

Oracle® Fusion Middleware

Developing Integration Projects with Oracle Data Integrator

12c (12.2.1.1)

E69522-01

May 2016

Oracle Fusion Middleware Developing Integration Projects with Oracle Data Integrator, 12c (12.2.1.1)

E69522-01

Copyright © 2010, 2016, Oracle and/or its affiliates. All rights reserved.

Primary Author: Joshua Stanley, Rick Sapir, Aslam Khan

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface	xv
Audience	xv
Documentation Accessibility	xv
Related Documents	xv
Conventions	xvi
What's New In Oracle Data Integrator?	xvii
1 Overview of an Integration Project	
1.1 Oracle Data Integrator Project Quick Start List	1-1
2 Overview of Oracle Data Integrator Topology	
2.1 Introduction to the Oracle Data Integrator Topology	2-1
2.1.1 Physical Architecture	2-1
2.1.2 Contexts	2-2
2.1.3 Logical Architecture	2-2
2.1.4 Agents	2-2
2.1.5 Languages	2-5
2.1.6 Repositories	2-5
3 Creating and Using Data Models and Datastores	
3.1 Introduction to Models	3-1
3.1.1 Datastores	3-2
3.1.2 Data Integrity	3-2
3.1.3 Reverse-engineering	3-2
3.1.4 Changed Data Capture	3-3
3.2 Creating and Reverse-Engineering a Model	3-3
3.2.1 Creating a Model	3-3
3.2.2 Creating a Model and Topology Objects	3-4
3.2.3 Reverse-engineering a Model	3-5
3.3 Creating and Reverse-Engineering a Datastore	3-7
3.3.1 Creating a Datastore	3-7
3.3.2 Reverse-Engineering File Datastores	3-8
3.3.2.1 Reverse-Engineering Fixed Files	3-8

3.3.2.2	Reverse-Engineering Delimited Files	3-8
3.3.2.3	Reverse-Engineering COBOL Files	3-8
3.3.3	Adding and Deleting Datastore Attributes	3-9
3.3.4	Adding and Deleting Constraints and Filters	3-9
3.3.4.1	Keys	3-9
3.3.4.2	References	3-10
3.3.4.3	Conditions	3-10
3.3.4.4	Mandatory Attributes	3-11
3.3.4.5	Filter	3-11
3.4	Editing and Viewing a Datastore's Data.....	3-11
3.5	Using Partitioning.....	3-12
3.5.1	Manually Defining Partitions and Sub-Partitions of Model Datastores.....	3-12
3.6	Checking Data Quality in a Model.....	3-13
3.6.1	Introduction to Data Integrity.....	3-13
3.6.2	Checking a Constraint.....	3-13
3.6.3	Perform a Static Check on a Model, Sub-Model or Datastore.....	3-13
3.6.4	Reviewing Erroneous Records.....	3-14

4 Using Journalizing

4.1	Introduction to Changed Data Capture.....	4-1
4.1.1	The Journalizing Components	4-2
4.1.2	Simple vs. Consistent Set Journalizing	4-2
4.2	Setting up Journalizing	4-3
4.2.1	Setting up and Starting Journalizing	4-3
4.2.2	Journalizing Infrastructure Details	4-7
4.2.3	Journalizing Status.....	4-7
4.3	Using Changed Data	4-8
4.3.1	Viewing Changed Data.....	4-8
4.3.2	Using Changed Data: Simple Journalizing	4-8
4.3.3	Using Changed Data: Consistent Set Journalizing	4-9
4.3.4	Journalizing Tools.....	4-11
4.3.5	Package Templates for Using Journalizing.....	4-11

5 Creating Data Models with Common Format Designer

5.1	Introduction to Common Format Designer	5-1
5.1.1	What is a Diagram?	5-1
5.1.2	Why assemble datastores and attributes from other models?	5-2
5.1.3	Graphical Synonyms	5-2
5.2	Using the Diagram.....	5-2
5.2.1	Creating a New Diagram.....	5-2
5.2.2	Create Datastores and Attributes	5-2
5.2.3	Creating Graphical Synonyms.....	5-3
5.2.4	Creating and Editing Constraints and Filters.....	5-3
5.2.5	Printing a Diagram	5-4
5.3	Generating DDL scripts	5-5
5.4	Generating Mapping IN/OUT.....	5-6

6 Creating an Integration Project

6.1	Introduction to Integration Projects	6-1
6.1.1	Oracle Data Integrator Project Components.....	6-1
6.1.1.1	Oracle Data Integrator Project Components	6-1
6.1.1.2	Global Components.....	6-3
6.1.2	Project Life Cycle	6-3
6.2	Creating a New Project	6-3
6.3	Managing Knowledge Modules	6-3
6.3.1	Project and Global Knowledge Modules.....	6-4
6.3.2	Knowledge Module Naming Conventions.....	6-4
6.3.3	Choosing the Right Knowledge Modules	6-7
6.3.4	Importing and Replacing Knowledge Modules.....	6-7
6.3.5	Encrypting and Decrypting a Knowledge Module	6-9
6.4	Organizing the Project with Folders	6-10

7 Creating and Using Packages

7.1	Introduction to Packages	7-1
7.1.1	Introduction to Steps	7-1
7.1.2	Introduction to Creating Packages.....	7-3
7.1.3	Introduction to the Package editor.....	7-3
7.2	Creating a new Package.....	7-4
7.3	Working with Steps	7-4
7.3.1	Adding a Step.....	7-4
7.3.1.1	Adding a Mapping step.....	7-4
7.3.1.2	Adding a Procedure step.....	7-5
7.3.1.3	Variable Steps.....	7-5
7.3.1.4	Adding Oracle Data Integrator Tool Steps	7-6
7.3.1.5	Adding a Model, Sub-Model or Datastore	7-7
7.3.2	Deleting a Step	7-7
7.3.3	Duplicating a Step.....	7-8
7.3.4	Running a Step	7-8
7.3.5	Editing a Step's Linked Object	7-8
7.3.6	Arranging the Steps Layout	7-8
7.4	Defining the Sequence of Steps.....	7-9
7.5	Running a Package	7-11

8 Creating and Using Mappings

8.1	Introduction to Mappings.....	8-1
8.1.1	Parts of a Mapping	8-1
8.1.2	Navigating the Mapping Editor	8-3
8.2	Creating a Mapping.....	8-5
8.2.1	Creating a New Mapping.....	8-5
8.2.2	Adding and Removing Components.....	8-6
8.2.3	Connecting and Configuring Components.....	8-7
8.2.3.1	Attribute Matching.....	8-7
8.2.3.2	Connector Points and Connector Ports	8-7

8.2.3.3	Defining New Attributes.....	8-8
8.2.3.4	Defining Expressions and Conditions.....	8-9
8.2.4	Defining a Physical Configuration.....	8-10
8.2.5	Running Mappings.....	8-10
8.3	Using Mapping Components.....	8-11
8.3.1	The Expression Editor.....	8-12
8.3.2	Source and Target Datastores.....	8-13
8.3.3	Creating Multiple Targets.....	8-14
8.3.3.1	Specifying Target Order.....	8-14
8.3.4	Adding a Reusable Mapping.....	8-15
8.3.5	Creating Aggregates.....	8-15
8.3.6	Creating Distincts.....	8-16
8.3.7	Creating Expressions.....	8-17
8.3.8	Creating Filters.....	8-17
8.3.9	Creating Joins and Lookups.....	8-18
8.3.10	Creating Pivots.....	8-21
8.3.10.1	Example: Pivoting Sales Data.....	8-22
8.3.10.2	The Row Locator.....	8-22
8.3.10.3	Using the Pivot Component.....	8-22
8.3.11	Creating Sets.....	8-23
8.3.12	Creating Sorts.....	8-24
8.3.13	Creating Splits.....	8-25
8.3.14	Creating Subquery Filters.....	8-25
8.3.15	Creating Table Functions.....	8-26
8.3.16	Creating Unpivots.....	8-28
8.3.16.1	Example: Unpivoting Sales Data.....	8-28
8.3.16.2	The Row Locator.....	8-28
8.3.16.3	Using the Unpivot Component.....	8-29
8.3.17	Creating Flatten Components.....	8-30
8.3.17.1	Using a Flatten Component in a Mapping.....	8-30
8.3.17.2	Considerations for using Flatten component with JSON Source.....	8-31
8.3.18	Creating Jagged Components.....	8-31
8.4	Creating a Mapping Using a Dataset.....	8-32
8.4.1	Differences Between Flow and Dataset Modeling.....	8-32
8.4.2	Creating a Dataset in a Mapping.....	8-33
8.4.3	Converting a Dataset to Flow-Based Mapping.....	8-33
8.5	Physical Design.....	8-33
8.5.1	About the Physical Mapping Diagram.....	8-34
8.5.2	Selecting LKMs, IKMs and CKMs.....	8-35
8.5.3	Configuring Execution Locations.....	8-36
8.5.3.1	Moving Physical Nodes.....	8-37
8.5.3.2	Moving Expressions.....	8-37
8.5.3.3	Defining New Execution Units.....	8-37
8.5.4	Adding Commands to be Executed Before and After a Mapping.....	8-37
8.5.5	Configuring In-Session Parallelism.....	8-38
8.5.6	Configuring Parallel Target Table Load.....	8-38
8.5.7	Configuring Temporary Indexes.....	8-39

8.5.8	Configuring Journalizing.....	8-39
8.5.9	Configuring Extraction Options.....	8-39
8.5.10	Creating and Managing Physical Mapping Designs.....	8-39
8.6	Reusable Mappings.....	8-40
8.6.1	Creating a Reusable Mapping.....	8-40
8.7	Editing Mappings Using the Property Inspector and the Structure Panel.....	8-41
8.7.1	Adding and Removing Components.....	8-41
8.7.1.1	Adding Components.....	8-41
8.7.1.2	Removing Components.....	8-42
8.7.2	Editing a Component.....	8-42
8.7.3	Customizing Tables.....	8-42
8.7.4	Using Keyboard Navigation for Common Tasks.....	8-42
8.8	Flow Control and Static Control.....	8-43
8.8.1	Setting up Flow Control.....	8-44
8.8.2	Setting up Static Control.....	8-44
8.8.3	Defining the Update Key.....	8-44
8.9	Designing E-LT and ETL-Style Mappings.....	8-45

9 Creating and Using Dimensions and Cubes

9.1	Overview of Dimensional Objects.....	9-1
9.1.1	Overview of Dimensions.....	9-1
9.1.2	Overview of Cubes.....	9-5
9.1.2.1	Understanding Measure (Fact).....	9-5
9.1.2.2	Cube Implementation.....	9-5
9.2	Creating Dimensional Objects through ODI.....	9-5
9.2.1	Dimension and Cube Accordion.....	9-5
9.2.2	Using Dimensions in ODI.....	9-6
9.2.3	Using Cubes in ODI.....	9-7
9.2.3.1	Generic Properties.....	9-7
9.2.3.2	Cube Measures.....	9-8
9.2.4	Creating New Dimensional Models.....	9-8
9.2.5	Creating and Editing Dimensional Objects using the Editor.....	9-9
9.2.5.1	Using Dimension Editor.....	9-9
9.2.5.1.1	Definition Tab.....	9-9
9.2.5.1.2	Levels Tab.....	9-10
9.2.5.1.3	Hierarchies Tab.....	9-13
9.2.5.2	Using the Cube Editor.....	9-14
9.2.5.2.1	Definition Tab.....	9-14
9.2.5.2.2	Details Tab.....	9-14
9.3	Using Dimensional Components in Mappings.....	9-15
9.3.1	Using Dimension Component in Mapping.....	9-15
9.3.1.1	Dimension Component Properties Editor.....	9-16
9.3.1.1.1	Attributes.....	9-16
9.3.1.1.2	General Properties.....	9-16
9.3.1.1.3	Connector Points.....	9-17
9.3.1.1.4	History Properties.....	9-17
9.3.1.1.5	Target Properties.....	9-17

9.3.2	Using Cube Component in Mappings	9-18
9.3.2.1	Cube Component Properties Editor	9-18
9.3.2.1.1	Attributes	9-19
9.3.2.1.2	General Properties	9-20
9.3.2.1.3	Target Properties	9-20
9.3.2.1.4	Connector Points	9-21
9.4	Expanding Dimensional Components	9-21
9.4.1	Expanding Dimension Component	9-21
9.4.2	Expanding Cube Component	9-22

10 Using Compatibility Mode

10.1	About Compatibility Mode	10-1
10.2	Creating Compatible Mappings	10-2
10.2.1	Creating Mappings using Upgrade Assistant	10-2
10.2.2	Creating Mappings with the 11g SDK in ODI 12c	10-2
10.3	About Internal Identifiers (IDs)	10-2
10.4	Renumbering Repositories	10-3

11 Creating and Using Procedures, Variables, Sequences, and User Functions

11.1	Working with Procedures	11-1
11.1.1	Introduction to Procedures	11-1
11.1.2	Creating Procedures	11-2
11.1.2.1	Create a New Procedure	11-2
11.1.2.2	Define the Procedure's Options	11-3
11.1.2.3	Create and Manage the Procedure's Tasks	11-4
11.1.3	Using Procedures	11-10
11.1.3.1	Executing the Procedure	11-10
11.1.3.2	Using a Procedure in a Package	11-10
11.1.3.3	Generating a Scenario for a Procedure	11-11
11.1.4	Encrypting and Decrypting Procedures	11-11
11.2	Working with Variables	11-11
11.2.1	Introduction to Variables	11-11
11.2.2	Creating Variables	11-12
11.2.3	Using Variables	11-14
11.2.3.1	Using Variables in Packages	11-15
11.2.3.2	Using Variables in Mappings	11-16
11.2.3.3	Using Variables in Object Properties	11-17
11.2.3.4	Using Variables in Procedures	11-17
11.2.3.5	Using Variables within Variables	11-18
11.2.3.6	Using Variables in the Resource Name of a Datastore	11-19
11.2.3.7	Using Variables in a Server URL	11-19
11.2.3.8	Using Variables in On Connect/Disconnect Commands	11-21
11.2.3.9	Passing a Variable to a Scenario	11-21
11.2.3.10	Generating a Scenario for a Variable	11-21
11.2.3.11	Tracking Variables and Sequences	11-21
11.3	Working with Sequences	11-22
11.3.1	Introduction to Sequences	11-22

11.3.2	Creating Sequences.....	11-23
11.3.2.1	Creating Standard Sequences	11-23
11.3.2.2	Creating Specific Sequences.....	11-23
11.3.2.3	Creating Native Sequences	11-24
11.3.3	Using Sequences and Identity Columns	11-24
11.3.3.1	Tips for Using Standard and Specific Sequences.....	11-26
11.3.3.2	Identity Columns.....	11-26
11.3.4	Sequence Enhancements.....	11-26
11.4	Working with User Functions.....	11-27
11.4.1	Introduction to User Functions.....	11-27
11.4.2	Creating User Functions	11-28
11.4.3	Using User Functions	11-29

12 Using Scenarios

12.1	Introduction to Scenarios.....	12-1
12.2	Generating a Scenario.....	12-2
12.3	Regenerating a Scenario.....	12-3
12.4	Generating a Group of Scenarios.....	12-3
12.5	Controlling Concurrent Execution of Scenarios and Load Plans	12-4
12.6	Exporting Scenarios	12-5
12.7	Importing Scenarios in Production	12-6
12.7.1	Import Scenarios	12-7
12.7.2	Replace a Scenario	12-7
12.7.3	Working with a Scenario from a Different Repository	12-7
12.8	Encrypting and Decrypting a Scenario.....	12-8

13 Using Load Plans

13.1	Introduction to Load Plans.....	13-1
13.1.1	Load Plan Execution Lifecycle	13-2
13.1.2	Differences between Packages, Scenarios, and Load Plans.....	13-2
13.1.3	Load Plan Structure.....	13-2
13.1.4	Introduction to the Load Plan Editor.....	13-4
13.2	Creating a Load Plan	13-6
13.2.1	Creating a New Load Plan	13-6
13.2.2	Defining the Load Plan Step Sequence.....	13-8
13.2.2.1	Adding Load Plan Steps.....	13-8
13.2.2.2	Editing Load Plan Steps.....	13-12
13.2.2.3	Deleting a Step	13-14
13.2.2.4	Duplicating a Step	13-14
13.2.3	Working with Variables in Load Plans.....	13-15
13.2.3.1	Declaring Load Plan Variables	13-15
13.2.3.2	Setting Variable Values in a Step.....	13-15
13.2.4	Handling Load Plan Exceptions and Restartability.....	13-16
13.2.4.1	Defining Exceptions Flows.....	13-17
13.2.4.2	Using Exception Handling	13-17
13.2.4.3	Defining the Restart Behavior.....	13-18

