siebel on Informix</td>
<td>Application</td>
<td>100303</td>
</tr>
<tr>
<td>Siebel on Microsoft SQL Server</td>
<td>Application</td>
<td>100304</td>
</tr>
<tr>
<td>Siebel on DB2</td>
<td>Application</td>
<td>100305</td>
</tr>
</tbody>
</table>
The ALLOWEDCONNECTION element has the following child element:

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.

Table 4-19 describes the attributes of the HIDDENCONNECTIONATTRIBUTETOEXTENSION element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNXATTRIBUTEID</td>
<td>Required</td>
<td>Attribute ID of the connection attribute to hide from a session extension.</td>
</tr>
</tbody>
</table>

**Table 4-18. Predefined IDs for the CONNECTIONSUBTYPE Attribute**

<table>
<thead>
<tr>
<th>Connection Subtype</th>
<th>Type of Connection</th>
<th>Connection Subtype ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teradata</td>
<td>External Loader</td>
<td>103</td>
</tr>
<tr>
<td>Teradata_TPUMP</td>
<td>External Loader</td>
<td>104</td>
</tr>
<tr>
<td>DB2EE_LOAD</td>
<td>External Loader</td>
<td>105</td>
</tr>
<tr>
<td>DB2EEE_AUTOLOAD</td>
<td>External Loader</td>
<td>106</td>
</tr>
<tr>
<td>Teradata_FASTLOAD</td>
<td>External Loader</td>
<td>107</td>
</tr>
<tr>
<td>Teradata_WB</td>
<td>External Loader</td>
<td>108</td>
</tr>
<tr>
<td>MQ Series</td>
<td>Queue</td>
<td>101</td>
</tr>
</tbody>
</table>

The ALLOWEDCONNECTION element has the following child element:

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.

Table 4-19 describes the attributes of the HIDDENCONNECTIONATTRIBUTETOEXTENSION element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNXATTRIBUTEID</td>
<td>Required</td>
<td>Attribute ID of the connection attribute to hide from a session extension.</td>
</tr>
</tbody>
</table>

**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.
Table 4-20 describes the attributes of the CONNECTION element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/ Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Required</td>
<td>Name of the connection.</td>
</tr>
<tr>
<td>CONNECTIONTYPE</td>
<td>Required</td>
<td>Type of connection. Set to one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- RELATIONAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- APPLICATION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- FTP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- EXTERNALLOADER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- QUEUE</td>
</tr>
<tr>
<td>CONNECTIONSUBTYPE</td>
<td>Required</td>
<td>Identifier for the connection subtype. For a list of predefined connection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>subtype IDs, see Table 4-18.</td>
</tr>
<tr>
<td>RUNTIMECHECK</td>
<td>Optional</td>
<td>License key for validating the plug-in. This attribute is currently not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>used.</td>
</tr>
<tr>
<td>HASUSERNAME</td>
<td>Optional</td>
<td>Set to YES if the connection requires a username. Set to NO if the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>connection does not require a username.</td>
</tr>
<tr>
<td>HASUSERPASSWORD</td>
<td>Optional</td>
<td>Set to YES if the connection requires a password. Set to NO if the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>connection does not require a password.</td>
</tr>
<tr>
<td>HASCONNECTSTRING</td>
<td>Optional</td>
<td>Set to YES if the connection requires a connect string. Set to NO if the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>connection does not require a connect string.</td>
</tr>
<tr>
<td>HASCODEPAGE</td>
<td>Optional</td>
<td>Set to YES if the connection has a code page. Set to NO if the connection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>does not have a code page.</td>
</tr>
<tr>
<td>HASPERMISSIONS</td>
<td>Optional</td>
<td>Set to YES to display the permission properties in the Designer. Set to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO to disable the permission properties.</td>
</tr>
<tr>
<td>ISRELATIONAL</td>
<td>Optional</td>
<td>Indicates whether this is an SQL connection to a relational database. Set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to YES if the connection is relational. Set to NO if the connection is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>non-relational.</td>
</tr>
<tr>
<td>COMPONENTVERSION</td>
<td>Required</td>
<td>Version of the CONNECTION. Indicates that the attributes of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONNECT have changed since the previous version. Use this attribute to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>keep track of updates to the V element.</td>
</tr>
</tbody>
</table>

The CONNECTION element has the following child elements:

- ATTRIBUTE
  For more information, see "ATTRIBUTE Element" on page 34.
- LIBRARY
  For more information, see "LIBRARY Element" on page 36.

**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:

```xml
<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.

Table 4-21 describes the attributes of the DBTYPETOEXTENSION element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTENSIONTYPE</td>
<td>Required</td>
<td>Set to READER if the referenced session extension is a reader. Set to WRITER if the referenced session extension is a writer.</td>
</tr>
<tr>
<td>EXTENSIONSUBTYPE</td>
<td>Required</td>
<td>Set the session extension subtype for the extension.</td>
</tr>
<tr>
<td>DBTYPE</td>
<td>Required</td>
<td>Set to the ID of the referenced DBTYPE. For a list of predefined DBTYPE IDs, see Table 4-15 on page 37.</td>
</tr>
<tr>
<td>ISDEFAULT</td>
<td>Optional</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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:

```xml
<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.

Table 4-22 describes the attributes of the CONNECTIONTOEXTENSION element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTENSIONTYPE</td>
<td>Required</td>
<td>Set to READER if the referenced session extension is a reader. Set to WRITER if the referenced session extension is a writer.</td>
</tr>
<tr>
<td>EXTENSIONSUBTYPE</td>
<td>Required</td>
<td>Set the session extension subtype for the extension.</td>
</tr>
<tr>
<td>CONNECTIONTYPE</td>
<td>Required</td>
<td>Connection type of the extension. Set to one of the following connection types: RELATIONAL, APPLICATION, FTP, EXTERNALLOADER, QUEUE.</td>
</tr>
<tr>
<td>CONNECTIONSUBTYPE</td>
<td>Required</td>
<td>Identifier for the connection subtype. For a list of predefined connection subtype IDs, see Table 4-18.</td>
</tr>
</tbody>
</table>
The CONNECTIONTOEXTENSION element has the following child elements:

- **HIDDENCONNECTIONATTRIBUTETOEXTENSION**
  For more information, see “HIDDENCONNECTIONATTRIBUTETOEXTENSION Element” on page 40
- **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.