13.3	Running Load Plans	13-19
13.4	Using Load Plans in Production.....	13-20
13.4.1	Scheduling and Running Load Plans in Production	13-20
13.4.2	Exporting, Importing and Versioning Load Plans.....	13-20
13.4.2.1	Exporting Load Plans.....	13-20
13.4.2.2	Importing Load Plans	13-20
13.4.2.3	Versioning Load Plans.....	13-21

14 Using Web Services

14.1	Introduction to Web Services in Oracle Data Integrator.....	14-1
14.2	Oracle Data Integrator Run-Time Services and Data Services.....	14-2
14.3	Invoking Third-Party Web Services	14-2
14.3.1	Introduction to Web Service Invocation.....	14-3
14.3.2	Using HTTP Analyzer.....	14-3
14.3.2.1	Using HTTP Analyzer: Main Steps.....	14-4
14.3.2.2	What Happens When You Run the HTTP Analyzer.....	14-4
14.3.2.3	How to Specify HTTP Analyzer Settings.....	14-5
14.3.2.4	How to Use the Log Window	14-5
14.3.2.5	How to Use the Test Window.....	14-6
14.3.2.6	How to Use the Instances Window.....	14-8
14.3.2.7	How to Use Multiple Instances	14-9
14.3.2.8	Using Credentials With HTTP Analyzer	14-9
14.3.2.9	Using SSL With HTTP Analyzer	14-9
14.3.2.10	How to Debug Web Pages Using the HTTP Analyzer	14-10
14.3.2.11	How to Use Rules to Determine Behavior	14-10
14.3.2.11.1	Using the Pass Through Rule	14-11
14.3.2.11.2	Using the Forward Rule	14-11
14.3.2.11.3	Using the URL Substitution Rule.....	14-11
14.3.2.11.4	Using the Tape Rule.....	14-11
14.3.2.12	How to Set Rules.....	14-12
14.3.2.13	Reference: Troubleshooting the HTTP Analyzer.....	14-12
14.3.2.13.1	Running the HTTP Analyzer While Another Application is Running	14-12
14.3.2.13.2	Changing Proxy Settings.....	14-13
14.3.3	Using the OdiInvokeWebService Tool	14-13

15 Using Shortcuts

15.1	Introduction to Shortcuts.....	15-1
15.1.1	Shortcutting Concepts.....	15-1
15.1.2	Shortcut Objects	15-2
15.2	Introduction to the Shortcut Editor	15-2
15.3	Creating a Shortcut	15-3
15.4	Working with Shortcuts in your Projects	15-4
15.4.1	Duplicating a Selection with Shortcuts.....	15-4
15.4.2	Jump to the Reference Shortcut	15-5
15.4.3	Jump to the Base Object	15-5
15.4.4	Executing Shortcuts.....	15-5
15.4.5	Materializing Shortcuts.....	15-5

15.4.6	Exporting and Importing Shortcuts	15-5
15.4.7	Using Release Tags	15-6
15.4.8	Advanced Actions	15-7
16	Using Groovy Scripting	
16.1	Introduction to Groovy	16-1
16.2	Introduction to the Groovy Editor	16-1
16.3	Using the Groovy Editor	16-2
16.3.1	Create a Groovy Script	16-3
16.3.2	Open and Edit an Existing Groovy Script	16-3
16.3.3	Save a Groovy Script	16-3
16.3.4	Execute a Groovy Script	16-3
16.3.5	Stop the Execution of a Groovy Script	16-4
16.3.6	Perform Advanced Actions	16-4
16.4	Automating Development Tasks - Examples	16-6
17	Exchanging Global ODI Objects	
17.1	Using the Check for Updates Wizard	17-1
18	Organizing and Documenting Integration Projects	
18.1	Organizing Projects with Folders	18-1
18.1.1	Creating a New Folder	18-1
18.1.2	Arranging Project Folders	18-2
18.2	Organizing Models with Folders	18-2
18.2.1	Creating a New Model Folder	18-2
18.2.2	Arranging Model Folders	18-2
18.2.3	Creating and Organizing Sub-Models	18-2
18.3	Using Cross-References	18-4
18.3.1	Browsing Cross-References	18-4
18.3.2	Resolving Missing References	18-5
18.4	Using Markers and Memos	18-6
18.4.1	Markers	18-6
18.4.2	Memos	18-7
18.5	Handling Concurrent Changes	18-7
18.5.1	Concurrent Editing Check	18-8
18.5.2	Object Locking	18-8
18.6	Creating PDF Reports	18-9
18.6.1	Generating a Topology Report	18-9
18.6.2	Generating a Report for the Version Comparison Results	18-10
18.6.3	Generating a Report for an Oracle Data Integrator Object	18-10
18.6.4	Generating a Diagram Report	18-10
19	Using Version Control (Legacy Mode)	
19.1	Working with Object Flags	19-1
19.2	Working with Versions	19-2

19.3	Working with the Version Comparison Tool	19-4
19.3.1	Viewing the Differences between two Versions	19-5
19.3.2	Using Comparison Filters	19-6
19.3.3	Generating and Printing a Report of your Comparison Results	19-6
19.4	Working with Labels	19-7
19.4.1	Working with Elements in a Label	19-8
19.4.2	Synchronizing Labels	19-8
19.4.3	Restoring and Checking in a Label	19-8
19.4.4	Importing and Exporting Labels	19-9

20 Integrating ODI with Version Control Systems

20.1	Introduction to ODI-VCS integration	20-2
20.2	Selecting the VCS to use with ODI	20-2
20.3	Creating an SVN Connection	20-3
20.3.1	HTTP Basic Authentication Options	20-3
20.3.2	Subversion Basic Authentication Options	20-4
20.3.3	SSH Authentication Options	20-4
20.3.4	SSL Authentication Options	20-4
20.3.5	File Based Authentication Options	20-4
20.4	Editing an SVN Connection	20-5
20.5	Configuring Subversion Settings	20-5
20.5.1	Subversion Settings	20-5
20.6	Configuring Subversion Repository with ODI	20-6
20.6.1	Options to Configure Subversion Repository with ODI	20-6
20.7	Creating a Default Subversion Project Structure	20-6
20.8	Populating a New ODI Repository from a Subversion Branch/Trunk	20-7
20.9	Populating a Restored ODI Repository from a Subversion Branch/Trunk	20-7
20.10	Understanding Generic Profiles in ODI	20-7
20.11	Creating a Full Tag in the Subversion Repository	20-8
20.12	Creating a Partial Tag in the Subversion Repository	20-8
20.12.1	Create Partial Tag Options	20-9
20.13	Creating a Branch from a Tag	20-9
20.13.1	Create Branch from Tag Options	20-9
20.14	Unlocking the ODI Repository	20-10
20.15	Adding Non-versioned ODI Objects to the Subversion Repository	20-10
20.16	Adding a Single Non-versioned ODI Object to the Subversion Repository	20-10
20.17	Creating versions of a version controlled ODI Object	20-11
20.18	Restoring a Version Controlled ODI Object from its Previous Version	20-11
20.18.1	Restore Object from Subversion Options	20-11
20.19	Restoring a Version Controlled ODI Object Deleted in ODI Repository	20-12
20.20	Viewing the Version History of a Version Controlled ODI Object	20-12
20.20.1	Version Search Criteria	20-13
20.21	Comparing Versions of an ODI Object from the Version History Dialog	20-13
20.21.1	Icons on the Version Compare Results dialog	20-14
20.22	Viewing Version Tree of a Version Controlled ODI Object	20-15
20.23	Comparing Versions of an ODI Object from the Version Tree Editor	20-15
20.24	Performing a Merge	20-16

20.25	Performing a Branch Merge	20-16
20.25.1	Viewing Merge Summary	20-17

21 Release Management

21.1	Managing ODI Releases	21-1
21.2	Types of Deployment Archives	21-1
21.3	Creating a Deployment Archive from a VCS Label.....	21-2
21.4	Creating an Initial Deployment Archive from the ODI Repository	21-2
21.5	Creating a Patch Deployment Archive from the ODI Repository	21-3
21.6	Viewing Available Deployment Archives.....	21-4
21.7	Initializing an ODI Repository Using an Initial Deployment Archive	21-4
21.8	Updating an ODI Repository Using a Patch Deployment Archive.....	21-5
21.9	Viewing Deployment Archives Applied in an ODI Repository	21-6
21.10	Rolling Back a Patch Deployment Archive	21-6

22 Life Cycle Management Guidelines

22.1	Guidelines for Choosing the Authentication Type	22-1
22.2	General Branching Guidelines	22-2
22.3	General Tagging Guidelines.....	22-2
22.4	Branching Guidelines for Single Development Team	22-3
22.5	Branching Guidelines for Parallel Development Teams	22-4
22.6	Guidelines for Release Branches for Parallel Development Teams.....	22-4
22.7	Guidelines for Versioning During Development.....	22-5
22.8	Guidelines for Deployment in Testing and Production Environments.....	22-6
22.9	Guidelines for Initial Deployment and Patching	22-7

23 Exporting and Importing

23.1	Import and Export Concepts	23-1
23.1.1	Global Identifiers (GUIDs)	23-1
23.1.2	Export Keys.....	23-2
23.1.3	Relationships between Objects	23-2
23.1.4	Import Modes.....	23-3
23.1.5	Tips for Import/Export.....	23-7
23.2	Exporting and Importing Objects	23-8
23.2.1	Exporting an Object with its Child Components	23-9
23.2.2	Exporting an Object without its Child Components	23-9
23.2.3	Partial Export/Import.....	23-9
23.2.4	Exporting one ODI Object	23-10
23.2.5	Export Multiple ODI Objects	23-11
23.2.6	Importing Objects	23-11
23.2.7	Smart Export and Import.....	23-14
23.2.7.1	Performing a Smart Export	23-14
23.2.7.2	Performing a Smart Import	23-17
23.3	Repository-Level Export/Import	23-19
23.3.1	Exporting and Importing the Master Repository.....	23-19
23.3.2	Export/Import Topology and Security Settings	23-22

23.3.3	Exporting and Importing a Work Repository	23-23
23.4	Exporting the Technical Environment	23-24
23.5	Exporting and Importing the Log.....	23-25

Preface

This manual describes how to develop data integration projects using Oracle Data Integrator.

This preface contains the following topics:

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)

Audience

This document is intended for developers and administrators who want to use Oracle Data Integrator (ODI) as a development tool for their integration processes. This guide explains how to work with the ODI graphical user interface, primarily ODI Studio and ODI Console. It guides you through common tasks and examples of development, as well as conceptual and background information on the features of ODI.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Related Documents

For more information, see the following documents in the *Oracle Data Integrator Library*.

- Release Notes for Oracle Data Integrator
- Understanding Oracle Data Integrator
- Administering Oracle Data Integrator

- Installing and Configuring Oracle Data Integrator
- Upgrading Oracle Data Integrator
- Application Adapters Guide for Oracle Data Integrator
- Developing Knowledge Modules with Oracle Data Integrator
- Connectivity and Knowledge Modules Guide for Oracle Data Integrator
- Migrating From Oracle Warehouse Builder to Oracle Data Integrator
- Oracle Data Integrator Tool Reference
- Data Services Java API Reference for Oracle Data Integrator
- Open Tools Java API Reference for Oracle Data Integrator
- Getting Started with SAP ABAP BW Adapter for Oracle Data Integrator
- Java API Reference for Oracle Data Integrator
- Getting Started with SAP ABAP ERP Adapter for Oracle Data Integrator
- *Oracle Data Integrator 12c Online Help*, which is available in ODI Studio through the JDeveloper Help Center when you press **F1** or from the main menu by selecting **Help**, and then **Search** or **Table of Contents**.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New In Oracle Data Integrator?

This chapter describes changes in the Oracle Data Integrator documentation organization introduced with Oracle Data Integrator 12c (12.2.1.1).

See Also: For details about new and changed features in Oracle Data Integrator, see "What's New in Oracle Data Integrator?" in *Administering Oracle Data Integrator*.

To improve the organization of information about Oracle Data Integrator, three new books have been added to the ODI documentation library:

- *Understanding Oracle Data Integrator*: This book presents introductory and conceptual information about ODI, including ODI terminology, architecture, typical integration project designs, and ODI environments.
- *Administering Oracle Data Integrator*: This manual describes how to perform configuration and user management tasks in ODI. This includes configuring ODI components, performing basic administrative tasks, running and monitoring integration processes, and managing security in ODI.
- *Oracle Data Integrator Tool Reference*: This guide describes how to use and develop Open Tools using ODI to design integration scenarios.

Several chapters and many topics previously found in the *Developer's Guide for Oracle Data Integrator 12c (12.1.2)* have been moved into these new books. Customers familiar with previous versions of the ODI developer documentation may find useful the following table, which provides the new locations of topics that have been moved to other books in the ODI library.

Topic	New Location
What's New in Oracle Data Integrator?	<i>Administering Oracle Data Integrator</i>
Introduction to Oracle Data Integrator	<i>Understanding Oracle Data Integrator</i>
Administering Repositories	<i>Administering Oracle Data Integrator</i>
Setting Up the Topology	<i>Administering Oracle Data Integrator</i>
Managing Agents	<i>Administering Oracle Data Integrator</i>
Creating and Using Data Services	<i>Administering Oracle Data Integrator</i>
Running and Monitoring Integration Processes	<i>Administering Oracle Data Integrator</i>
Managing Security Settings	<i>Administering Oracle Data Integrator</i>
Oracle Data Integrator Tools Reference	<i>Oracle Data Integrator Tool Reference</i>

Part I

Introduction to Developing with Oracle Data Integrator

This part provides an introduction to Oracle Data Integrator and the basic steps of creating an integration project with Oracle Data Integrator.

This part contains the following chapters:

- [Chapter 1, "Overview of an Integration Project"](#)
- [Chapter 2, "Overview of Oracle Data Integrator Topology"](#)

Overview of an Integration Project

This chapter introduces the basic steps to creating an integration project with Oracle Data Integrator (ODI). It will help you get started with ODI by outlining the basic functionalities and the minimum required steps.

This section is not intended to be used for advanced configuration, usage or troubleshooting.

1.1 Oracle Data Integrator Project Quick Start List

To perform the minimum required steps of a simple Oracle Data Integrator integration project, follow the ODI Project Quick Start list and go directly to the specified sections of this guide.

Prerequisites

Before performing the Quick Start procedure ensure that you have installed Oracle Data Integrator, including setting up and configuring ODI agents, according to the instructions in *Installing and Configuring Oracle Data Integrator*. You (or an administrator) should perform post-installation configuration and user management tasks as described in *Administering Oracle Data Integrator*

You should also be familiar with the material presented in *Understanding Oracle Data Integrator*.

ODI Project Quick Start List

The ODI Project Quick Start list describes the essential steps to creating and running a simple integration project. Once you are familiar with these essential steps, you can review additional material in this guide to help you with more advanced topics, complex integrations, and optional steps.

1. In Oracle Data Integrator, you perform developments on top of a logical topology. Refer to [Chapter 2, "Overview of Oracle Data Integrator Topology"](#) if you are not familiar with the topology. Create logical schemas and associate them with physical schemas in the Global context. See "Creating a Logical Schema" in *Administering Oracle Data Integrator* for more information.
2. Mappings use data models containing the source and target datastores, corresponding to data structures contained in a physical schema: tables, files, JMS messages, or elements from an XML file. Data models are usually reverse-engineered from your data server's metadata into an Oracle Data Integrator repository.

Create one or more models and datastores according to [Chapter 3, "Creating and Using Data Models and Datastores."](#)

3. The developed integration components are stored in a project. Creating a new project is covered in [Chapter 6, "Creating an Integration Project."](#)
4. Consider how you will organize your integration project. [Chapter 18, "Organizing and Documenting Integration Projects"](#), describes several built-in tools to assist with organization: projects and models can be organized into hierarchical folders, you can cross-reference and annotate objects with metadata using markers and memos, and you can generate PDF-formatted reports for non-ODI users to review.
5. Version control can be a powerful tool for working on an integration project when there are multiple developers involved, or if you want to preserve versions of files so that you can revert to previous states of the project. Review [Chapter 19, "Using Version Control \(Legacy Mode\)"](#), to see if ODI's build-in version control suits your development requirements.
6. Mappings use Knowledge Modules to generate their code. For more information refer to "What is E-LT?" in *Understanding Oracle Data Integrator*. Before creating mappings you need to import the Knowledge Modules corresponding to the technology of your data. Importing Knowledge Modules is described in ["Importing Objects"](#) on page 23-11. Which Knowledge Modules you need to import is discussed in *Connectivity and Knowledge Modules Guide for Oracle Data Integrator*.
7. To load your target datastores with data from source datastores, you need to create an mapping. A mapping consists of a set of rules that define the loading from one or more source datastores to one or more target datastores. Creating a new mapping for your integration project is described in ["Creating a Mapping"](#) on page 8-5.
8. Once you have finished creating a mapping, you can run it, as described in ["Running Mappings"](#) on page 8-10. Select **Local (No Agent)** to execute the mapping directly by Oracle Data Integrator.
9. An integration workflow may require the loading of several target datastores in a specific sequence. If you want to sequence your mappings, create a package. This is an optional step described in ["Creating a new Package"](#) on page 7-4.
10. If your workflow is complex, involves many source or target datastores, or if you need to manage execution of multiple mappings, packages, procedures, and variables in a specific logical sequence, consider using load plans. Load plans allow you to organize and run scenarios in a hierarchy of sequential and parallel conditional steps. Load plans are described in [Chapter 13, "Using Load Plans."](#)
11. You can view and monitor the execution results in the Operator navigator. Follow a mapping's execution using the Operator navigator is described in ["Monitoring Integration Processes"](#) in *Administering Oracle Data Integrator*.
12. While developing your integration, you can use the debugger functionality to identify and eliminate bugs in your project. The debugging tools are described in ["Debugging Integration Processes"](#) in *Administering Oracle Data Integrator*.

Overview of Oracle Data Integrator Topology

This chapter provides an overview of Oracle Data Integrator topology concepts and components relevant to ODI developers.

This chapter includes the following section:

- [Introduction to the Oracle Data Integrator Topology](#)

See Also: "Setting Up a Topology" in *Administering Oracle Data Integrator*

2.1 Introduction to the Oracle Data Integrator Topology

The Oracle Data Integrator Topology is the physical and logical representation of the Oracle Data Integrator architecture and components.

Note: The *Installation Guide for Oracle Data Integrator* uses the term "topology" in some sections to refer to the organization of servers, folders, and files on your physical servers. This chapter refers to the "topology" configured using the Topology Navigator in ODI Studio.

This section contains these topics:

- [Physical Architecture](#)
- [Contexts](#)
- [Logical Architecture](#)
- [Agents](#)
- [Languages](#)
- [Repositories](#)

2.1.1 Physical Architecture

The physical architecture defines the different elements of the information system, as well as their characteristics taken into account by Oracle Data Integrator. Each type of database (Oracle, DB2, etc.), Big Data source (Hive, HBase), file format (XML, Flat File), or application software is represented in Oracle Data Integrator by a technology.

A *technology* handles formatted data. Therefore, each technology is associated with one or more data types that allow Oracle Data Integrator to generate data handling scripts.

The physical components that store and expose structured data are defined as *data servers*. A data server is always linked to a single technology. A data server stores information according to a specific technical logic which is declared into *physical schemas* attached to this data server. Every database server, JMS message file, group of flat files, and so forth, that is used in Oracle Data Integrator, must be declared as a data server. Every schema, database, JMS Topic, etc., used in Oracle Data Integrator, must be declared as a physical schema.

Finally, the physical architecture includes the definition of the *Physical Agents*. These are the Java software components that run Oracle Data Integrator jobs.

2.1.2 Contexts

Contexts bring together components of the physical architecture (the real Architecture) of the information system with components of the Oracle Data Integrator logical architecture (the Architecture on which the user works).

For example, contexts may correspond to different execution environments (*Development*, *Test* and *Production*) or different execution locations (*Boston Site*, *New-York Site*, and so forth.) where similar physical resource exist.

Note that during installation the default *GLOBAL* context is created.

2.1.3 Logical Architecture

The logical architecture allows you to identify as a single Logical Schema a group of similar physical schemas (that contain datastores that are structurally identical) that are located in different physical locations. Logical Schemas, like their physical counterparts, are attached to a technology.

Contexts allow logical schemas to resolve to physical schemas. In a given context, a logical schema resolves to a single physical schema.

For example, the Oracle logical schema *Accounting* may correspond to two Oracle physical schemas:

- *Accounting Sample* used in the *Development* context
- *Accounting Corporate* used in the *Production* context

These two physical schemas are structurally identical (they contain accounting data), but are located in different physical locations. These locations are two different Oracle schemas (Physical Schemas), possibly located on two different Oracle instances (Data Servers).

All the components developed in Oracle Data Integrator are designed on top of the logical architecture. For example, a data model is always attached to logical schema, and data flows are defined with this model. By specifying a context at run-time (either *Development* or *Production*), the model's logical schema (*Accounting*) resolves to a single physical schema (either *Accounting Sample* or *Accounting Corporate*), and the data contained in this schema in the data server can be accessed by the integration processes.

2.1.4 Agents

Oracle Data Integrator run-time Agents orchestrate the execution of jobs. These agents are Java components.

The run-time agent functions as a *listener* and a *scheduler* agent. The agent executes jobs on demand (model reverses, packages, scenarios, mappings, and so forth), for example

when the job is manually launched from a user interface or from a command line. The agent is also used to start the execution of scenarios according to a schedule defined in Oracle Data Integrator.

Third party scheduling systems can also trigger executions on the agent. See "Scheduling a Scenario or a Load Plan with an External Scheduler" in *Administering Oracle Data Integrator* for more information.

Typical projects only require a single Agent in production; however, "Load balancing Agents" in *Administering Oracle Data Integrator* describes how to set up multiple load-balanced agents.

ODI Studio can also directly execute jobs on demand. This internal "agent" can be used for development and initial testing. However, it does not have the full production features of external agents, and is therefore unsuitable for production data integration. When running a job, in the **Run** dialog, select `Local (No Agent)` as the **Logical Agent** to directly execute the job using ODI Studio. Note the following features are not available when running a job locally:

- Stale session cleanup
- Ability to stop a running session
- Load balancing

If you need any of these features, you should use an external agent.

Agent Lifecycle

The lifecycle of an agent is as follows:

1. When the agent starts it connects to the master repository.
2. Through the master repository it connects to any work repository attached to the Master repository and performs the following tasks at startup:
 - Execute any outstanding tasks in all work repositories that need to be executed upon startup of this agent.
 - Clean stale sessions in each work repository. These are the sessions left incorrectly in a running state after an agent or repository crash.
 - Retrieve its list of scheduled scenarios in each work repository, and compute its schedule.
3. The agent starts listening on its port.
 - When an execution request is received by the agent, the agent acknowledges this request and starts the session.
 - The agent launches sessions according to the schedule.
 - The agent is also able to process other administrative requests in order to update its schedule, stop a session, respond to a ping, or clean stale sessions. The standalone agent can also process a stop signal to terminate its lifecycle.

Refer to "Running Integration Processes" in *Administering Oracle Data Integrator* for more information about a session lifecycle.

Agent Features

Agents are not data transformation servers. They do not perform any data transformation, but instead only orchestrate integration processes. They delegate data transformation to database servers, operating systems, and scripting engines.

Agents are multi-threaded lightweight components. An agent can run multiple sessions in parallel. When declaring a physical agent, Oracle recommends that you adjust the maximum number of concurrent sessions it is allowed to execute simultaneously from a work repository. When this maximum number is reached, any new incoming session will be queued by the agent and executed later when other sessions have terminated. If you plan to run multiple parallel sessions, you can consider load balancing executions, as described in "Load balancing Agents" in *Administering Oracle Data Integrator*.

Agent Types

Oracle Data Integrator agents are available with three types: standalone agents, standalone colocated agents, and Java EE agents.

For more information about agent types, see: "Run-Time Agent" in *Understanding Oracle Data Integrator*.

Physical and Logical Agents

A physical agent corresponds to a single standalone agent or a Java EE agent. A physical agent should have a unique name in the Topology.

Similarly to schemas, physical agents having an identical role in different environments can be grouped under the same logical agent. A logical agent is related to physical agents through contexts. When starting an execution, you indicate the logical agent and the context. Oracle Data Integrator will translate this information into a single physical agent that will receive the execution request.

Agent URL

An agent runs on a *host* and a *port* and is identified on this port by an *application name*. The agent URL also indicates the *protocol* to use for the agent connection. Possible values for the protocol are `http` or `https`. These four components make the agent URL. The agent is reached using this URL.

For example:

- A standalone agent started on port 8080 on the `odi_production` machine will be reachable at the following URL:

```
http://odi_production:8080/oraclediagent.
```

Note: The application name for a standalone agent is always **oraclediagent** and cannot be changed.

- A Java EE agent started as an application called `oracledi` on port 8000 in a WLS server deployed on the `odi_wls` host will be reachable at the following URL:

```
http://odi_wls:8000/oracledi.
```

Apache Oozie

Apache Oozie is a workflow scheduler that helps you manage Apache Hadoop jobs. It is a server-based Workflow Engine specialized in running workflow jobs with actions that run Hadoop MapReduce jobs. Refer to *Integrating Big Data with Oracle Data Integrator* for more information.

2.1.5 Languages

Languages defines the programming and scripting languages, and language elements, available when creating and editing expressions during integration development. Languages provided by default in Oracle Data Integrator do not require any user change.

2.1.6 Repositories

The topology contains information about the Oracle Data Integrator repositories. Repository definition, configuration and installation is described in "Creating the Oracle Data Integrator Master and Work Repository Schema" in *Installing and Configuring Oracle Data Integrator*.

Part II

Managing and Reverse-Engineering Metadata

This part describes how to manage and reverse-engineer metadata in Oracle Data Integrator.

This part contains the following chapters:

- [Chapter 3, "Creating and Using Data Models and Datastores"](#)
- [Chapter 4, "Using Journalizing"](#)
- [Chapter 5, "Creating Data Models with Common Format Designer"](#)

Creating and Using Data Models and Datastores

This chapter describes how to create a model, how to reverse-engineer this model to populate it with datastores and how to create manually datastores of a model. This chapter also explains how to use partitioning and check the quality of the data in a model.

This chapter includes the following sections:

- [Introduction to Models](#)
- [Creating and Reverse-Engineering a Model](#)
- [Creating and Reverse-Engineering a Datastore](#)
- [Editing and Viewing a Datastore's Data](#)
- [Using Partitioning](#)
- [Checking Data Quality in a Model](#)

3.1 Introduction to Models

A Model is the description of a set of datastores. It corresponds to a group of tabular data structures stored in a data server. A model is based on a Logical Schema defined in the topology. In a given Context, this Logical Schema is mapped to a Physical Schema. The Data Schema of this Physical Schema contains physical data structure: tables, files, JMS messages, elements from an XML file, that are represented as datastores.

Models as well as all their components are based on the relational paradigm (table, attributes, keys, etc.). Models in Data Integrator only contain *Metadata*, that is the description of the data structures. They do not contain a copy of the actual data.

Note: Frequently used technologies have their reverse and model creation methods detailed in the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator*.

Models can be organized into model folders and the datastores of a model can be organized into sub-models. "[Organizing Models with Folders](#)" on page 18-2 describes how to create and organize model folders and sub-models.

3.1.1 Datastores

A datastore represents a data structure. It can be a table, a flat file, a message queue or any other data structure accessible by Oracle Data Integrator.

A datastore describes data in a tabular structure. Datastores are composed of attributes.

As datastores are based on the relational paradigm, it is also possible to associate the following elements to a datastore:

- **Keys**

A Key is a set of attributes with a specific role in the relational paradigm. Primary and Alternate Keys identify each record uniquely. Non-Unique Indexes enable optimized record access.
- **References**

A Reference is a functional link between two datastores. It corresponds to a Foreign Key in a relational model. For example: The INVOICE datastore references the CUSTOMER datastore through the customer number.
- **Conditions and Filters**

Conditions and Filters are a WHERE-type SQL expressions attached to a datastore. They are used to validate or filter the data in this datastore.

3.1.2 Data Integrity

A model contains constraints such as Keys, References or Conditions, but also non-null flags on attributes. Oracle Data Integrator includes a data integrity framework for ensuring the quality of a data model.

This framework allows to perform:

- **Static Checks** to verify the integrity of the data contained in a data model. This operation is performed to assess the quality of the data in a model when constraints do not physically exist in the data server but are defined in Data Integrator only.
- **Flow Check** to verify the integrity of a data flow before it is integrated into a given datastore. The data flow is checked against the constraints defined in Oracle Data Integrator for the datastore that is the target of the data flow.

3.1.3 Reverse-engineering

A new model is created with no datastores. Reverse-engineering is the process that populates the model in Oracle Data Integrator by retrieving metadata from the data server containing the data structures. There are two different types of reverse-engineering:

- **Standard reverse-engineering** uses standard JDBC driver features to retrieve the metadata. Note that unique keys are not reverse-engineered when using a standard reverse-engineering.
- **Customized reverse-engineering** uses a technology-specific Reverse Knowledge Module (RKM) to retrieve the metadata, using a method specific to the given technology. This method is recommended if a technology specific RKM exists because it usually retrieves more information than the Standard reverse-engineering method. See the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator* for a list of available RKMs.

Other methods for reverse-engineering exist for flat file datastores. They are detailed in ["Reverse-Engineering File Datastores"](#) on page 3-8.

Oracle Data Integrator is able to reverse-engineer models containing datastore shortcuts. For more information, see [Chapter 15, "Using Shortcuts"](#).

3.1.4 Changed Data Capture

Change Data Capture (CDC), also referred to as *Journalizing*, allows to trap changes occurring on the data. CDC is used in Oracle Data Integrator to eliminate the transfer of unchanged data. This feature can be used for example for data synchronization and replication.

Journalizing can be applied to models, sub-models or datastores based on certain type of technologies.

For information about setting up Changed Data Capture, see [Chapter 4, "Using Journalizing"](#).

3.2 Creating and Reverse-Engineering a Model

Now that the key components of an ODI model have been described, an overview is provided on how to create and reverse-engineer a model:

- [Creating a Model](#)
- [Creating a Model and Topology Objects](#)
- [Reverse-engineering a Model](#)

3.2.1 Creating a Model

A Model is a set of datastores corresponding to data structures contained in a Physical Schema.

Tip: To create a new model and new topology objects at the same time, use the procedure described in ["Creating a Model and Topology Objects"](#) on page 3-4.

To create a Model:

1. In Designer Navigator expand the **Models** panel.
2. Right-click then select **New Model**.
3. Fill in the following fields in the Definition tab:
 - **Name:** Name of the model used in the user interface.
 - **Technology:** Select the model's technology.
 - **Logical Schema:** Select the Logical Schema on which your model will be based.
4. On the Reverse Engineer tab, select a **Context** which will be used for the model's reverse-engineering.

Note that if there is only one context that maps the logical schema, this context will be set automatically.

5. Select **Save** from the File main menu.

The model is created, but contains no datastore yet.

3.2.2 Creating a Model and Topology Objects

You can create a Model along with other topology objects, including new data servers, contexts and schemas, at the same time:

1. In ODI Studio, select **File** and click **New...**
2. In the **New Gallery** dialog, select **Create a New Model and Topology Objects** and click **OK**.

The **Create New Model and Topology Objects** wizard appears.