Table 4-23 describes the attributes of the HIDDENEXTENSIONATTRIBUTETOCONNECTION element:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTNATTRIBUTEID</td>
<td>Required</td>
<td>Attribute ID of session extension attribute to hide from a connection.</td>
</tr>
</tbody>
</table>

### 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.
Table 4-24 describes the attributes of the MEDOMAIN element:

Table 4-24. Attributes for the MEDOMAIN Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Required</td>
<td>Enter a name for the metadata extension domain.</td>
</tr>
<tr>
<td>ID</td>
<td>Required</td>
<td>Enter an ID from the range of metadata extension domain IDs obtained from Informatica.</td>
</tr>
<tr>
<td>KEY</td>
<td>Optional</td>
<td>Enter an encrypted domain key for the metadata extension domain. You also use this key to access private metadata extensions. Use the pmpasswd &lt;password&gt; command to encrypt a password.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Optional</td>
<td>Enter a description for the metadata extension domain.</td>
</tr>
<tr>
<td>CLIENTVISIBLE</td>
<td>Optional</td>
<td>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.</td>
</tr>
<tr>
<td>COMPONENTVERSION</td>
<td>Required</td>
<td>Enter the version of the MEDOMAIN. This allows you to keep track of updates to the MEDOMAIN element.</td>
</tr>
</tbody>
</table>

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

```xml
<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**

For more information, see “LIBRARY Element” on page 36.

**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.

Table 4-25 describes the attributes of the MEDEFINITION element:

Table 4-25. Attributes for the MEDEFINITION Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Required</td>
<td>Enter a name for the metadata extension.</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>Required</td>
<td>Enter a datatype for the metadata extension. You can enter STRING, NUMERIC, BOOLEAN, or XML. Note: If you set the DATATYPE attribute to XML or STRING, you must set the MAXLENGTH attribute to a value greater than 0.</td>
</tr>
<tr>
<td>MAXLENGTH</td>
<td>Optional</td>
<td>Enter the maximum length for the metadata extension. You can specify a value up to 2,147,483,647. Note: If you set the DATATYPE attribute to XML or STRING, you must set the MAXLENGTH attribute to a value greater than 0.</td>
</tr>
</tbody>
</table>
### Table 4-25. Attributes for the MEDEFINITION Element

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBTYPE</td>
<td>Required</td>
<td>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.</td>
</tr>
<tr>
<td>OBJECTTYPE</td>
<td>Required</td>
<td>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.</td>
</tr>
<tr>
<td>DEFAULTVALUE</td>
<td>Optional</td>
<td>Enter the default value for the metadata extension.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Optional</td>
<td>Enter a description for the metadata extension.</td>
</tr>
<tr>
<td>ISSHAREREAD</td>
<td>Optional</td>
<td>Set to YES to enable shared reading for the metadata extension. Set to NO to disable share reading for the metadata extension.</td>
</tr>
<tr>
<td>ISSHAREWRITE</td>
<td>Optional</td>
<td>Set to YES to enable shared writing for the metadata extension. Set to NO to disable share writing for the metadata extension.</td>
</tr>
<tr>
<td>ISCLIENTVISIBLE</td>
<td>Optional</td>
<td>Set to YES to enable the Designer to display the metadata extension. Set to NO to disable the Designer from displaying the metadata extension.</td>
</tr>
<tr>
<td>ISCLIENTEDITABLE</td>
<td>Optional</td>
<td>Set to YES to enable the Designer to edit the metadata extension. Set to NO to disable the Designer from displaying the metadata extension.</td>
</tr>
</tbody>
</table>
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.

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 /javlib 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plugin Id</td>
<td>305050</td>
</tr>
<tr>
<td>Plugin Name</td>
<td>PWX JDBC</td>
</tr>
<tr>
<td>Plugin Vendor</td>
<td>INFA</td>
</tr>
<tr>
<td>PC Version</td>
<td>8.7</td>
</tr>
</tbody>
</table>

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:

```xml
<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 some of the DATATYPE definitions in the pmJDBC.xml file:

```xml
<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"/>
```
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:

<table>
<thead>
<tr>
<th>JDBC Datatypes</th>
<th>PowerCenter ODBC Datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>SQL_CHAR</td>
</tr>
<tr>
<td>CHAR</td>
<td>SQL_WCHAR</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>SQL_VARCHAR</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>SQL_LONGVARCHAR</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>SQL_WVARCHAR</td>
</tr>
<tr>
<td>LONGVARCHAR</td>
<td>SQL_LONGVARCHAR</td>
</tr>
<tr>
<td>LONGVARCHAR</td>
<td>SQL_WLONGVARCHAR</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>SQL_DECIMAL</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>SQL_NUMERIC</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>SQL_DECIMAL</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>SQL_MONEY</td>
</tr>
<tr>
<td>BIT</td>
<td>SQL_BIT</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>SQL_BIT</td>
</tr>
<tr>
<td>TINYINT</td>
<td>SQL_TINYINT</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SQL_SMALLINT</td>
</tr>
<tr>
<td>INTEGER</td>
<td>SQL_INTEGER</td>
</tr>
<tr>
<td>BIGINT</td>
<td>SQL_BIGINT</td>
</tr>
<tr>
<td>REAL</td>
<td>SQL_FLOAT</td>
</tr>
<tr>
<td>REAL</td>
<td>SQL_REAL</td>
</tr>
<tr>
<td>FLOAT</td>
<td>SQL_DOUBLE</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>SQL_DOUBLE</td>
</tr>
<tr>
<td>BINARY</td>
<td>SQL_BINARY</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>SQL_VARBINARY</td>
</tr>
<tr>
<td>LONGVARBINARY</td>
<td>SQL_LONGVARBINARY</td>
</tr>
<tr>
<td>LONGVARBINARY</td>
<td>SQL_IDENTITY</td>
</tr>
<tr>
<td>DATE</td>
<td>SQL_DATE</td>
</tr>
<tr>
<td>TIME</td>
<td>SQL_TIME</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>SQL_TIMESTAMP</td>
</tr>
</tbody>
</table>
The following code shows the DBTYPETOWIDGETATTR element definition in the pmJDBC.xml file:

```xml
<DBTYPETOWIDGETATTR
    OBJECTTYPE="TARGET"
    WIDGETATTRIBUTENAME="Load Scope"
    ISREADONLY="YES"
    ISDISABLED="YES"
    IS HIDDEN="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:

```xml
<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:

<table>
<thead>
<tr>
<th>Session Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Defined Join</td>
<td>Specifies a custom join.</td>
</tr>
<tr>
<td>Number Of Sorted Ports</td>
<td>The number of ports to be used for sorting.</td>
</tr>
<tr>
<td>Select Distinct</td>
<td>Select distinct values only.</td>
</tr>
<tr>
<td>Tracing Level</td>
<td>Tracing level for the log messages to send to the session log.</td>
</tr>
<tr>
<td>Pre SQL</td>
<td>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.</td>
</tr>
<tr>
<td>Post SQL</td>
<td>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.</td>
</tr>
<tr>
<td>SQL Query</td>
<td>Overrides the default SQL query with a custom SQL query.</td>
</tr>
<tr>
<td>Source Filter</td>
<td>Sets a filter for the rows in the data source.</td>
</tr>
</tbody>
</table>

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

```xml
<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:

<table>
<thead>
<tr>
<th>Session Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>Insert data into the target.</td>
</tr>
</tbody>
</table>
| Update            | Update strategy to use when updating target data. You can use one of the following update strategies:  
- Update as Update. Perform an update on the target.  
- Update as Insert. Perform an insert on the target.  
- None. Perform no operation on the target.  
- 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:

```xml
<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:

<table>
<thead>
<tr>
<th>Connection Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC Driver Name</td>
<td>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 For the Java DB database, the JDBC driver name is &quot;org.apache.derby.jdbc.ClientDriver&quot;</td>
</tr>
<tr>
<td>Connection Environment SQL</td>
<td>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. This is an optional attribute.</td>
</tr>
<tr>
<td>Transaction Environment SQL</td>
<td>Transaction environment SQL to run each time a new transaction is started in the external database. This is an optional attribute.</td>
</tr>
<tr>
<td>Connection Retry Period</td>
<td>Length of time in seconds that the Integration Service attempts to re-connect to the database if the connection fails.</td>
</tr>
</tbody>
</table>

The following screenshot shows the connection dialog box for the JDBC connection to the Java DB database:
The following code shows a CONNECTION element defined in the pmJDBC.xml file:

```xml
<CONNECTION NAME ="PWX JDBC"
    HASCODEPAGE ="YES"
    HASUSERNAME ="YES"
    CONNECTIONTYPE ="RELATIONAL"
    HASPERMISSIONS ="NO"
    HASUSERPASSWORD ="YES"
    COMNPCOMPONENTVERSION ="1.0.0"
    HASCONNECTSTRING ="YES"
    CONNECTIONSUBTYPE ="305050">
    <ATTRIBUTE ID ="1"
        NAME ="JDBC Driver Name"
    />
</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:

```xml
<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:

<table>
<thead>
<tr>
<th>Metadata Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Defined Join</td>
<td>Specifies a custom join.</td>
</tr>
<tr>
<td>Number Of Sorted Ports</td>
<td>The number of ports to be used for sorting.</td>
</tr>
<tr>
<td>Select Distinct</td>
<td>Select distinct values only.</td>
</tr>
<tr>
<td>SQL Query</td>
<td>Overrides the default SQL query with a custom SQL query.</td>
</tr>
<tr>
<td>Source Filter</td>
<td>Sets a filter for the rows in the data source.</td>
</tr>
</tbody>
</table>
The following screenshot shows the metadata extensions for a JDBC Source Qualifier:

![Metadata Extensions Screenshot]

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

```xml
<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:

![Diagram showing sequence of calls during a writer session]

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:

<table>
<thead>
<tr>
<th>JDBC Datatypes</th>
<th>PowerCenter Datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char</td>
<td>String</td>
</tr>
<tr>
<td>Varchar</td>
<td>String</td>
</tr>
<tr>
<td>LongVarchar</td>
<td>String</td>
</tr>
<tr>
<td>Numeric</td>
<td>Decimal</td>
</tr>
<tr>
<td>Decimal</td>
<td>Decimal</td>
</tr>
<tr>
<td>Real</td>
<td>Float</td>
</tr>
<tr>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>Float</td>
<td>Double</td>
</tr>
<tr>
<td>Date</td>
<td>Date/time</td>
</tr>
<tr>
<td>Int</td>
<td>Integer</td>
</tr>
<tr>
<td>Small int</td>
<td>Small integer</td>
</tr>
<tr>
<td>Long</td>
<td>Text</td>
</tr>
<tr>
<td>Binary</td>
<td>Binary</td>
</tr>
<tr>
<td>Varbinary</td>
<td>Binary</td>
</tr>
<tr>
<td>LongVarBinary</td>
<td>Longvarbinary</td>
</tr>
<tr>
<td>Bit</td>
<td>String (Precision 1)</td>
</tr>
<tr>
<td>Boolean</td>
<td>String (Precision 1)</td>
</tr>
<tr>
<td>Blob</td>
<td>Binary</td>
</tr>
<tr>
<td>Clob</td>
<td>Text</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>JDBC Datatypes</th>
<th>PowerCenter Datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>short</td>
</tr>
<tr>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>Short / Small int</td>
<td>X</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>JDBC Datatypes</th>
<th>PowerCenter Datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>short</td>
</tr>
<tr>
<td>Int</td>
<td>X</td>
</tr>
<tr>
<td>long</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>X</td>
</tr>
<tr>
<td>Date/Time</td>
<td></td>
</tr>
<tr>
<td>Binary</td>
<td></td>
</tr>
</tbody>
</table>
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:

```java
/** Message catalog initialization*/
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 logs it to the session log.