3. In the **Model** panel of the wizard, fill in the following fields.
 - **Name:** Name of the model used in the user interface.
 - **Technology:** Select the model's technology.
 - **Logical Schema:** Select the Logical Schema on which your model will be based.
 - **Context:** Select the context that will be used for the model's reverse-engineering.
4. Click **Next**.
5. In the **Data Server** panel of the wizard, fill in the following data server fields.
 - **Name:** Name of the data server, as it appears in the user interface.
Note: or naming data servers, it is recommended to use the following naming standard: <TECHNOLOGY_NAME>_<SERVER_NAME>.
 - **Technology:** Technology linked to the data server.
Note: Appears only if the **Technology** selected for the Model is of the type Generic SQL.
 - **User:** User name used for connecting to the data server.
 - **Password:** Password linked with the user name.
Note: This password is stored encrypted in the repository.
 - **Driver List:** Provides a list of available drivers available to be used with the data server.
 - **Driver:** Name of the driver used for connecting to the data server.
 - **URL:** Provides the connection details.
 - **Properties:** Lists the properties that you can set for the selected driver.
6. Click **Next**.
7. In the **Physical Schema** panel of the wizard, fill in the physical schema fields.
 - **Name:** Name of the physical schema. It is calculated automatically and is read-only.
 - **Datasource (Catalog):** Name of the catalog in the data server.
Note: Appears only if the **Technology** selected supports Catalogs.
 - **Schema (Schema):** Name of the schema in the data server. Schema, owner, or library where the required data is stored.
Note: Oracle Data Integrator lists all the schemas present in the data server. Sometimes, Oracle Data Integrator cannot draw up this list. In this case, you should enter the schema name, respecting the case.

- **Datasource (Work Catalog):** Indicate the catalog in which you want to create these objects. For some data validation or transformation operations, Oracle Data Integrator may require work objects to be created.

Note: Appears only if the **Technology** selected supports Catalogs.

- **Schema (Work Schema):** Indicates the schema in which you want to create these objects. For some data validation or transformation operations, Oracle Data Integrator may require work objects to be created. If you do not set a **Work Schema**, it defaults to the **Schema** during execution

It is recommended that you create a specific schema dedicated to any work tables. By creating a schema named SAS or ODI in all your data servers, you ensure that all Oracle Data Integrator activity remains totally independent from your applications.

Note: Oracle Data Integrator lists all the schemas present in the data server. Sometimes, Oracle Data Integrator cannot draw up this list. In this case, you should enter the schema name, respecting the case.

- **Driver:** Name of the driver used for connecting to the data server.
- **URL:** Provides the connection details.
- **Properties:** Specifies the properties for the selected driver.

8. Click **Next**.

9. Click **Finish**. The model and topology objects are created, but the model contains no datastore yet.

3.2.3 Reverse-engineering a Model

To automatically populate datastores into the model you need to perform a reverse-engineering for this model.

Standard Reverse-Engineering

A Standard Reverse-Engineering uses the capacities of the JDBC driver used to connect the data server to retrieve the model metadata.

To perform a Standard Reverse- Engineering:

1. In the **Reverse Engineer** tab of your Model:
 - Select **Standard**.
 - Select the **Context** used for the reverse-engineering
 - Select the **Types of objects to reverse-engineer**. Only object of these types will be taken into account by the reverse-engineering process.
 - Enter in the **Mask** field the mask of tables to reverse engineer. The mask selects the objects to reverse. This mask uses the SQL LIKE syntax. The percent (%) symbol means zero or more characters, and the underscore (_) symbol means one character.
 - Optionally, you can specify the **characters to remove for the table alias**. These are the characters to delete in order to derive the alias. Note that if the datastores already exist, the characters specified here will not be removed from the table alias. Updating a datastore is not applied to the table alias.
2. In the **Selective Reverse-Engineering** tab select **Selective Reverse-Engineering, New Datastores, Existing Datastores** and **Objects to Reverse Engineer**.

3. A list of datastores to be reverse-engineered appears. Leave those you wish to reverse-engineer checked.
4. Select **Save** from the File main menu.
5. Click **Reverse Engineer** in the Model toolbar menu.
6. Oracle Data Integrator launches a reverse-engineering process for the selected datastores. A progress bar indicates the progress of the reverse-engineering process.

The reverse-engineered datastores appear under the model node in the **Models** panel.

Customized Reverse-Engineering

A Customized Reverse-Engineering uses a Reverse-engineering Knowledge Module (RKM), to retrieve metadata for a specific type of technology and create the corresponding datastore definition in the data model.

For example, for the Oracle technology, the RKM Oracle accesses the database dictionary tables to retrieve the definition of tables, attributes, keys, etc., that are created in the model.

Note: The RKM must be available as a global RKM or imported into the project. Refer to [Chapter 6, "Creating an Integration Project,"](#) for more information on KM import.

To perform a Customized Reverse-Engineering using a RKM:

1. In the **Reverse Engineer** tab of your Model:
 - Select **Customized**.
 - Select the **Context** used for the reverse-engineering
 - Select the **Types of objects to reverse-engineer**. Only object of these types will be taken into account by the reverse-engineering process.
 - Enter in the **Mask** the mask of tables to reverse engineer.
 - Select the KM that you want to use for performing the reverse-engineering process. This KM is typically called `RKM <technology>.<name of the project>`.
 - Optionally, you can specify the **characters to remove for the table alias**. These are the characters to delete in order to derive the alias. Note that if the datastores already exist, the characters specified here will not be removed from the table alias. Updating a datastore is not applied to the table alias.
2. Click **Reverse Engineer** in the Model toolbar menu, then **Yes** to validate the changes.
3. Click **OK**.
4. The **Session Started Window** appears.
5. Click **OK**.

You can review the reverse-engineering tasks in the Operator Navigator. If the reverse-engineering process completes correctly, reverse-engineered datastores appear under the model node in the **Models** panel.

3.3 Creating and Reverse-Engineering a Datastore

Although the recommended method for creating datastores in a model is reverse-engineering, it is possible to manually define datastores in a blank model. It is the recommended path for creating flat file datastores.

3.3.1 Creating a Datastore

To create a datastore:

1. From the Models tree in Designer Navigator, select a Model or a Sub-Model.
2. Right-click and select **New Datastore**.
3. In the **Definition** tab, fill in the following fields:
 - **Name of the Datastore:** This is the name that appears in the trees and that is used to reference the datastore from a project.

Note: Do not use ODI reserved names like, for example, `JRN_FLAG`, `JRN_SUBSCRIBER`, and `JRN_DATE` for the datastore name. These names would cause Duplicate Attribute name SQL errors in ODI intermediate tables such as error tables.

- **Resource Name:** Name of the object in the form recognized by the data server which stores it. This may be a table name, a file name, the name of a JMS Queue, etc.

Note: If the Resource is a database table, it must respect the Database rules for object and attributes identifiers. There is a limit in the object identifier for most of the Technologies (in Oracle, typically 30 characters). To avoid these errors, ensure in the topology for a specific technology that maximum lengths for the object names (tables and columns) correspond to your database configuration.

- **Alias:** This is a default alias used to prefix this datastore's attributes names in expressions.
4. If the datastore represents a flat file (delimited or fixed), in the **File** tab, fill in the following fields:
 - **File Format:** Select the type of your flat file, fixed or delimited.
 - **Header:** Number of header lines for the flat file.
 - **Record Separator** and **Field Separator** define the characters used to separate records (lines) in the file, and fields within one record.

Record Separator: One or several characters separating lines (or records) in the file:

- MS-DOS: DOS carriage return
- Unix: UNIX carriage return
- Other: Free text you can input as characters or hexadecimal codes

Field Separator: One or several characters separating the fields in a record.

- Tabulation

- Space
- Other: Free text you can input as characters or hexadecimal code

5. Select **Save** from the File main menu.

The datastore is created. If this is a File datastore, refer to [Reverse-Engineering File Datastores](#) for creating attributes for this datastore. It is also possible to manually edit attributes for all datastores. See [Adding and Deleting Datastore Attributes](#) for more information.

3.3.2 Reverse-Engineering File Datastores

Oracle Data Integrator provides specific methods for reverse-engineering flat files. The methods for reversing flat files are described below.

3.3.2.1 Reverse-Engineering Fixed Files

Fixed files can be reversed engineered using a wizard into which the boundaries of the fixed attributes and their parameters can be defined.

1. Go to the **Attributes** tab the file datastore that has a fixed format.
2. Click the **Reverse Engineer** button. A window opens displaying the first records of your file.
3. Click on the ruler (above the file contents) to create markers delimiting the attributes. Right-click in the ruler to delete a marker.
4. Attributes are created with pre-generated names (C1, C2, and so on). You can edit the attribute name by clicking in the attribute header line (below the ruler).
5. In the properties panel (on the right), you can edit the parameters of the selected attribute.
6. You must set at least the **Attribute Name**, **Datatype** and **Length** for each attribute. Note that attribute names of File datastores cannot contain spaces.
7. Click **OK** when the attributes definition is complete to close the wizard.
8. Select **Save** from the File main menu.

3.3.2.2 Reverse-Engineering Delimited Files

Delimited files can be reversed engineered using a a built-in JDBC which analyzes the file to detect the attributes and reads the attribute names from the file header.

1. Go to the **Attributes** tab the file datastore that has a delimited format.
2. Click the **Reverse Engineer** button.
3. Oracle Data Integrator creates the list of attributes according to your file content. The attribute type and length are set to default values. Attribute names are pre-generated names (C1, C2, and so on) or names taken from the first Header line declared for this file.
4. Review and if needed modify the **Attribute Name**, **Datatype** and **Length** for each attribute. Note that attribute names of File datastores cannot contain spaces.
5. Select **Save** from the File main menu.

3.3.2.3 Reverse-Engineering COBOL Files

Fixed COBOL files structures are frequently described in Copybook files. Oracle Data Integrator can reverse-engineer the Copybook file structure into a datastore structure.

1. Go to the **Attributes** tab the file datastore that has a fixed format.
2. Click the **Reverse Engineer COBOL Copybook** button.
3. Fill in the following fields:
 - **File**: Location of the Copybook file.
 - **Character Set**: Copybook file character set.
 - **Description format** (EBCDIC or ASCII): Copybook file format
 - **Data format** (EBCDIC or ASCII): Data file format.
4. Click **OK**. The attributes described in the Copybook are reverse-engineered and appear in the attribute list.
5. Select **Save** from the File main menu.

3.3.3 Adding and Deleting Datastore Attributes

To add attributes to a datastore:

1. In the **Attributes** tab of the datastore, click **Add Attribute** in the tool bar menu.
2. An empty line appears. Fill in the information about the new attribute. You should at least fill in the **Name**, **Datatype** and **Length** fields.
3. Repeat steps 1 and 2 for each attribute you want to add to the datastore.
4. Select **Save** from the File main menu.

To delete attributes from a datastore:

1. In the **Attributes** tab of the datastore, select the attribute to delete.
2. Click the **Delete Attribute** button. The attribute disappears from the list.

3.3.4 Adding and Deleting Constraints and Filters

Oracle Data Integrator manages constraints on data model including Keys, References, Conditions and Mandatory Attributes. It includes a data integrity framework for ensuring the quality of a data model based on these constraints.

Filters are not constraints but are defined similarly to Conditions. A Filter is not used to enforce a data quality rule on a datastore, but is used to automatically filter this datastore when using it as a source.

3.3.4.1 Keys

To create a key for a datastore:

1. In the Designer Navigator, expand in the **Model** tree the model and then the datastore into which you want to add the key.
2. Select the **Constraints** node, right-click and select **New Key**.
3. Enter the **Name** for the constraint, and then select the **Key or Index Type**. Primary Keys and Alternate Keys can be checked and can act as an update key in an interface. Non-Unique Index are used mainly for performance reasons.
4. In the **Attributes** tab, select the list of attributes that belong to this key.
5. In the **Control** tab, select whether this constraint should be checked by default in a **Static** or **Flow** check.

6. By clicking the **Check** button, you can retrieve the number of records that do not respect this constraint.
7. Select **Save** from the File main menu.

3.3.4.2 References

To create a reference between two datastores:

1. In the Designer Navigator, expand in the **Model** tree the model and then one of the datastores into which you want to add the reference.
2. Select the **Constraints** node, right-click and select **New Reference**.
3. Enter the **Name** for the constraint, and then select the **Type** for the reference. In a **User Reference** the two datastores are linked based on attribute equality. In a **Complex User Reference** any expression can be used to link the two datastores. A **Database Reference** is a reference based on attribute equality that has been reverse-engineered from a database engine.
4. If you want to reference a datastore that exists in a model, select the **Model** and the **Table** that you want to link to the current datastore.
5. If you want to link a table that does not exist in a model, leave the **Model** and **Table** fields undefined, and set the **Catalog**, **Schema** and **Table** names to identify your datastore.
6. If you are defining a User or Database reference, in the **Attributes** tab, define the matching attributes from the two linked datastores.
7. If you are defining a Complex User reference, enter in the **Expression** tab the expression that relates attributes from the two linked datastores.
8. In the **Control** tab, select whether this constraint should be checked by default in a **Static** or **Flow** check.
9. By clicking the **Check** button, you can retrieve the number of records that respect or do not respect this constraint.
10. Select **Save** from the File main menu.

3.3.4.3 Conditions

To create a condition for a datastore:

1. In the Designer Navigator, expand in the **Model** tree the model and then one of the datastores into which you want to add the condition.
2. Select the **Constraints** node, right-click and select **New Condition**.
3. Enter the **Name** for the constraint, and then select the **Type** for the condition. An **Oracle Data Integrator Condition** is a condition that exists only in the model and does not exist in the database. A **Database Condition** is a condition that is defined in the database and has been reverse-engineered.
4. In the **Where** field enter the expression that implements the condition. This expression is a SQL WHERE expression that valid records should respect.
5. Type in the **Message** field the error message for this constraint.
6. In the **Control** tab, select whether this constraint should be checked by default in a **Static** or **Flow** check.
7. By clicking the **Check** button, you can retrieve the number of records that do not respect this constraint.

8. Select **Save** from the File main menu.

3.3.4.4 Mandatory Attributes

To define mandatory attributes for a datastore:

1. In the Designer Navigator, expand in the **Model** tree the model containing the datastores.
2. Double-click the datastore containing the attribute that must be set as mandatory. The **Datastore** Editor appears.
3. In the **Attributes** tab, check the **Not Null** field for each attribute that is mandatory.
4. Select **Save** from the File main menu.

3.3.4.5 Filter

To add a filter to a datastore:

1. In the Designer Navigator, expand in the **Model** tree the model and then one of the datastores into which you want to add the filter.
2. Select the **Filter** node, right-click and select **New Condition**.
3. Enter the **Name** for the filter.
4. In the **Where** field enter the expression that implements the filter. This expression is a SQL WHERE expression used to filter source records.
5. In the **Control** tab, check **Filter Active for Static Control** if you want data from this table to be filtered prior to checking it a static control.
6. Select **Save** from the File main menu.

3.4 Editing and Viewing a Datastore's Data

To view a datastore's data:

1. Select the datastore from the model in the Designer Navigator.
2. Right-click and select **View Data**.

The data appear in a non editable grid.

To edit a datastore's data:

1. Select the datastore from the model in the Designer Navigator.
2. Right-click and select **Data...**

The data appear in an editable grid in the Data Editor. The **Refresh** button enables you to edit and run again the query returning the datastore data. You can filter the data and perform free queries on the datastore using this method.

It is possible to edit a datastore's data if the connectivity used and the data server user's privileges allow it, and if the datastore structure enables to identify each row of the datastore (PK, etc.).

Note: The data displayed is the data stored in the physical schema corresponding to the model's logical schema, in the current working context.

3.5 Using Partitioning

Oracle Data Integrator is able to use database-defined partitions when processing data in partitioned tables used as source or targets of mappings. These partitions are created in the datastore corresponding to the table, either through the reverse-engineering process or manually. For example with the Oracle technology, partitions are reverse-engineered using the RKM Oracle.

The partitioning methods supported depend on the technology of the datastore. For example, for the Oracle technology the following partitioning methods are supported: Range, Hash, List.

Once defined on a datastore, partitions can be selected when this datastore is used as a source or a target of a mapping. Refer to [Chapter 8, "Creating and Using Mappings,"](#) for information.

If using the common format designer, you can also create partitions when performing the Generate DDL operation.

3.5.1 Manually Defining Partitions and Sub-Partitions of Model Datastores

Partition information can be reverse-engineered along with the datastore structures or defined manually.

Note: Standard reverse-engineering does not support the reverse-engineering of partitions. To reverse-engineer partitions and sub-partitions, you have to use customized reverse-engineering.

To manually define partitions and sub-partitions for a datastore:

1. In the Models accordion, double-click the datastore for which you want to define the partition or sub-partition. The Datastore Editor opens.
2. In the **Partitions** tab, enter the following details to define the partition and sub-partition:
 - **Partition by**
Select the partitioning method. This list displays the partitioning methods supported by the technology on which the model relies.
 - **Sub-Partition by**
If you want to define sub-partitions in addition to partitions, select the sub-partitioning method. This list displays the partitioning methods supported by the technology on which the model relies.
3. Click **Add Partition**.
4. In the **Name** field, enter a name for the partition, for example: FY08.
5. In the **Description** field, enter a description for the partition, for example: Operations for Fiscal Year 08.
6. If you want to add:
 - additional partitions, repeat steps 3 through 5.
 - a sub-partition of a selected partition, click **Add Sub-Partition** and repeat steps 4 and 5.
7. From the File menu, select **Save**.

3.6 Checking Data Quality in a Model

Data Quality control is essential in ensuring the overall consistency of the data in your information system's applications.

Application data is not always valid for the constraints and declarative rules imposed by the information system. You may, for instance, find orders with no customer, or order lines with no product, etc. In addition, such incorrect data may propagate via integration flows.

3.6.1 Introduction to Data Integrity

Oracle Data Integrator provides a working environment to detect these constraint violation and store them for recycling or reporting purposes.

There are two different main types of controls: **Static Control** and **Flow Control**. We will examine the differences between the two.

Static Control

Static Control implies the existence of rules that are used to verify the integrity of your application data. Some of these rules (referred to as constraints) may already be implemented in your data servers (using primary keys, reference constraints, etc.)

With Oracle Data Integrator, you can refine the validation of your data by defining additional constraints, without implementing them directly in your servers. This procedure is called **Static Control** since it allows you to perform checks directly on existing - or static - data. Note that the active database constraints (these are those that have **Defined in the Database** and **Active** selected on the Controls tab) need no additional control from Oracle Data Integrator since they are already controlled by the database.

Flow Control

The information systems targeted by transformation and integration processes often implement their own declarative rules. The **Flow Control** function is used to verify an application's incoming data according to these constraints before loading the data into these targets. Setting up flow control is detailed in to [Chapter 8, "Creating and Using Mappings."](#)

3.6.2 Checking a Constraint

While creating a constraint in Oracle Data Integrator, it is possible to retrieve the number of lines violating this constraint. This action, referred as **Synchronous Control** is performed from the **Control** tab of the given constraint Editor by clicking the **Check** button.

The result of a synchronous control is not persistent. This type of control is used to quickly evaluate the validity of a constraint definition.

3.6.3 Perform a Static Check on a Model, Sub-Model or Datastore

To perform a Static Check on a Model, Sub-Model or Datastore:

1. In the **Models** tree in the Designer Navigator, select the model that you want to check.
2. Double-click this model to edit it.

3. In the **Control** tab of the model Editor, select the Check Knowledge Module (CKM) used in the static check.
4. From the File menu, select **Save All**.
5. Right-click the model, sub-model or datastore that you want to check in the **Model** tree in the Designer Navigator and select **Control > Check**. Or, in the model editor menu bar, click the **Check Model** button.
6. In the **Run** dialog, select the execution parameters:
 - Select the **Context** into which the step must be executed.
 - Select the **Logical Agent** that will run the step.
 - Select a **Log Level**.
 - Check the **Delete Errors from the Checked Tables** option if you want rows detected as erroneous to be removed from the checked tables.
 - Select **Recurse Sub-Models** to check sub-models of this models
 - Optionally, select **Simulation**. This option performs a simulation of the run operation and generates a run report, without actually affecting data.

See "Execution Parameters" in *Administering Oracle Data Integrator* for more information about the execution parameters.
7. Click **OK**.
8. The **Session Started Window** (or, if running a simulation, the **Simulation** window) appears.
9. Click **OK**.

You can review the check tasks in the Operator Navigator. If the control process completes correctly, you can review the erroneous records for each datastore that has been checked.

3.6.4 Reviewing Erroneous Records

To view a datastore's errors:

1. Select the datastore from the model in the Designer Navigator.
2. Right-click and select **Control > Errors....**

The erroneous rows detected for this datastore appear in a grid.

Using Journalizing

This chapter describes how to use Oracle Data Integrator's Changed Data Capture feature to detect changes occurring on the data and only process these changes in the integration flows.

This chapter includes the following sections:

- [Introduction to Changed Data Capture](#)
- [Setting up Journalizing](#)
- [Using Changed Data](#)

4.1 Introduction to Changed Data Capture

Changed Data Capture (CDC) allows Oracle Data Integrator to track changes in source data caused by other applications. When running mappings, thanks to CDC, Oracle Data Integrator can avoid processing unchanged data in the flow.

Reducing the source data flow to only changed data is useful in many contexts, such as data synchronization and replication. It is essential when setting up an event-oriented architecture for integration. In such an architecture, applications make changes in the data ("Customer Deletion," "New Purchase Order") during a business process. These changes are captured by Oracle Data Integrator and transformed into events that are propagated throughout the information system.

Changed Data Capture is performed by journalizing models. Journalizing a model consists of setting up the infrastructure to capture the changes (inserts, updates and deletes) made to the records of this model's datastores.

Oracle Data Integrator supports two journalizing modes:

- **Simple Journalizing** tracks changes in individual datastores in a model.
- **Consistent Set Journalizing** tracks changes to a group of the model's datastores, taking into account the referential integrity between these datastores. The group of datastores journalized in this mode is called a **Consistent Set**.

Notes:

- The Changed Data Capture functionality described in this chapter is only used for tracking and acting on changes in the source data. You must still perform initial data loading, if you want to populate a source datastore with data.
 - Changed Data Capture is often performed using Oracle GoldenGate technology. For more information, see "Oracle GoldenGate" in *Connectivity and Knowledge Modules Guide for Oracle Data Integrator*.
-

4.1.1 The Journalizing Components

The journalizing components are:

- **Journals:** Where changes are recorded. Journals only contain references to the changed records along with the type of changes (insert/update, delete).
- **Capture processes:** Journalizing captures the changes in the source datastores either by creating triggers on the data tables, or by using database-specific programs to retrieve log data from data server log files. See the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator* for more information on the capture processes available for the technology you are using.
- **Subscribers:** CDC uses a publish/subscribe model. Subscribers are entities (applications, integration processes, etc.) that use the changes tracked on a datastore or on a consistent set. They subscribe to a model's CDC to have the changes tracked for them. Changes are captured only if there is at least one subscriber to the changes. When all subscribers have consumed the captured changes, these changes are discarded from the journals.
- **Journalizing views:** Provide access to the changes and the changed data captured. They are used by the user to view the changes captured, and by integration processes to retrieve the changed data.

These components are implemented in the journalizing infrastructure.

4.1.2 Simple vs. Consistent Set Journalizing

Simple Journalizing enables you to journalize one or more datastores. Each journalized datastore is treated separately when capturing the changes.

This approach has a limitation, illustrated in the following example: You want to process changes in the ORDER and ORDER_LINE datastores (with a referential integrity constraint based on the fact that an ORDER_LINE record should have an associated ORDER record). If you have captured insertions into ORDER_LINE, you have no guarantee that the associated new records in ORDERS have also been captured. Processing ORDER_LINE records with no associated ORDER records may cause referential constraint violations in the integration process.

Consistent Set Journalizing provides the guarantee that when you have an ORDER_LINE change captured, the associated ORDER change has been also captured, and vice versa. Note that consistent set journalizing guarantees the consistency of the captured changes. The set of available changes for which consistency is guaranteed is called the **Consistency Window**. Changes in this window should be processed in the correct sequence (ORDER followed by ORDER_LINE) by designing and sequencing mappings into packages.

Although consistent set journalizing is more powerful, it is also more difficult to set up. It should be used when referential integrity constraints need to be ensured when capturing the data changes. For performance reasons, consistent set journalizing is also recommended when a large number of subscribers are required.

It is not possible to journalize a model (or datastores within a model) using both consistent set and simple journalizing.

4.2 Setting up Journalizing

This section explains how to set up and start the journalizing infrastructure, and check that this infrastructure is running correctly. It also details the components of this infrastructure.

4.2.1 Setting up and Starting Journalizing

The basic process for setting up CDC on an Oracle Data Integrator data model is as follows:

- Set the CDC parameters in the data model
- Add the datastores to the CDC
- For consistent set journalizing, set the datastores order
- Add subscribers
- Start the journals

Set the CDC parameters

Setting up the CDC parameters is performed on a data model. This consists of selecting or changing the journalizing mode and journalizing Knowledge Module used for the model.

To set up the CDC parameters:

1. In the **Models** tree in the Designer Navigator, select the model that you want to journalize.
2. Double-click this model to edit it.
3. In the **Journalizing** tab, select the journalizing mode you want to use: **Consistent Set** or **Simple**.
4. Select the Journalizing Knowledge Module (JKM) you want to use for this model. Only Knowledge Modules suitable for the data model's technology and journalizing mode, and that have been previously imported into at least one of your projects will appear in the list.
5. Set the **Options** for this KM. See the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator* for more information about this KM and its options.
6. From the File menu, select **Save All**.

Note: If the model is already being journalized, it is recommended that you stop journalizing with the existing configuration before modifying the data model journalizing parameters.

Add or remove datastores for the CDC:

You must flag the datastores that you want to journalize within the journalized model. A change in the datastore flag is taken into account the next time the journals are (re)started. When flagging a model or a sub-model, all of the datastores contained in the model or sub-model are flagged.

To add or remove datastores for the CDC:

1. Right-click the model, sub-model or datastore that you want to add to/remove from the CDC in the **Model** tree in the Designer Navigator.
2. Right-click then select **Changed Data Capture > Add to CDC** or **Changed Data Capture > Remove from CDC** to add to the CDC or remove from the CDC the selected datastore, or all datastores in the selected model/sub-model.

The datastores added to CDC should now have a marker icon. The journal icon represents a small clock. It should be yellow, indicating that the journal infrastructure is not yet in place.

Note: It is possible to add datastores to the CDC after the journal creation phase. In this case, the journals should be re-started.

If a datastore with journals running is removed from the CDC in simple mode, the journals should be stopped for this individual datastore. If a datastore is removed from CDC in Consistent Set mode, the journals should be restarted for the model (Journalizing information is preserved for the other datastores).

Set the datastores order (consistent set journalizing only):

You only need to arrange the datastores in order when using consistent set journalizing. You should arrange the datastores in the consistent set in an order which preserves referential integrity when using their changed data. For example, if an ORDER table has references imported from an ORDER_LINE datastore (i.e. ORDER_LINE has a foreign key constraint that references ORDER), and both are added to the CDC, the ORDER datastore should come before ORDER_LINE. If the PRODUCT datastore has references imported from both ORDER and ORDER_LINE (i.e. both ORDER and ORDER_LINE have foreign key constraints to the PRODUCT table), its order should be lower still.

To set the datastores order:

1. In the **Models** tree in the Designer Navigator, select the model journalized in consistent set mode.
2. Double-click this model to edit it.
3. Go to the **Journalized Tables** tab.
4. If the datastores are not currently in any particular order, click the **Reorganize** button. This feature suggests an order for the journalized datastores based on the foreign keys defined in the model. Review the order suggested and edit the datastores order if needed.
5. Select a datastore from the list, then use the Up and Down buttons to move it within the list. You can also directly edit the **Order** value for this datastore.
6. Repeat the previous step until the datastores are ordered correctly.
7. From the File menu, select **Save All**.

Note: Changes to the order of datastores are taken into account the next time the journals are (re)started.

If existing scenarios consume changes from this CDC set, you should regenerate them to take into account the new organization of the CDC set.

Add or remove subscribers:

Each subscriber consumes in a separate thread changes that occur on individual datastores for Simple Journalizing or on a model for Consistent Set Journalizing. Adding or removing a subscriber registers it to the CDC infrastructure in order to trap changes for it.

To add subscribers:

1. In the **Models** tree in the Designer Navigator, select the journalized data model if using Consistent Set Journalizing or select a data model or an individual datastore if using Simple Journalizing.
2. Right-click, then select **Changed Data Capture > Subscriber > Subscribe**. A window appears which lets you select your subscribers.
3. Type a **Subscriber** name, then click the **Add Subscriber** button. Repeat the operation for each subscriber you want to add.

Note: Subscriber names cannot contain single quote characters.

4. Click **OK**.
5. In the **Execution** window, select the execution parameters:
 - Select the **Context** into which the subscribed must be registered.
 - Select the **Logical Agent** that will run the journalizing tasks.
6. Click **OK**.
7. The **Session Started Window** appears.
8. Click **OK**.

You can review the journalizing tasks in the Operator Navigator.

Removing a subscriber is a similar process. Select the **Changed Data Capture > Subscriber > Unsubscribe** option instead.

You can also add subscribers after starting the journals. Subscribers added after journal startup will only retrieve changes captured since they were added to the subscribers list.

Start/Drop the journals:

Starting the journals creates the CDC infrastructure if it does not exist yet. It also validates the addition, removal and order changes for journalized datastores.

Dropping the journals deletes the entire journalizing infrastructure.

Note: Dropping the journals deletes all captured changes as well as the infrastructure. For simple journalizing, starting the journal in addition deletes the journal contents. Consistent Set JKMs support restarting the journals without losing any data.

To start or drop the journals:

1. In the **Models** tree in the Designer Navigator, select the journalized data model if using Consistent Set Journalizing or select a data model or an individual datastore if using Simple Journalizing.
2. Right-click, then select **Changed Data Capture > Start Journal** if you want to start the journals, or **Changed Data Capture > Drop Journal** if you want to stop them.
3. In the **Execution** window, select the execution parameters:
 - Select the **Context** into which the journals must be started or dropped.
 - Select the **Logical Agent** that will run the journalizing tasks.
4. Click **OK**.
5. The **Session Started Window** appears.
6. Click **OK**.

A session begins to start or drops the journals. You can review the journalizing tasks in the Operator Navigator.

Automate journalizing setup:

The journalizing infrastructure is implemented by the journalizing KM at the physical level. Consequently, *Add Subscribers* and *Start Journals* operations should be performed in each context where journalizing is required for the data model. It is possible to automate these operations using Oracle Data Integrator packages. Automating these operations is recommended to deploy a journalized infrastructure across different contexts.

For example, a developer will manually configure CDC in the Development context. When the development phase is complete, he provides a package that automates the CDC infrastructure. CDC is automatically deployed in the Test context by using this package. The same package is also used to deploy CDC in the Production context.

An overview designing such a package follows. See [Chapter 7, "Creating and Using Packages,"](#) for more information on package creation.

To automate CDC configuration:

1. Create a new package.
2. Drag and drop from the **Models** accordion the model or datastore you want to journalize into the package **Diagram** tab. A new package step appears.
3. Double-Click the step icon in the package diagram. The properties inspector for this steps opens.
4. In the **Type** list, select **Journalizing Model/Datastore**.
5. Check the **Start** box to start the journals.
6. Check the **Add Subscribers** box, then enter the list of subscribers into the Subscribers group.

7. Enter the first subscriber in the subscriber field, and click the **Add** button to add it to the **Subscribers** list. Repeat this operation for all your subscribers.
8. From the File menu, select **Save**.

When this package is executed in a context, it starts the journals according to the model configuration and creates the specified subscribers in this context.

It is possible to split subscriber and journal management into different steps and packages. Deleting subscribers and stopping journals can be automated in the same manner.

4.2.2 Journalizing Infrastructure Details

When the journals are started, the journalizing infrastructure (if not installed yet) is deployed or updated in the following locations:

- When the journalizing Knowledge Module creates triggers, they are installed on the tables in the Work Schema for the Oracle Data Integrator physical schema containing the journalized tables. Journalizing trigger names are prefixed with the prefix defined in the Journalizing Elements Prefixes for the physical schema. The default value for this prefix is T\$. For details about database-specific capture processes see the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator*.
- A CDC common infrastructure for the data server is installed in the Work Schema for the Oracle Data Integrator physical schema that is flagged as Default for this data server. This common infrastructure contains information about subscribers, consistent sets, etc. for all the journalized schemas of this data server. This common infrastructure consists of tables whose names are prefixed with SNP_CDC_.
- Journal tables and journalizing views are installed in the Work Schema for the Oracle Data Integrator physical schema containing the journalized tables. The journal table and journalizing view names are prefixed with the prefixes defined in the Journalizing Elements Prefixes for the physical schema. The default value is J\$ for journal tables and JV\$ for journalizing views

All components (except the triggers) of the journalizing infrastructure (like all Data Integrator temporary objects, such as integration, error and loading tables) are installed in the Work Schema for the Oracle Data Integrator physical schemas of the data server. These work schemas should be kept separate from the schema containing the application data (Data Schema).