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

```java
//get the sq instance
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:

```java
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:

```java
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.
   For more information, see “Informatica Development Platform” on page 7.

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.
Design API

This chapter includes the following topics:
- Overview, 63
- Browsing Metadata in PowerCenter Repository, 63
- Creating Objects, 64
- Exporting and Importing Metadata in the Repository, 72

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 Java DB adapter 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:

```java
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
    Vector folders = rep.getFolder(new NameFilter() {
        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++) {
        Vector listOfSources = ((Folder)folders.get(i)).getSource(); // get the list of sources
        System.out.println("***** List of Sources *****");
        for(int j=0; j < listOfSources.size(); j++) {
            System.out.println(((Source)listOfSources.get(j)).getName());
        }
    }

    for(int i=0; i < folderSize; i++) {
        Vector listOfTargets = ((Folder)folders.get(i)).getTarget(); // get the list of targets
        System.out.println("***** List of Targets *****");
        for(int j=0; j < listOfTargets.size(); j++) {
            System.out.println(((Target)listOfTargets.get(j)).getName());
        }
    }

    for(int i=0; i < folderSize; i++) {
        Vector listOfMapplets = ((Folder)folders.get(i)).getMapplets(); // get the list of mapplets
        System.out.println("***** List of Mapplets *****");
        for(int j=0; j < listOfMapplets.size(); j++) {
            System.out.println(((Mapplet)listOfMapplets.get(j)).getName());
        }
    }

    for(int i=0; i < folderSize; i++) {
        Vector listOfMappings = ((Folder)folders.get(i)).getMappings(); // get the list of mappings
        System.out.println("***** List of Mappings *****");
        for(int j=0; j < listOfMappings.size(); j++) {
            System.out.println(((Mapping)listOfMappings.get(j)).getName());
        }
    }
}
```

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:

```java
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.

```java
protected Source createOrderDetailSource() {
  Vector fields = new Vector();
  Field field1 = new Field("OrderID", "OrderID", "", DataTypeConstants.INTEGER, "10", "0",
                           FieldConstants.FOREIGN_KEY, Field.FIELDTYPE_SOURCE, false);
  fields.add(field1);
  Field field2 = new Field("ProductID", "ProductID", "", DataTypeConstants.INTEGER, "10", "0",
                           FieldConstants.FOREIGN_KEY, Field.FIELDTYPE_SOURCE, false);
  fields.add(field2);
  Field field3 = new Field("UnitPrice", "UnitPrice", "", DataTypeConstants.DECIMAL, "28", "4",
                           FieldConstants.NOT_A_KEY, Field.FIELDTYPE_SOURCE, false);
  fields.add(field3);
  Field field4 = new Field("Quantity", "Quantity", "", DataTypeConstants.INTEGER, "10", "0",
                           FieldConstants.NOT_A_KEY, Field.FIELDTYPE_SOURCE, false);
  fields.add(field4);
  Field field5 = new Field("Discount", "Discount", "", DataTypeConstants.INTEGER, "10", "0",
                           FieldConstants.NOT_A_KEY, Field.FIELDTYPE_SOURCE, false);
  fields.add(field5);
  Field field6 = new Field("VarcharFld", "VarcharFld", "", DataTypeConstants.VARCHAR, "5", "0",
                           FieldConstants.NOT_A_KEY, Field.FIELDTYPE_SOURCE, false);
  fields.add(field6);
  Field field7 = new Field("Varchar2Fld", "Varchar2Fld", "", DataTypeConstants.VARCHAR2, "5", "0",
                           FieldConstants.NOT_A_KEY, Field.FIELDTYPE_SOURCE, false);
  fields.add(field7);

  ConnectionInfo info = getFlatFileConnectionInfo();
  info.setPropInfo(ConnectionPropsConstants.SOURCE_FILENAME, "Order_Details.csv");
  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(SourceTargetTypes.FLATFILE_TYPE);
  infoProps.getConnProps().setProperty(ConnectionPropsConstants.FLATFILE_SKIPROWS, "1");
  infoProps.getConnProps().setProperty(ConnectionPropsConstants.FLATFILE_DELIMITERS, ",");
  infoProps.getConnProps().setProperty(ConnectionPropsConstants.FLATFILE_DATETIME_FORMAT, "A  21 yyyy/mm/dd hh24:mi:ss");
  infoProps.getConnProps().setProperty(ConnectionPropsConstants.FLATFILE_QUOTE_CHARACTER, "DOUBLE");
```
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:

![Diagram showing input and output sets]

**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 following objects:

- **Port propagation context object.** Context object for passing 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.
    - **Ketchup 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. For more information about expression patterns, see “Sample Patterns for Regular Expressions for Port Propagation” on page 101.

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

    ```java
    // 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" });

    Vector vInputSets = new Vector();
    vInputSets.add(new InputSet(dsqRS, dsqRSContext)); // remove
    // Manufacturer_id
    vInputSets.add(new InputSet(lookupRS, lkpRSContext)); // propagate
    // only
    // Manufacturer_Name
    ```
The following code example shows how to use the exclude rule to propagate ports:

```java
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.

```java
public Vector getLinkFields() {
    Vector fields = new Vector();
    Field field1 = new Field( "EmployeeID", "EmployeeID", "", DataTypeConstants.INTEGER,
        "10", "0", FieldConstants.PRIMARY_KEY, Field.FIELDTYPE_SOURCE, true );
    Fields.add( field1 );
    Field field2 = new Field( "LastName", "LastName", "", DataTypeConstants.STRING, "20", "0", FieldConstants.NOT_A_KEY, Field.FIELDTYPE_SOURCE, false );
    Fields.add( field2 );
    Field field3 = new Field( "FirstName", "FirstName", "", DataTypeConstants.STRING, "10", "0", FieldConstants.NOT_A_KEY, Field.FIELDTYPE_SOURCE, false );
    Fields.add( field3 );
    return fields;
}
```

```java
// create link fields
Vector 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 concatenated to produce a new
// field fullName
String expr = "string(80, 0) fullName= firstName || lastName";
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:

![Mapping Diagram]

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

```java
// create a stored procedure transformation
Vector vTransformFields = new Vector();
Field field1 = new Field( "RetValue", "RetValue", "This is return value",
```
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
- Expression
- External Procedure
- Filter
- HTTP
- Input
- Joiner
- Lookup
- Normalizer
- Rank
- Router
- Sequence Generator
- Sorter
- Source Qualifier
The following sample code shows how 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.

```java
// 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" });

Vector vInputSets = new Vector();
vInputSets.add(new InputSet(dsqRS, dsqRSContext)); // remove
// Manufacturer_id
vInputSets.add(new InputSet(lookupRS, lkpRSContext)); // propagate
// only
// Manufacturer_Name

// 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:

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

```java
/**
 * 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:

```java
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.CONTROL_OPTION_VALUE_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_TOLEVEL_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("$$var1",PowerMartDataTypeConstants.INTEGER,"1","var1 ");
    WorkflowVariable wfVar2 = new WorkflowVariable("$$var2",PowerMartDataTypeConstants.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(command);
    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:

```java
Properties prop = ConnectionAttributes.getDefaultRelationalProperties(ConnectionAttributes.DB_CONN_TYPE_ORACLE);
prop.list(System.out);

prop.setProperty(ConnectionAttributes.CONN_ATTR_CONNECTION_NAME, "myConn");
prop.setProperty(ConnectionAttributes.CONN_ATTR_USER_NAME, "sramamoo");
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 (ConnectionUpdateFailedException e) {
    e.printStackTrace();
} catch (ConnectionFailedException 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 code shows the contents of a sample `pcconfig.properties` file that includes attributes for connecting to the domain and repository.

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

```java
public void generateOutput() throws Exception {
    MapFwkOutputContext outputContext = new MapFwkOutputContext(
        MapFwkOutputContext.OUTPUT_FORMAT_XML,
        MapFwkOutputContext.OUTPUT_TARGET_FILE, mapFileName);
    try {
        initializeLocalProps();
    } 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 );
    }
    rep.save(outputContext, doImport);
    System.out.println( "Metadata generated in " + mapFileName );
}
```
Chapter 7

Design API Example

This chapter includes the following topics:
♦ Overview, 75
♦ Overview of the Process Flow, 75
♦ Design API Methods, 79
♦ Installing and Running the Sample Application, 90

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 1.5, 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:

```java
// 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 jakarta command line interpreter // needs common.cli.jar!
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 Integration 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 = SESSIONSTARTTIME – 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:

<table>
<thead>
<tr>
<th>OP</th>
<th>SRC COPY</th>
<th>SRC, SRC_COPY</th>
<th>Product</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

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:

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:

\[ \text{valid.to} \geq '31.12.9999' \]

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:

```java
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());

    // 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 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:

```java
// 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:

```java
/**
 * 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 separate array
    pkColNameList << name
    // set the attributes for the column as needed 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 complete 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:

```java
/**
 * 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 pconfig.properties file.

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

```java
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:

```java
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:

```java
// 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:

```java
Vector fields = new Vector();
```
// Iterate over all columns of given table, 
// starting with pk, then with payload columns 
for col in tab.mapColumns(it) 
    // Create a field for this column 
    Field f = new Field(it, it, col.remarks?page: col.remarks, col.typeName.toLowerCase(), col.columnSize, col.decimalDigits, col.isPrimaryKey?FieldConstants.PRIMARY_KEY:FieldConstants.NOT_A_KEY, col.isNullable?false:true) 
    // TODO: Check for inconsistencies: refTable set, refField set 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) 

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
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:

```java
/**
 * 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 instance 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:

```java
// 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 instance. Setting 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:

```java
// 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 agetContextForAllIncludeCols() 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:

```java
// 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:

```java
// collect a set (vector) for the added transform fields in the coming expression
Vector vFields = new Vector()
vFields.add(new TransformField("date (29, 9) NOW= SESSSTARTTIME"))
vFields.add(new TransformField("date (29, 9) JUSTBEFORESESSIONSTART= ADD_TO_DATE(SESSSTARTTIME,'NS',-1)"))
vFields.add(new TransformField("date (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= \$PAMRepositoryServiceName"))
```
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:

```java
// 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:

```java
/*
 * 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 ""
    };
```
Design API Methods  87

```java
// 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.

```java
// 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", "pworkflow", "preposervice")
.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:

```java
// 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:

```java
/**
 * 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 {
        def fName = it.getName()
        // take field name
        def f = fromRS.getField(fName + 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)
        }
    } else {
        // check for valid pm fields
        if (fName == /pmmapping/ ||
            fName == /pmsession/ ||
            fName == /pmworkflow/ ||
            fName == /pmreposervice/)
        {
            def f = fromRS.getField(fName + "_new"+suffix)
            if (f) {
                linkMap.put(f, it)
            }
        }
    }
    return linkMap;
}
```
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. For more information, see “Sample Patterns for Regular Expressions for Port Propagation” on page 101.

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, sourceConnectionName)
    source_tgt.getConnInfo().getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME, targetConnectionName)
    insertTarget.getConnInfo().getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME, targetConnectionName)
    updateTarget.getConnInfo().getConnProps().setProperty(ConnectionPropsConstants.CONNECTIONNAME, 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());
    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:

```java
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. For more information, see “Exporting and Importing Metadata in the Repository” on page 72.

The following sample code shows the generateOutput() method:

```java
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 1.5.** 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:

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

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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>JDBC URL. Connection string to the PowerCenter repository database. Default is jdbc:oracle:thin:@localhost:1521:ORCL.</td>
</tr>
<tr>
<td>userName</td>
<td>User name for the user account to log in to the repository.</td>
</tr>
<tr>
<td>password</td>
<td>Password for the user account to log in to the repository.</td>
</tr>
<tr>
<td>driverName</td>
<td>Name of the JDBC driver. Default is oracle.jdbc.OracleDriver.</td>
</tr>
<tr>
<td>defaultSourceConnection</td>
<td>PowerCenter connection object for the source.</td>
</tr>
<tr>
<td>defaultTargetConnection</td>
<td>PowerCenter connection object for the target.</td>
</tr>
<tr>
<td>logLevel</td>
<td>Level of error messages to write to the log file. Default is INFO.</td>
</tr>
<tr>
<td>pcsISName</td>
<td>Name of the Integration Service to run the workflow.</td>
</tr>
<tr>
<td>defaultTableName</td>
<td>Name of the source table.</td>
</tr>
<tr>
<td>cloneSuffix</td>
<td>Suffix to append to the name of a cloned table. Default is _SCL.</td>
</tr>
</tbody>
</table>

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 1.5 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 version used to create these examples were 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.
Overview

This appendix provides examples that you can use when 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 Using the Design API

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

Using Multiple Instance 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:

```java
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" );
}
```
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:

```java
// Logic to create a DSQ Transformation using 2 sources. They
// should satisfy PKFK constraint.
Vector vSources = new Vector();
InputSet itemIS = new InputSet( itemSource );
InputSet productIS = new InputSet( productSource );
vSources.add( itemIS );
vSources.add( productIS );
TransformationContext tc = new TransformationContext( vSources );
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( "number(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 Mapplets

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

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

// create helper for mapplet
TransformHelper helperMapplet = new TransformHelper(mapplet);
outputSet = helperMapping.mapplet(mapplet, vInSets, "myMapplet");
Vector vMappRS = outputSet.getRowSets();

// write to target
mapping.writeTarget((RowSet) vMappRS.get(0), idPostTrg);
mapping.writeTarget((RowSet) vMappRS.get(1), ageTrg);
mapping.writeTarget((RowSet) vMappRS.get(2), fullNameTrg);
```

You can use a mapplet to read data from data sources that do not have native support in PowerCenter, such as application data sources like SAP and PeopleSoft and real-time data sources like JMS and MSMQ. Create a mapplet that uses a PowerExchange adapter to get data from the data source. You can then use the Design API to read data from the mapplet and write to the target.