Important: The journalizing triggers are the only components for journalizing that must be installed, when needed, in the same schema as the journalized data. Before creating triggers on tables belonging to a third-party software package, please check that this operation is not a violation of the software agreement or maintenance contract. Also ensure that installing and running triggers is technically feasible without interfering with the general behavior of the software package.

4.2.3 Journalizing Status

Datastores in models or mappings have an icon marker indicating their journalizing status in Designer's current context:

- **OK** - Journalizing is active for this datastore in the current context, and the infrastructure is operational for this datastore.
- **No Infrastructure** - Journalizing is marked as active in the model, but no appropriate journalizing infrastructure was detected in the current context. Journals should be started. This state may occur if the journalizing mode implemented in the infrastructure does not match the one declared for the model.
- **Remnants** - Journalizing is marked as inactive in the model, but remnants of the journalizing infrastructure such as the journalizing table have been detected for this datastore in the context. This state may occur if the journals were not stopped and the table has been removed from CDC.

4.3 Using Changed Data

Once journalizing is started and changes are tracked for subscribers, it is possible to use the changes captured. These can be viewed or used when the journalized datastore is used as a source of a mapping.

4.3.1 Viewing Changed Data

To view the changed data:

1. In the **Models** tree in the Designer Navigator, select the journalized datastore.
2. Right-click and then select **Changed Data Capture > Journal Data...**

The changes captured for this datastore in the current context appear in a grid with three additional columns describing the change details:

- **JRN_FLAG**: Flag indicating the type of change. It takes the value I for an inserted/updated record and D for a deleted record.
- **JRN_SUBSCRIBER**: Name of the Subscriber.
- **JRN_DATE**: Timestamp of the change.

Journalized data is mostly used within integration processes. Changed data can be used as the source of mappings. The way it is used depends on the journalizing mode.

4.3.2 Using Changed Data: Simple Journalizing

Using changed data from simple journalizing consists of designing mappings using journalized datastores as sources. See [Chapter 8, "Creating and Using Mappings,"](#) for detailed instructions for creating mappings.

Designing Mappings with Simple Journalizing

When a journalized datastore is inserted into a mapping diagram, a **Journalized Data Only** check box appears in this datastore's property panel when viewing it in the Physical tab of the Mapping Editor.

When this box is checked:

- The journalizing columns (**JRN_FLAG**, **JRN_DATE** and **JRN_SUBSCRIBER**) become available for the datastore.
- A **journalizing filter** is also automatically generated on this datastore. This filter will reduce the amount of source data retrieved to the journalized data only. It is always executed on the source. You can customize this filter (for instance, to process changes in a time range, or only a specific type of change). The filter is displayed in the Logical diagram of the Mapping Editor. A typical filter for

retrieving all changes for a given subscriber is: JRN_SUBSCRIBER = '<subscriber_name>'.

In simple journalizing mode all the changes taken into account by the mapping (after the journalizing filter is applied) are automatically considered consumed at the end of the mapping and removed from the journal. They cannot be used by a subsequent mapping.

When processing journalized data, the SYNC_JRN_DELETE option of the integration Knowledge Module should be set carefully. It invokes the deletion from the target datastore of the records marked as deleted (D) in the journals and that are not excluded by the journalizing filter. If this option is set to No, integration will only process inserts and updates.

Note: Only one datastore per dataset can have the **Journalized Data Only** option checked.

4.3.3 Using Changed Data: Consistent Set Journalizing

Using Changed data in Consistent journalizing is similar to simple journalizing for mapping design. It requires extra steps before and after processing the changed data in the mappings in order to enforce changes consistently within the set.

These operations can be performed either manually from the context menu of the journalized model or automated with packages.

Operations Before Using the Changed Data

The following operations should be undertaken before using the changed data when using consistent set journalizing:

- **Extend Window:** The **Consistency Window** is a range of available changes in all the tables of the consistency set for which the insert/update/delete are possible without violating referential integrity. The extend window operation (re)computes this window to take into account new changes captured since the latest Extend Window operation. This operation is implemented using a package step with the **Journalizing Model** Type. This operation can be scheduled separately from other journalizing operations.
- **Lock Subscribers:** Although the extend window is applied to the entire consistency set, subscribers consume the changes separately. This operation performs a subscriber(s) specific "snapshot" of the changes in the consistency window. This snapshot includes all the changes within the consistency window that have not been consumed yet by the subscriber(s). This operation is implemented using a package step with the **Journalizing Model** Type. It should be always performed before the first mapping using changes captured for the subscriber(s).

Designing Mappings

The changed data in consistent set journalizing are also processed using mappings sequenced into packages.

Designing mappings when using consistent set journalizing is similar to simple journalizing, except for the following differences:

- The changes taken into account by the mapping (that is filtered with JRN_FLAG, JRN_DATE and JRN_SUBSCRIBER) are not automatically purged at the end of the mapping. They can be reused by subsequent mappings. The unlock subscriber and

purge journal operations described below are required to commit consumption of these changes, and remove useless entries from the journal respectively.

- In consistent mode, the JRN_DATE column should not be used in the journalizing filter. Using this timestamp to filter the changes consumed does not entirely ensure consistency in these changes.

Operations after Using the Changed Data

After using the changed data, the following operations should be performed:

- **Unlock Subscribers:** This operation commits the use of the changes that were locked during the **Lock Subscribers** operations for the subscribers. It should be processed only after all the changes for the subscribers have been processed. This operation is implemented using a package step with the **Journalizing Model** Type. It should be always performed after the last mapping using changes captured for the subscribers. If the changes need to be processed again (for example, in case of an error), this operation should not be performed.
- **Purge Journal:** After all subscribers have consumed the changes they have subscribed to, entries still remain in the journalizing tables and should be deleted. This is performed by the Purge Journal operation. This operation is implemented using a package step with the **Journalizing Model** Type. This operation can be scheduled separately from the other journalizing operations.

Note: It is possible to perform an Extend Window or Purge Journal on a datastore. These operations process changes for tables that are in the same consistency set at different frequencies. These options should be used carefully, as consistency for the changes may be no longer maintained at the consistency set level

Automate Consistent Set CDC Operations

To automate the consistent set CDC usage, you can use a package performing these operations.

1. Create a new package.
2. Drag and drop from the **Models** tree the journalized model into the package **Diagram** tab. A new package step appears.
3. Double-Click the step icon in the package diagram. The properties inspector for this step opens.
4. In the **Type** list, select **Journalizing Model/Datastore**.
5. Check the consistent set operations you want to perform.
6. If you checked the **Lock Subscriber** or **Unlock Subscriber** operations, enter the first subscriber in the subscriber field, and click the **Add** button to add it to the **Subscribers** list. Repeat this operation for all the subscribers you want to lock or unlock.
7. From the File menu, select **Save All**.

Note: Only one datastore per dataset can have the **Journalized Data Only** option checked.

4.3.4 Journalizing Tools

Oracle Data Integrator provides a set of tools that can be used in journalizing to refresh information on the captured changes or trigger other processes:

- **OdiWaitForData** waits for a number of rows in a table or a set of tables.
- **OdiWaitForLogData** waits for a certain number of modifications to occur on a journalized table or a list of journalized tables. This tool calls `OdiRefreshJournalCount` to perform the count of new changes captured.
- **OdiWaitForTable** waits for a table to be created and populated with a pre-determined number of rows.
- **OdiRetrieveJournalData** retrieves the journalized events for a given table list or CDC set for a specified journalizing subscriber. Calling this tool is required if using Database-Specific Processes to load journalizing tables. This tool needs to be used with specific Knowledge Modules. See the Knowledge Module description for more information.
- **OdiRefreshJournalCount** refreshes the number of rows to consume for a given table list or CDC set for a specified journalizing subscriber.

See *Oracle Data Integrator Tool Reference* for more information on these functions.

4.3.5 Package Templates for Using Journalizing

A number of templates may be used when designing packages to use journalized data. Below are some typical templates. See [Chapter 7, "Creating and Using Packages,"](#) for more information on package creation.

Template 1: One Simple Package (Consistent Set)

- Step 1: Extend Window + Lock Subscribers
- Step 2 to n-1: Mappings using the journalized data
- Step n: Unlock Subscribers + Purge Journal

This package is scheduled to process all changes every minutes. This template is relevant if changes are made regularly in the journalized tables.

Template 2: One Simple Package (Simple Journalizing)

Step 1 to n: Mappings using the journalized data

This package is scheduled to process all changes every minutes. This template is relevant if changes are made regularly in the journalized tables.

Template 3: Using OdiWaitForLogData (Consistent Set or Simple)

- Step 1: `OdiWaitForLogData`. If no new log data is detected after a specified interval, end the package.
- Step 2: Execute a scenario equivalent to the template 1 or 2, using `OdiStartScen`

This package is scheduled regularly. Changed data will only be processed if new changes have been detected. This avoids useless processing if changes occur sporadically to the journalized tables (i.e. to avoid running mappings that would process no data).

Template 4: Separate Processes (Consistent Set)

This template dissociates the consistency window, the purge, and the changes consumption (for two different subscribers) in different packages.

Package 1: Extend Window

- Step 1: OdiWaitForLogData. If no new log data is detected after a specified interval, end the package.
- Step 2: Extend Window.

This package is scheduled every minute. Extend Window may be resource consuming. It is better to have this operation triggered only when new data appears.

Package 2: Purge Journal (at the end of week)

Step 1: Purge Journal

This package is scheduled once every Friday. We will keep track of the journals for the entire week.

Package 3: Process the Changes for Subscriber A

- Step 1: Lock Subscriber A
- Step 2 to n-1: Mappings using the journalized data for subscriber A
- Step n: Unlock Subscriber A

This package is scheduled every minute. Such a package is used for instance to generate events in a MOM.

Package 4: Process the Changes for Subscriber B

- Step 1: Lock Subscriber B
- Step 2 to n-1: Mappings using the journalized data for subscriber B
- Step n: Unlock Subscriber B

This package is scheduled every day. Such a package is used for instance to load a data warehouse during the night with the changed data.

Creating Data Models with Common Format Designer

This chapter describes how to use Oracle Data Integrator's Common Format Designer feature for creating a data model by assembling elements from other models. It also details how to generate the DDL scripts for creating or updating a model's implementation in your data servers, and how to automatically generate the mappings to load data from and to a model.

This chapter includes the following sections:

- [Introduction to Common Format Designer](#)
- [Using the Diagram](#)
- [Generating DDL scripts](#)
- [Generating Mapping IN/OUT](#)

5.1 Introduction to Common Format Designer

Common Format Designer (CFD) is used to quickly design a data model in Oracle Data Integrator. This data model may be designed as an entirely new model or assembled using elements from other data models. CFD can automatically generate the Data Definition Language (DDL) scripts for implementing this model into a data server.

Users can for example use Common Format Designer to create operational datastores, datamarts, or master data canonical format by assembling heterogeneous sources.

CFD enables a user to modify an existing model and automatically generate the DDL scripts for synchronizing differences between a data model described in Oracle Data Integrator and its implementation in the data server.

5.1.1 What is a Diagram?

A diagram is a graphical view of a subset of the datastores contained in a sub-model (or data model). A data model may have several diagrams attached to it.

A diagram is built:

- by assembling datastores from models and sub-models.
- by creating blank datastores into which you either create new attributes or assemble attributes from other datastores.

5.1.2 Why assemble datastores and attributes from other models?

When assembling datastores and attributes from other models or sub-models in a diagram, Oracle Data Integrator keeps track of the origin of the datastore or attribute that is added to the diagram. These references to the original datastores and attributes enable Oracle Data Integrator to automatically generate the mappings to the assembled datastores (mappings IN)

Automatic mapping generation does not work to load datastores and attributes that are not created from other model's datastores and attributes. It is still possible to create the mappings manually, or complete generated mapping for the attributes not automatically mapped.

5.1.3 Graphical Synonyms

In a diagram, a datastore may appear several times as a **Graphical Synonym**. A synonym is a graphical representation of a datastore. Graphical synonyms are used to make the diagram more readable.

If you delete a datastore from a diagram, Designer prompts you to delete either the synonym (the datastore remains), or the datastore itself (all synonyms for this datastore are deleted).

References in the diagram are attached to a datastore's graphical synonym. It is possible create graphical synonyms at will, and move the references graphical representation to any graphical synonym of the datastores involved in the references.

5.2 Using the Diagram

From a diagram, you can edit all the model elements (datastore, attributes, references, filters, etc.) visible in this diagram, using their popup menu, directly available from the diagram. Changes performed in the diagram immediately apply to the model.

5.2.1 Creating a New Diagram

To create a new diagram:

1. In the **Models** tree in Designer Navigator, expand the data model or sub-model into which you want to create the diagram, then select the **Diagrams** node.
2. Right-click, then select **New Diagram** to open the Diagram Editor.
3. On the Definition tab of the Diagram Editor enter the diagram's **Name** and **Description**.
4. Select **Save** from the File main menu.

The new diagram appears under the **Diagrams** node of the model.

5.2.2 Create Datastores and Attributes

To insert an existing datastore in a diagram:

1. Open the Diagram Editor by double-clicking the diagram under the Diagrams node under the model's node.
2. In the Diagram Editor, select the **Diagram** tab.
3. Select the datastore from the **Models** tree in Designer Navigator.

4. Drag this datastore into the diagram. If the datastore comes from a model/sub-model different from the current model/sub-model, Designer will prompt you to create a copy of this datastore in the current model. If the datastore already exists in the diagram, Oracle Data Integrator will prompt you to either create new graphical synonym, or create a duplicate of the datastore.

The new graphical synonym for the datastore appears in the diagram. If you have added a datastore from another model, or chosen to create a duplicate, the new datastore appears in model.

If references (join) existed in the original models between tables inserted to the diagram, these references are also copied.

To create a graphical synonym of a datastore already in the diagram select **Create Graphical Synonym** in the popup menu of the datastore.

To create a new datastore in a diagram:

1. In the Diagram Editor, select the **Diagram** tab.
2. In the Diagram Editor toolbar, click **Add Datastore**.
3. Click into the diagram workbench.
4. An Editor appears for this new datastore. Follow the process described in [Chapter 3, "Creating and Using Data Models and Datastores,"](#) for creating your datastore.

To add attributes from another datastore:

1. In the Diagram Editor, select the **Diagram** tab.
2. Select a attribute under a datastore from the **Models** tree of the Designer Navigator.
3. Drag this attribute into the datastore in the diagram to which you want to append this attribute. The Attribute Editor appears to edit this new attribute. Edit the attribute according to your needs.
4. Select **Save** from the File main menu. The new attribute is added to the datastore.

5.2.3 Creating Graphical Synonyms

To create a graphical synonym for a datastore:

1. In the **Diagram** tab, select the datastore.
2. Right-click, then select **Create Graphical Synonym**.

The new graphical synonym appears in the diagram.

This operation does not add a new datastore. It creates only a new representation for the datastore in the diagram.

5.2.4 Creating and Editing Constraints and Filters

To add a new condition, filter, key to a datastore:

1. In the **Diagram** tab, select the datastore.
2. Right-click then select the appropriate option: **Add Key**, **Add Filter**, etc.
3. A new Editor appears for the new condition, filter, key, etc. Follow the process described in [Chapter 3, "Creating and Using Data Models and Datastores,"](#) for creating this element.

Conditions, filters and references are added to the diagram when you add the datastore which references them into the diagram. It is possible to drag into the diagram these objects if they have been added to the datastore after you have added it to the diagram.

To edit a key on a attribute:

If a attribute is part of a key (Primary, Alternate), it is possible to edit the key from this attribute in the diagram.

1. In the **Diagram** tab, select one of the attribute participating to the key.
2. Right-click then select the name of the key in the pop-up menu, then select **Edit** in the sub-menu.