The following sample code shows how to read data from a mapplet and write to a target:
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:

```java
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);
OutputSet outSet = helper.sourceQualifier(scSrc);
RowSet dsqRS = (RowSet)outSet.getRowSets().get(0);
```

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:

```java
//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",null);
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);
```

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:

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

StringWriter outputSummary = new StringWriter();
// validate mapping
```
try {
    connMgr.validate(mapping.getName(), RepositoryObjectConstants.OBJTYPE_MAPPING, 
    folder.getName(), true, outputSummary);
} catch (ConnectionFailedException e) {
    e.printStackTrace();
} catch (ValidationFailedException e) {

Creating Parameter Files

The following sample code shows how to create a PowerCenter session parameter file. The parameter file
contains parameters for mappings and sessions and is used when the Integration Service runs a session.

Vector listOfParams = new Vector();
MappingVariable mpVar1 = new MappingVariable("MAX", DataTypeConstants.STRING, "default value 1", "description", true, "$$var1", "20", "0", true);
MappingVariable mpVar2 = new MappingVariable("MIN", DataTypeConstants.INTEGER, "10", "description", false, "$$var2", "10", "0", true);
MappingVariable mpVar3 = new MappingVariable("MAX", DataTypeConstants.NSTRING, "default value 3", "description", true, "$var3", "10", "0", true);
MappingVariable mpVar4 = new MappingVariable("MAX", DataTypeConstants.INTEGER, "101", "description", false, "$$var4", "10", "0", true);
MappingVariable mpVar5 = new MappingVariable("MIN", DataTypeConstants.STRING, "default value 5", "description", true, "$var5", "15", "0", true);
MappingVariable mpVar6 = new MappingVariable("MAX", DataTypeConstants.NSTRING, "default value 6", "description", false, "$var6", "20", "0", true);

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

ParameterFile pmFile = new ParameterFile("C:\work\param.save");
Iterator listOfParamsIter = listOfParams.iterator();
int i=0;
while(listOfParamsIter.hasNext())
{
    pmFile.setParameterValue((String) listOfParamsIter.next(), new Integer(i).toString());
i++;
}
pmFile.save();

Creating Sources

The following sample code shows how to create a relational source object:

protected Source createJobSource() {
    Source jobSource = null;
    Vector vFields = new Vector();
    Field jobIDField = new Field( "JOB_ID", "JOB_ID", ", DataTypes.VARCHAR2, ", 10", ", 0", FieldConstants.PRIMARY_KEY, Field.FIELDTYPE_SOURCE, true );
vFields.add( jobIDField );
    Field jobTitleField = new Field( "JOB_TITLE", "JOB_TITLE", ", DataTypes.VARCHAR2, ", 35", ", 0", FieldConstants.NOT_A_KEY, Field.FIELDTYPE_SOURCE, false );
vFields.add( jobTitleField );
Field minSalField = new Field( "MIN_SALARY", "MIN_SALARY", ",",
DataTypeConstants.DECIMAL, "6", "0", FieldConstants.NOT_A_KEY, Field.FIELDTYPE_SOURCE,
false );
vFields.add( minSalField );

Field maxSalField = new Field( "MAX_SALARY", "MAX_SALARY", ",",
DataTypeConstants.DECIMAL, "6", "0", FieldConstants.NOT_A_KEY, Field.FIELDTYPE_SOURCE,
false );
vFields.add( maxSalField );

ConnectionInfo info = getRelationalConnInfo(SourceTargetTypes.RELATIONAL_TYPE_ORACLE,
"Oracle_Source" );

jobSource = new Source( "JOBS", "JOBS", "This is JOBS table", "JOBS", info );
jobSource.setFields(vFields);
return jobSource;

protected ConnectionInfo getRelationalConnInfo( int dbType, String dbName ) {
ConnectionInfo connInfo = null;
connInfo = new ConnectionInfo( dbType );
connInfo.getConnProps().setProperty( ConnectionPropsConstants.DBNAME, dbName );
return connInfo;
}

---

Defining a PowerExchange 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",
"mySourceDB", "mySource", "mySource");
SourceGroup srcGrp = new SourceGroup("ct_ALLDTYPES_SRC",null);
source.createField("DTL__CAPXRESTART1",srcGrp,"","PACKED","25","0",FieldConstants.NOT_A_KEY, Field.FIELDTYPE_SOURCE, false);
source.createField("DTL__CAPXRESTART2",srcGrp,"","DataTypeConstants.STRING,"10","0",Fi eldConstants.NOT_A_KEY, Field.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");
Vector connInfos = source.getConnInfos();
for (int i=0;i<connInfos.size();i++)
{
ConnectionInfo connInfo = (ConnectionInfo) connInfos.elementAt(i);
connInfo.getConnProps().setProperty( ConnectionPropsConstants.CONNECTIONNAME,
"myTestConnection");
connInfo.getConnProps().setProperty( ConnectionPropsConstants.DBNAME, "myDBName");
}
} catch (SourceDefinitionNotFoundException 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";
}
Creating Transformations

The following code examples show how to create different types of transformations:

**Sorter**

```java
// create a sorter Transformation
RowSet sorterRS = (RowSet) helper.sorter( dsqRS, new String[] { "FirstName", "LastName" },
        new boolean[] { true, false }, "sorter_transform" ).getRowSets().get( 0 );
```

**Router**

```java
TransformHelper helper = new TransformHelper( mapping );
// Create a TransformGroup
Vector vTransformGrp = new Vector();
TransformGroup transGrp = new TransformGroup( "LONDON_GROUP", "City = 'London'" );
vTransformGrp.add( transGrp );
transGrp = new TransformGroup( "SEATTLE_GROUP", "City = 'Seattle'" );
vTransformGrp.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, vTransformGrp,
        "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 );
```

**Union**

```java
RowSet rsGroupFld = new RowSet();
    Field field1 = new Field( "ItemId", "ItemId", "", DataTypeConstants.INTEGER,
        "10", "0", FieldConstants.NOT_A_KEY, Field.FIELDTYPE_TRANSFORM, false );
    rsGroupFld.add( field1 );
    Field field2 = new Field( "Item_Name", "Item_Name", "", DataTypeConstants.STRING,
        "72", "0", FieldConstants.NOT_A_KEY, Field.FIELDTYPE_TRANSFORM, false );
    rsGroupFld.add( field2 );
    Field field3 = new Field( "Item_Price", "Item_Price", "", DataTypeConstants.DECIMAL, "10",
        "2", FieldConstants.NOT_A_KEY, Field.FIELDTYPE_TRANSFORM, false );
    rsGroupFld.add( field3 );
// creating DSQ Transformation
OutputSet itemOSet = helper.sourceQualifier( itemsSrc );
RowSet itemRowSet = (RowSet) itemOSet.getRowSets().get( 0 );
OutputSet productOSet = helper.sourceQualifier( productSrc );
RowSet productRowSet = (RowSet) productOSet.getRowSets().get( 0 );
// 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" } );
Vector vInputSet = new Vector();
Sample Code Using the Design API

```java
vInputSet.add( new InputSet( itemRowSet, itemRSContext ) );
vInputSet.add( new InputSet( productRowSet, productRSContext ) );
// create a Union Transformation
RowSet unionRS = (RowSet) helper.union( vInputSet, rsGroupFld, "Union_transform"
)
.getResultSets().get( 0 );
// write to target
mapping.writeTarget( unionRS, outputTarget );

Expression

// create an expression Transformation
// the fields LastName and FirstName are concatenated 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 = GET_DATE_PART(BirthDate, 'YYYY')";
TransformField outField2 = new TransformField( expr2 );
Vector vFields = new Vector();
vFields.add( outField );
vFields.add( outField2 );
RowSet expRS = (RowSet) helper.expression( dsqRS, vFields, "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 );

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(
  "number(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);

Update Strategy

// create a 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 );

Unconnected Lookup

// create an expression Transformation
// the fields LastName and FirstName are concatenated 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", ",", DataTypeConstants.INTEGER, "10", "0", FieldConstants.PRIMARY_KEY, Field.FIELDTYPE_SOURCE, false );
// create input and output fields
Vector input = new Vector();
Vector output = new Vector();
createUncLkpFields( input, output );
// create an unconnected lookup
String condition = "ItemId = EmployeeID";
UnconnectedLookup uncLkp = helper.unconnectedLookup( "lkp", retField, input, output );
uncLkp.setPortType( "EmployeeID", OutputField.TYPE_LOOKUP );
// write to target
mapping.writeTarget( expRS, outputTarget );

Stored Procedure

// create a stored procedure transformation
Vector vTransformFields = new Vector();
Field field1 = new Field( "RetValue", "RetValue", "This is return value", DataTypeConstants.INTEGER, "10", "0", FieldConstants.NOT_A_KEY, OutputField.TYPE_RETURN_OUTPUT, false );
TransformField tField1 = new TransformField( field1, OutputField.TYPE_RETURN_OUTPUT );
vTransformFields.add( tField1 );
Field field2 = new Field( "nID1", "nID1", "This is the ID field", DataTypeConstants.INTEGER, "10", "0", FieldConstants.NOT_A_KEY, OutputField.TYPE_INPUT, false );
TransformField tField2 = new TransformField( field2, OutputField.TYPE_INPUT );
// vTransformFields.add( tField2 );
Field field3 = new Field( "outVar", "outVar", "This is the Output field", DataTypeConstants.STRING, "20", "0", FieldConstants.NOT_A_KEY, OutputField.TYPE_INPUT_OUTPUT, false );
TransformField tField3 = new TransformField( field3, OutputField.TYPE_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 );

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 );
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:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
</table>
| `[A-Za-z_][A-Za-z_0-9]*` (TO) x_$0 | Prefix all port names with `x_`. Examples: 
  - EMPLOYEE_ID => x_EMPLOYEE_ID 
  - EMPLOYEE_NAME => x_EMPLOYEE_NAME 
  - GENDER => x_GENDER 
  - 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: 
  - EMPLOYEE_ID => EMPLOYEE_ID_IN 
  - EMPLOYEE_NAME => EMPLOYEE_NAME_IN 
  - GENDER is not selected. 
  - ETHIC_GROUP is not selected. |
| `pattern = \[[0-9]\] (.TO) $0` | Select port names that end with a digit and keep the name as is. 
  Examples: 
  - EMPLOYEE_ID is not selected. 
  - EMPLOYEE_NAME1 => EMPLOYEE_NAME1 
  - GENDER35 => GENDER35 
  - ETHIC_GROUP is not selected. |
| `_IN$` | Select port names with the suffix `_IN` and remove the suffix from the port names. Examples: 
  - EMPLOYEE_ID is not selected. 
  - EMPLOYEE_NAME is not selected. 
  - GENDER_IN => GENDER 
  - ETHIC_GROUP_IN => ETHIC_GROUP |
| `^IN_` | Select port names that with the prefix `IN_` and remove the prefix from the port names. Examples: 
  - IN_EMPLOYEE_ID => EMPLOYEE_ID 
  - IN_EMPLOYEE_NAME => EMPLOYEE_NAME 
  - GENDER is not selected. 
  - IN_ETHIC_GROUP => ETHIC_GROUP |
Appendix B

Interface Limitations

This chapter includes the following topics:
♦ Overview, 103
♦ Design API, 103
♦ PowerExchange API, 104

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. You cannot use the Design API to generate metadata for mappings that require these data sources:
  – Mainframe relational data sources such as DB2 for z/OS
  – Nonrelational sources on MVS such as IMS
  – XML data sources
♦ Transformations. You cannot use the Design API to generate transformation metadata for the following transformations:
  – Unstructured Data