To create a reference between two datastores:

1. In the Diagram Editor, select the **Diagram** tab.
2. In the toolbar click the **Add Reference** button.
3. Click the first datastore of the reference, then drag the cursor to the second datastore while keeping the mouse button pressed.
4. Release the mouse button. The Reference Editor appears.
5. Set this reference's parameters according to the process described in [Chapter 3, "Creating and Using Data Models and Datastores."](#)

To move a reference to another graphical synonym:

1. In the Diagram Editor, select the **Diagram** tab.
2. In the **Diagram** tab, select the reference you wish to modify.
3. Right-click and select **Display Options**.
4. Select the synonyms to be used as the parent and child of the reference.
5. Click **OK**. The reference representation appears now on the selected synonyms.

This operation does not change the reference itself. It only alters its representation in the diagram.

5.2.5 Printing a Diagram

Once you have saved your diagram you can save the diagram in PNG format, print it or generate a complete PDF report.

To print or generate a diagram report:

1. On the Diagram tab of your diagram, select **Print Options** from the Diagram menu.
2. In the Data Model Printing editor select according to your needs one of the following options:
 - Generate the complete PDF report
 - Save the diagram in PNG
 - Print your diagram
3. Click **OK**.

Note: If you do not specify a different location, The PDF and PNG files are saved to the default locations specified in the user preferences. To set these values, select the **Preferences...** option from the **Tools** menu. Expand the **ODI** node, and then the **System** node, and select **Reports**. Enter (or search for) the location of your **Default PDF generation directory** and **Directory for saving your diagrams (PNG)**.

5.3 Generating DDL scripts

When data structure changes have been performed in a data server, you usually perform an incremental reverse-engineering in Oracle Data Integrator to retrieve the new metadata from the data server.

When a diagram or data model is designed or modified in Oracle Data Integrator, it is necessary to implement the data model or the changes in the data server containing the model implementation. This operation is performed with DDL scripts. The DDL scripts are generated in the form of Oracle Data Integrator procedures containing DDL commands (create table, alter table, etc). This procedure may be executed on the data server to apply the changes.

The DDL generation supports two types of datastores: tables and system tables.

Note: The templates for the DDL scripts are defined as Action Groups. Check in the Topology Navigator that you have the appropriate action group for the technology of the model before starting DDL scripts generation. For more information on action groups, please refer to the *Knowledge Module Developer's Guide for Oracle Data Integrator*.

Before generating DDL Scripts

In certain cases, constraints that are defined in the data model but not in the database, are not displayed in the Generate DDL editor. To ensure that these conditions appear in the Generate DDL editor, perform the following tasks:

- For *Keys*: Select **Defined in the Database** and **Active** in the Control tab of the Key editor
- For *References*: Select **Defined in the Database** in the Definition tab of the Reference editor
- For *Conditions*: Select **Defined in the Database** and **Active** in the Control tab of the Condition editor

Generating DDL Scripts

To generate the DDL scripts:

1. In the **Models** tree of Designer Navigator, select the data model for which you want to generate the DDL scripts.
2. Right-click, then select **Generate DDL**. The Generate DDL for Oracle Data Integrator Model dialog is displayed.
3. In the Generate DDL dialog, click **Yes** if you want to process only tables that are in the Oracle Data Integrator model, otherwise click **No** and tables that are not in the model will be also included.

Oracle Data Integrator retrieves current state of the data structure from the data server, and compares it to the model definition. The progression is displayed in the status bar. The **Generate DDL** Editor appears, with the differences detected.

4. Select the **Action Group** to use for the DDL script generation.
5. Click **Search** to select the **Generation Folder** into which the procedure will be created.
6. Select the folder and click **OK**.
7. Filter the type of changes you want to display using the **Filters** check boxes.
8. Select the changes to apply by checking the **Synchronization** option. The following icons indicate the type of changes:
 - - : Element existing in the data model but not in the data server.
 - + : Element existing in the data server but not in the data model.
 - = : Element existing in both the data model and the data server, but with differences in its properties (example: a column resized) or attached elements (example: a table including new columns).
9. Click **OK** to generate the DDL script.

Oracle Data Integrator generates the DDL scripts in a procedure and opens the Procedure Editor for this procedure.

5.4 Generating Mapping IN/OUT

For a given model or datastore assembled using Common Format Designer, Oracle Data Integrator is able to generate:

- **Mappings IN:** These mappings are used to load the model's datastores assembled from other datastores/attributes. They are the integration process merging data from the original datastores into the composite datastores.
- **Mappings OUT:** These mappings are used to extract data from the model's datastores. They are generated using the mappings (including the mappings IN) already loading the model's datastore. They reverse the integration process to propagate the data from the composite datastore to the original datastores.

For example, an Active Integration Hub (AIH) assembles information coming from several other applications. It is made up of composite datastores built from several data models, assembled in a diagram. The AIH is loaded using the Mappings IN, and is able to send the data it contains to the original systems using the Mappings OUT.

To generate the Mappings IN:

1. In the **Models** tree of Designer Navigator, select the data model or datastore for which you want to generate the mappings.
2. Right-click, then select **Generate Mappings IN**. Oracle Data Integrator looks for the original datastores and attributes used to build the current model or datastore. The **Generate Mappings IN** Editor appears with a list of datastores for which Mappings IN may be generated.
3. Select an **Optimization Context** for your mappings. This context will define how the flow for the generated mappings will look like, and will condition the automated selection of KMs.
4. Click the **Search** button to select the **Generation Folder** into which the mappings will be generated.

5. In the **Candidate Datastores** table, check the **Generate Mapping** option for the datastores to load.
6. Edit the content of the **Mapping Name** column to rename the integration mappings.
7. Click **OK**. Mapping generation starts.

The generated mappings appear in the specified folder.

Note: Mappings automatically generated are built using predefined rules based on repository metadata. These mappings can not be executed immediately. They must be carefully reviewed and modified before execution

Note: If no candidate datastore is found when generating the Mappings IN, then it is likely that the datastores you are trying to load are not built from other datastores or attributes. Automatic mapping generation does not work to load datastores and attributes that are not created from other model's datastores and attributes.

To generate the Mapping OUT:

1. In the **Models** tree of Designer Navigator, select the data model or datastore for which you want to generate the mappings.
2. Right-click, then select **Generate Mapping OUT**. Oracle Data Integrator looks for the existing mappings loading these the datastores. The **Generate Mappings OUT** Editor appears with a list of datastores for which Mappings OUT may be generated.
3. Select an **Optimization Context** for your mappings. This context will define how the flow for the generated mappings will look like, and will condition the automated selection of KMs.
4. Click the **Search** button to select the **Generation Folder** into which the mappings will be generated.
5. In the **Candidate Datastores**, check the **Generation** and **Generate Mapping** check boxes to select either all or some of the candidate datastore to load from the target datastore of the existing mappings.
6. Edit the content of the **Mapping Name** column to rename the integration mappings.
7. Click **OK**. Mapping generation starts.

The generated mappings appear in the specified folder.

Note: Mappings automatically generated are built using the available metadata and do not always render the expected rules. These mappings must be carefully reviewed and modified before execution.

Note: If no candidate datastore is found when generating the Mappings OUT, then it is likely that no mapping loads the datastores you have selected to generate the mappings OUT. The mappings OUT from a datastore are generated from the mappings loading this datastore. Without any valid mapping loading a datastore, no propagation mapping from this datastore can be generated.

Part III

Developing Integration Projects

This part describes how to develop integration projects in Oracle Data Integrator.

This part contains the following chapters:

- [Chapter 6, "Creating an Integration Project"](#)
- [Chapter 7, "Creating and Using Packages"](#)
- [Chapter 8, "Creating and Using Mappings"](#)
- [Chapter 9, "Creating and Using Dimensions and Cubes"](#)
- [Chapter 10, "Using Compatibility Mode"](#)
- [Chapter 11, "Creating and Using Procedures, Variables, Sequences, and User Functions"](#)
- [Chapter 12, "Using Scenarios"](#)
- [Chapter 13, "Using Load Plans"](#)
- [Chapter 14, "Using Web Services"](#)
- [Chapter 15, "Using Shortcuts"](#)

Creating an Integration Project

This chapter describes the different components involved in an integration project, and explains how to start a project.

This chapter includes the following sections:

- [Introduction to Integration Projects](#)
- [Creating a New Project](#)
- [Managing Knowledge Modules](#)
- [Organizing the Project with Folders](#)

6.1 Introduction to Integration Projects

An integration project may be composed of several types of components. These components include organizational objects, such as folders, and development objects such as mappings and variables. "[Oracle Data Integrator Project Components](#)" details the different components involved in an integration project.

A project has also a defined life cycle which can be adapted to your practices. "[Project Life Cycle](#)" on page 6-3 suggests a typical project life cycle.

6.1.1 Oracle Data Integrator Project Components

Components involved in a project include components contained in the project and global components referenced by the project. In addition, a project also uses components defined in the models and topology.

6.1.1.1 Oracle Data Integrator Project Components

The following components are stored into a project. They appear in the in the **Project** accordion in the Designer Navigator, under the project's node.

Folder

Folders are components that help organizing the work into a project. Folders contain packages, mappings, procedures, and subfolders.

Packages

A package is a workflow, made up of a sequence of steps organized into an execution diagram. Packages assemble and reference other components from a project such as mappings, procedure or variable. See [Chapter 7, "Creating and Using Packages,"](#) for more information on packages.

Mappings

A mapping is a reusable dataflow. It is a set of declarative rules that describes the loading of one or several target datastores from one or more source datastores. See [Chapter 8, "Creating and Using Mappings,"](#) for more information on mappings and reusable mappings.

Procedure

A Procedure is a reusable component that groups a sequence of operations that do not fit in the mapping concept.

Examples of procedures:

- Wait and unzip a file
- Send a batch of files via FTP
- Receive emails
- Purge a database

Variable

A variable's value is stored in Oracle Data Integrator. This value may change during the execution.

Sequence

A sequence is a variable automatically incremented when used. Between two uses the value is persistent.

User Functions

User functions allow you to define customized functions or "function aliases," for which you will define technology-dependent implementations. They are usable in mappings and procedures.

See [Chapter 11, "Creating and Using Procedures, Variables, Sequences, and User Functions,"](#) for more information about the components described above.

Knowledge Modules

Oracle Data Integrator uses Knowledge Modules at several points of a project design. A Knowledge Module is a code template related to a given technology that provides a specific function (loading data, reverse-engineering, journalizing).

Marker

A component of a project may be flagged in order to reflect a methodology or organization. Flags are defined using markers. These markers are organized into groups, and can be applied to most objects in a project. See [Chapter 18, "Organizing and Documenting Integration Projects,"](#) for more information on markers.

Scenario

When a package, mapping, procedure, or variable component has been fully developed, it is compiled in a scenario. A scenario is the execution unit for production. Scenarios can be scheduled for automated execution. See [Chapter 12, "Using Scenarios,"](#) for more information on scenarios.

6.1.1.2 Global Components

Global components are similar to project objects. The main difference is their scope. They have a global scope and can be used in any project. Global objects include Variables, Knowledge Modules, Sequences, Markers, Reusable Mappings, and User Functions.

6.1.2 Project Life Cycle

The project life cycle depends on the methods and organization of your development team. The following steps must be considered as guidelines for creating, working with and maintaining an integration project.

1. Create a new project and import Knowledge Modules for this project.
2. Define the project organization and practices using folders, markers and documentation.
3. Create reusable components: mappings, procedures, variables, sequences. Perform unitary tests.
4. Assemble these components into packages. Perform integration tests.
5. Release the work in scenarios.
6. Optionally, organize scenarios into Load Plans. See [Chapter 13, "Using Load Plans."](#)

6.2 Creating a New Project

To create a project:

1. In Designer Navigator, click **New Project** in the toolbar of the **Projects** accordion.
2. Enter the **Name** of the project.
3. Keep or change the automatically-generated project code. Because this code is used to identify objects within this project, oracle recommends using a compact string. For example, if the project is called *Corporate Datawarehouse*, a compact code could be *CORP_DWH*.
4. From the **File** menu, click **Save**.

The new project appears in the Projects tree with one empty folder.

6.3 Managing Knowledge Modules

Knowledge Modules (KMs) are components of Oracle Data Integrator's integration technology. KMs contain the knowledge required by ODI to perform a specific set of tasks against a specific technology or set of technologies.

Oracle Data Integrator uses six different types of Knowledge Modules:

- RKM (Reverse Knowledge Modules) are used to perform a customized reverse-engineering of data models for a specific technology. These KMs are used in data models. See [Chapter 3, "Creating and Using Data Models and Datastores."](#)
- LKM (Loading Knowledge Modules) are used to extract data from source systems (files, middleware, database, etc.). These KMs are used in mappings. See [Chapter 8, "Creating and Using Mappings."](#)
- JKM (Journalizing Knowledge Modules) are used to create a journal of data modifications (insert, update and delete) of the source databases to keep track of

the changes. These KMs are used in data models and used for Changed Data Capture. See [Chapter 4, "Using Journalizing."](#)

- IKM (Integration Knowledge Modules) are used to integrate (load) data to the target tables. These KMs are used in mappings. See [Chapter 8, "Creating and Using Mappings."](#)
- CKM (Check Knowledge Modules) are used to check that constraints on the sources and targets are not violated. These KMs are used in data models' static check and mappings' flow checks. See [Chapter 3, "Creating and Using Data Models and Datastores,"](#) and [Chapter 8, "Creating and Using Mappings."](#)
- SKM (Service Knowledge Modules) are used to generate the code required for creating data services. These KMs are used in data models. See "Generating and Deploying Data Services" in *Administering Oracle Data Integrator*.

6.3.1 Project and Global Knowledge Modules

Knowledge Modules can be created and used as *Project Knowledge Modules* or *Global Knowledge Modules*. Global Knowledge Modules can be used in all projects, while Project Knowledge Modules can only be used within the project into which they have been imported.

Global KMs are listed in Designer Navigator in the **Global Objects** accordion, while Project KMs appear under the project into which they have been imported. See ["Importing Objects"](#) on page 23-11 for more information on how to import a Knowledge Module.

ODI also provides Built-In KMs that are always present and don't need to be imported. All Built-In KMs are of type LKM or IKM and cover the technologies Oracle, File, and Generic. For more information about Built-In KMs see the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator*.

When using global KMs, note the following:

- Global KMs should only reference global objects. Project objects are not allowed.
- You can only use global markers to tag a global KM.
- It is not possible to transform a project KM into a global KM and vice versa.
- If a global KM is modified, the changes will be seen by any ODI object using the Knowledge Module.
- Be careful when deleting a global KM. A missing KM causes execution errors.
- To distinguish global from project KMs, the prefix GLOBAL is used for the name of global KMs if they are listed with project KMs.
- The order in which the global and project KMs are displayed changes depending on the context:
 - The KM Selector lists in the Mapping Editor displays first the project KMs, then the global KMs. The GLOBAL or PROJECT_CODE prefix is used.
 - The KM Selector lists in the Model editor displays first the global KMs, then the project KMs. The GLOBAL or PROJECT_CODE prefix is used.

6.3.2 Knowledge Module Naming Conventions

Oracle Data Integrator's KMs are named according to a convention that facilitates the choice of the KM. This naming convention is as follows:

Loading Knowledge Modules

They are named with the following convention: *LKM <source technology> to <target technology> [(loading method)]*.

In this convention the source and target technologies are the source and target of the data movement this LKM can manage. When the technology is *SQL*, then the technology can be any technology supporting JDBC and SQL. When the technology is *JMS*, the technology can be any technology supporting JMS connectivity.

The *loading method* is the technical method used for moving the data. This method is specific to the technology involved. When no method is specified, the technical method used is a standard Java connectivity (JDBC, JMS and such) and data is loaded via the run-time agent. Using a KM that uses a loading method specific to the source and/or target technology usually brings better performances.

Examples of LKMs are given below:

- *LKM Oracle to Oracle (DBLink)* loads data from an Oracle data server to another Oracle data server using the Oracle DBLink.
- *LKM File to Oracle (SQLLDR)* loads data from a file into an Oracle data server using SQLLoader.
- *LKM SQL to SQL (Built-In)* loads data from a data server supporting SQL into another one. This is the most generic loading Knowledge Module, which works for most data servers.