  – Data Quality
  – Java
  – XML Generator
  – XML Parser
  – XML Source Qualifier
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

A
ALLOWEDCONNECTION element
description 38
ALLOWEDDBTYPE element
description 37
ALLOWEDTEMPLATE element
description 37
API
Informatica Development Platform 7
ATTRIBUTE element
description 34
ATTRIBUTECATEGORY element
description 36

C
CLASS element
description 36
client plug-in
registering 16
command line programs
description 8
interface to PowerCenter 7
commit interval
Java DB adapter example 59
commit type
Java DB adapter example 59
compiler
selecting for development 16
CONNECTION element
description 40
Java DB adapter example 53
CONNECTIONREFERENCE element
description 37
CONNECTIONTOEXTENSION element
description 42
Java DB adapter example 54
Custom Function API
description 12
usage 12

D
datatype conversion
JDBC to PowerCenter 58
possible JDBC to PowerCenter conversion 58
DATATYPE element
description 29
DBSUBTYPE element
description 26
DBTYPE element
description 23
Java DB adapter example 48
DBTYPETOEXTENSION element
description 41
DBTYPETOWIDGETATTR element
description 31
Java DB adapter example 50
Design API
connection objects, creating 72
data flow 66
description 11, 63
exporting 72
folder object, creating 64
importing 72
linking ports 67
mappings, creating 66
multi-group transformations 66
objects, creating 64
port link context object 68
port propagation context object 67
propagating ports 67
regular expression, sample pattern 101
repository object, creating 64
repository, browsing 63
sessions, creating 70
single-group transformations 66
source, creating 65
target, creating 65
transformation inputs 66
transformation outputs 66
transformations, creating 69
usage 11
workflows, creating 70
Design API sample code
Aggregator transformation, creating 99
Expression transformation, creating 99
Filter transformation, creating 99
mapplets, using 94
objects, validating 95
parameter files, creating 96
PowerExchange sources, creating 97
relational sources, creating 96
reusable transformations, using 95
Router transformation, creating 98
Sequence Generator transformation, creating 101
shortcuts, using 95
Sorter transformation, creating 98
source and targets, using 93
source qualifier, using 94
SQL transformation, creating 101
Stored Procedure transformation, creating 100
Transaction Control transformation, creating 100
transformations, creating 98
unconnected Lookup transformation, creating 99
Union transformation, creating 98
Update Strategy transformation, creating 99
development environment
setting up 16
DLLs (dynamic linked libraries)
compiling C++ on Windows 17
path, defining 16
dynamic linked libraries
See DLLs

error handling
Java DB adapter example 60
EXTENSION element
description 33
Java DB adapter example 50

FIELDATTR element
description 30

HIDDENCONNECTIONATTRIBUTETOEXTENSION element
description 40
HIDDENEXTENSIONATTRIBUTETOCONNECTION element
description 43

Informatica Development Platform
Custom Function API 12
definition 7
Design API 11
installation 8
interface to PowerCenter 7
Operations API 10
PowerExchange API 9
Transformation API 10
interfaces
to PowerCenter 7

KEYTYPE element
description 29

LIBRARY element
description 32

MEDEFINITION element
description 44
MEDOMAIN element
description 43
Java DB adapter example 54
metadata
plug-in, defining for 15
plug-in, registering client 16
plug-in, registering server 15
MULTIVALUEATTRIBUTE element
description 32

Operations API
description 10
usage 11

partitions
Java DB adapter example 60
path
to DLLs 16
plug-in definition file
template example, pmJDBC.xml 47
PLUGIN element
description 22
Java DB adapter example 48
plug-in metadata
ALLOWEDCONNECTION element 38
ALLOWEDDBTYPE element 37
ALLOWEDTEMPLATE element 37
ATTRIBUTE element 34
ATTRIBUTECATEGORY element 36
CLASS element 36
CONNECTION element 40
CONNECTIONREFERENCE element 37
CONNECTIONTOEXTENSION element 42
DATATYPE element 29
DBSUBTYPE element 26
DBTYPE element 23
DBTYPOETOEXTENSION element 41
DBTYPOETOATTRIBUTE element 31
defining 15
EXTENSION element 33
FIELDATTR element 30
HIDDENCONNECTIONATTRIBUTETOEXTENSION element 40
HIDDENEXTENSIONATTRIBUTETOCONNECTION element 43
KEYTYPE element 29
LIBRARY element 32
MEDEFINITION element 44
MEDOMAIN element 43
MULTIVALUEATTRIBUTE element 32
PLUGIN element 22
registering 15, 16
structure 21
plugin.dtd
structure 21
plug-ins
building server and client 17
client, registering 16
debugging 18
repository ID attributes, obtaining 13
server, registering 15
unregistering 19
port propagation
regex 101
PowerExchange API
description 9
requirements 9
usage 9

R
reader buffer
Java DB adapter example 59
reader extension
Java DB adapter example 50
reader extension attributes
Java DB adapter example 51
reader session
Java DB adapter example 56
REG file
client plug-ins, registering 16
regex
sample patterns 101
repository ID attributes
plug-ins, obtaining for 13

S
server plug-in
example, pmJDBCplugin.jar. 47
registering 15
shared library
compiling on UNIX 18

T
Transformation API
description 10
usage 10

W
web services
description 8
interface to PowerCenter 7
Windows registry
client plug-ins, registering 16
writer buffer
Java DB adapter example 59
NOTICES

This Informatica product (the “Software”) includes certain drivers (the “DataDirect Drivers”) from DataDirect Technologies, an operating company of Progress Software Corporation (“DataDirect”) which are subject to the following terms and conditions:

1. THE DATADIRECT DRIVERS ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.

2. IN NO EVENT WILL DATADIRECT OR ITS THIRD PARTY SUPPLIERS BE LIABLE TO THE END-USER CUSTOMER FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR OTHER DAMAGES ARISING OUT OF THE USE OF THE ODBC DRIVERS, WHETHER OR NOT INFORMED OF THE POSSIBILITIES OF DAMAGES IN ADVANCE. THESE LIMITATIONS APPLY TO ALL CAUSES OF ACTION, INCLUDING, WITHOUT LIMITATION, BREACH OF CONTRACT, BREACH OF WARRANTY, NEGLIGENCE, STRICT LIABILITY, MISREPRESENTATION AND OTHER TORTS.