Integration Knowledge Modules

They are named with the following convention: *IKM [<staging technology>] to <target technology> [<integration mode>] [(integration method)]*.

In this convention, the *target technology* is the technology of the target into which data will be integrated. IKMs may have a *staging technology* when the target is not located on the same server as the staging area. These KMs are referred to as *Multi-technology IKMs*. They are used when the target cannot be used as the staging area. For example, with the *File* technology.

The *integration mode* is the mode used for integrating record from the data flow into the target. Common modes are:

- **Append:** Insert records from the flow into the target. It is possible to optionally delete all records from the target before the insert. Existing records are not updated.
- **Control Append:** Same as above, but in addition the data flow is checked in the process.
- **Incremental Update:** Same as above. In addition, it is possible to update existing records with data from the flow.
- **Slowly Changing Dimension:** Integrate data into a table using Type 2 slowly changing dimensions (SCD).

The *integration method* is the technical method used for integrating the data into the target. This method is specific to the technologies involved. When no method is specified, the technical method used is a standard Java connectivity (JDBC, JMS and such) and SQL language. Using a KM that uses an integration method specific to a given technology usually brings better performance.

Examples of IKMs are given below:

- *IKM Oracle Merge* integrates data from an Oracle staging area into an Oracle target located in the same data server using the incremental update mode. This KM uses the Oracle Merge Table feature.
- *IKM SQL to File Append* integrates data from a SQL-enabled staging area into a file. It uses the append mode.
- *IKM SQL Incremental Update* integrates data from a SQL-enabled staging area into a target located in the same data server. This IKM is suitable for all cases when the staging area is located on the same data server as the target, and works with most technologies.
- *IKM SQL to SQL Append* integrates data from a SQL-enabled staging area into a target located in a different SQL-enabled data server. This IKM is suitable for cases when the staging area is located on a different server than the target, and works with most technologies.

Check Knowledge Modules

They are named with the following convention: *CKM <staging technology>*.

In this convention, the *staging technology* is the technology of the staging area into which data will be checked.

Examples of CKMs are given below:

- *CKM SQL* checks the quality of an integration flow when the staging area is in a SQL-enabled data server. This is a very generic check Knowledge Module that works with most technologies.
- *CKM Oracle* checks the quality of an integration flow when the staging area is in an Oracle data server.

Reverse-engineering Knowledge Modules

They are named with the following convention: *RKM <reversed technology> [(reverse method)]*.

In this convention, the *reversed technology* is the technology of the data model that is reverse-engineered. The reverse method is the technical method used for performing the reverse-engineering process.

Examples of RKMs are given below:

- *RKM Oracle* reverse-engineers an Oracle data model
- *RKM Netezza* reverse-engineers a Netezza data model

Journalizing Knowledge Modules

They are named with the following convention: *JKM <journalized technology> <journalizing mode> (<journalizing method>)*.

In this convention, the *journalized technology* is the technology into which changed data capture is activated. The *journalizing mode* is either **Consistent** or **Simple**. For more information about these modes, see [Chapter 4, "Using Journalizing."](#)

The journalizing method is the technical method for capturing the changes. When not specified, the method used for performing the capture process is triggers.

Examples of JKMs are given below:

- *JKM Oracle to Oracle Consistent (OGG Online)* creates the infrastructure for consistent set journalizing on an Oracle staging server and generates the Oracle

GoldenGate configuration for replicating data from an Oracle source to this staging server.

- *JKM Oracle Simple* enables CDC for Oracle in simple mode using triggers.
- *JKM MSSQL Simple* Creates the journalizing infrastructure for simple journalizing on Microsoft SQL Server tables using triggers.

Service Knowledge Modules

They are named with the following convention: *SKM <data server technology>*.

In this convention, the *data server technology* is the technology into which the data to be accessed with web services is stored.

6.3.3 Choosing the Right Knowledge Modules

Oracle Data Integrator provides a large range of Knowledge Modules out of the box. When starting an integration project, you can start with the built-in KMs introduced in ODI 12c, and import additional Knowledge Modules as needed for your project.

It is possible to import additional KMs after setting up the project, and it is possible to change the KMs used afterwards. The following guidelines can be used for choosing the right KMs when starting a new project:

- Start with Generic KMs. The SQL KMs work with almost all technologies. If you are not comfortable with the source/target technologies you are working with, you can start by using the generic SQL KMs, as they use standard SQL. A simple project can start with the following generic KMs: *LKM File to SQL*, *LKM SQL to SQL (Built-In)*, *IKM SQL to SQL Append*, *IKM SQL Insert*, *CKM SQL*.
- Start with simple KMs. If you are not comfortable with the technologies you are integrating, do not start using the KMs using complex integration methods or modes.
- Select KMs that match your source/target combinations to increase performance. The more specific the KM to a technology combination, the better the performance. For achieving the best performances, make sure to switch to KMs that match the source/target combination you have, and that leverage the features from these sources/targets.
- Select KMs according to your infrastructure limitations. If it is not possible to use the target data servers as the staging area for security reasons, make sure to have multi-technology IKMs available in your project.
- Select JKMs and SKMs only if you need them. Do not import JKMs or SKMs if you do not plan to use Changed Data Capture or Data Services. You can import them later when needed.
- Review the KM documentation and options. KMs include a Description field that contain useful information. Each of the KM options is also described. All KMs are detailed in the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator*.

6.3.4 Importing and Replacing Knowledge Modules

Two main operations allow you to manage KMs into a project:

- When you create a new project you can use the Built-In KMs. If you want to use new KMs, you must import either a project KM or a global KM. See "[Project and Global Knowledge Modules](#)" on page 6-4 for more information on the knowledge module's scope.

- If you want to start using a new version of an existing global or project KM, or if you want to replace an existing KM in use with another one, then you can replace this KM.

This section includes the following topics:

- [Importing a Project Knowledge Module](#)
- [Replacing a Knowledge Module](#)
- [Importing a Global Knowledge Module](#)

Importing a Project Knowledge Module

To import a Project Knowledge Module into a project:

1. In the **Projects** accordion in Designer Navigator, select the project into which you want to import the KM.
2. Right-click and select **Import > Import Knowledge Modules...**
3. Specify the **File Import Directory**. A list of the KMs export files available in this directory appears. KMs included in the ODI installation are located in:


```
<Oracle_Home>/odi/sdk/xml-reference
```
4. Select several KMs from the list and then click **OK**.
5. Oracle Data Integrator imports the selected KMs and presents an import report.
6. Click **Close** to close this report.

The Knowledge Modules are imported into you project. They are arranged under the Knowledge Modules node of the project, grouped per KM type.

Note: Knowledge modules can be imported in Duplication mode only. To replace an existing Knowledge Modules, use the import replace method described below. When importing a KM in Duplication mode and if the KM already exists in the project, ODI creates a new KM with prefix `copy_of`.

Replacing a Knowledge Module

When you want to replace a global KM or a KM in a project by another one and have all mappings automatically use the new KM, you must use the Import Replace mode. See "[Import Modes](#)" on page 23-3 for more information.

To import a Knowledge Module in replace mode:

1. In Designer Navigator, select the Knowledge Module you wish to replace.
2. Right-click and select **Import Replace**.
3. In the Replace Object dialog, select the export file of the KM you want to use as a replacement. KMs included in the ODI installation are located in:

```
<Oracle_Home>/odi/sdk/xml-reference
```

4. Click **OK**.

The Knowledge Module is now replaced by the new one.

Note: When replacing a Knowledge module by another one, Oracle Data Integrator sets the options in mappings for the new module using the option name similarities with the old module's options. When a KM option was set by the user in a mapping, this value is preserved if the new KM has an option with the same name. New options are set to the default value. It is advised to check the values of these options in the mappings.

Replacing a KM by another one may lead to issues if the KMs have different structure or behavior, for example when you replace a IKM with a RKM. It is advised to check the mappings' design and execution with the new KM.

Importing a Global Knowledge Module

To import a global knowledge module in Oracle Data Integrator:

1. In the Navigator, select the Global Knowledge Modules node in the Global Objects accordion.
2. Right-click and select **Import Knowledge Modules**.
3. In the Import dialog:
 1. Select the **Import Type**. See "[Import Modes](#)" on page 23-3 for more information.
 2. Specify the **File Import Directory**. A list of the KMs export files available in this directory appears. KMs included in the ODI installation are located in:


```
<Oracle_Home>/odi/sdk/xml-reference
```
 3. Select the file(s) to import from the list.
4. Click **OK**.

The global KM is now available in all your projects.

6.3.5 Encrypting and Decrypting a Knowledge Module

Encrypting a Knowledge Module (KM) or Procedure allows you to protect valuable code. An encrypted KM or procedure cannot be read or modified if it is not decrypted. The commands generated in the log by an Encrypted KM or procedure are also unreadable.

Oracle Data Integrator uses a AES Encryption algorithm based on a personal encryption key. This key can be saved in a file and can be reused to perform encryption or decryption operations.

WARNING: There is no way to decrypt an encrypted KM or procedure without the encryption key. Oracle therefore strongly advises keeping this key in a safe location. Oracle also recommends using a unique key for each deployment.

To Encrypt a KM or a Procedure:

1. In the **Projects** tree in Designer Navigator, expand the project, and select the KM or procedure you want to encrypt.
2. Right-click and select **Encrypt**.

3. In the Encryption Options window, you can either:
 - **Encrypt with a personal key** that already exists by giving the location of the personal key file or by typing in the value of the personal key.
 - **Get a new encryption key** to have a new key generated by ODI.
4. Click **OK** to encrypt the KM or procedure. If you have chosen to generate a new key, a window will appear with the new key. You can save the key in a file from here.

To decrypt a KM or a procedure:

1. In the **Projects** tree in Designer Navigator, expand the project, and select the KM or procedure you want to decrypt.
2. Right-click and select **Decrypt**.
3. In the **KM Decryption** or **Procedure Decryption** window, either
 - Select an existing encryption key file;
 - or type in (or paste) the string corresponding to your personal key.
4. Click **OK** to decrypt.

6.4 Organizing the Project with Folders

In a project, mappings, procedures, and packages are organized into folders and sub-folders. Oracle recommends maintaining a good organization of the project by using folders. Folders simplify finding objects developed in the project and facilitate the maintenance tasks. Organization is detailed in [Chapter 18, "Organizing and Documenting Integration Projects."](#)

Creating and Using Packages

This chapter gives an introduction to Packages and Steps. It also passes through the creating process of a Package and provides additional information about handling steps within a Package.

This chapter includes the following sections:

- [Introduction to Packages](#)
- [Creating a new Package](#)
- [Working with Steps](#)
- [Defining the Sequence of Steps](#)
- [Running a Package](#)

7.1 Introduction to Packages

The Package is a large unit of execution in Oracle Data Integrator. A Package is made up of a sequence of steps organized into an execution diagram.

Each step can either succeed or fail its execution. Depending on the execution result (success or failure), a step can branch to another step.

7.1.1 Introduction to Steps

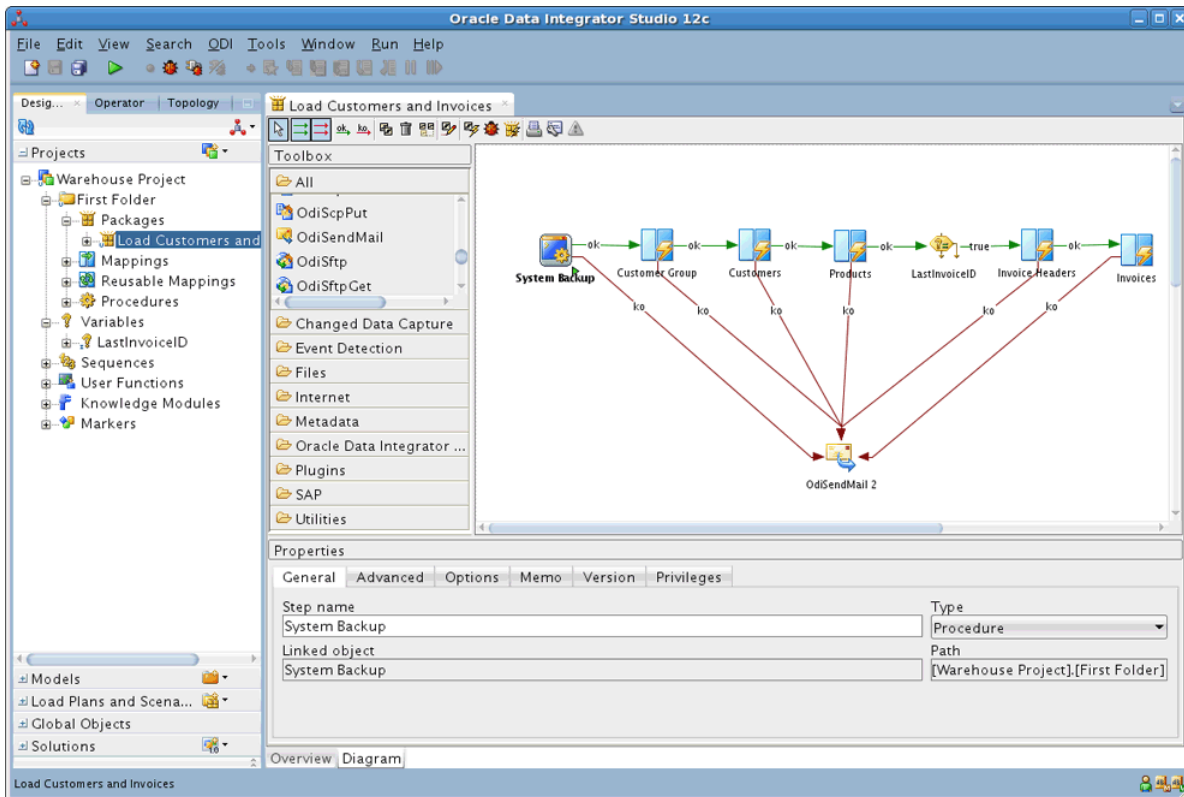
[Table 7-1](#) lists the different types of steps. References are made to sections that provide additional details

Table 7-1 Step Types

Type	Description	See Section
Flow (Mapping)	Executes a Mapping.	"Adding a Mapping step" on page 7-4
Procedure	Executes a Procedure.	"Adding a Procedure step" on page 7-5
Variable	Declares, sets, refreshes or evaluates the value of a variable.	"Variable Steps" on page 7-5
Oracle Data Integrator Tools	These tools, available in the Toolbox, provide access to all Oracle Data Integrator API commands, or perform operating system calls.	"Adding Oracle Data Integrator Tool Steps" on page 7-6

Table 7-1 (Cont.) Step Types

Type	Description	See Section
Models, Sub-models, and Datastores	Performs journalizing, static check or reverse-engineering operations on these objects	"Adding a Model, Sub-Model or Datastore" on page 7-7

Figure 7-1 Sample Package

For example, the "Load Customers and Invoice" Package example shown in [Figure 7-1](#) performs the following actions:

1. Execute procedure "System Backup" that runs some backup operations.
2. Execute mapping "Customer Group" that loads the customer group datastore.
3. Execute mapping "Customer" that loads the customer datastore.
4. Execute mapping "Product" that loads the product datastore.
5. Refresh variable "Last Invoice ID" step to set the value of this variable for use later in the Package.
6. Execute mapping "Invoice Headers" that load the invoice header datastore.
7. Execute mapping "Invoices" that load the invoices datastore.
8. If any of the steps above fails, then the Package runs the "OdiSendMail 2" step that sends an email to the administrator using an Oracle Data Integrator tool.

7.1.2 Introduction to Creating Packages

Packages are created in the Package Diagram Editor. See ["Introduction to the Package editor"](#) on page 7-3 for more information.

Creating a Package consists of the following main steps:

1. Creating a New Package. See ["Creating a new Package"](#) on page 7-4 for more information.
2. Working with Steps in the Package (add, duplicate, delete, and so on). See ["Working with Steps"](#) on page 7-4 for more information.
3. Defining Step Sequences. See ["Defining the Sequence of Steps"](#) on page 7-9 for more information.
4. Running the Package. See ["Running a Package"](#) on page 7-11 for more information.

7.1.3 Introduction to the Package editor

The Package editor provides a single environment for designing Packages. [Figure 7–2](#) gives an overview of the Package editor.

Figure 7–2 Package editor

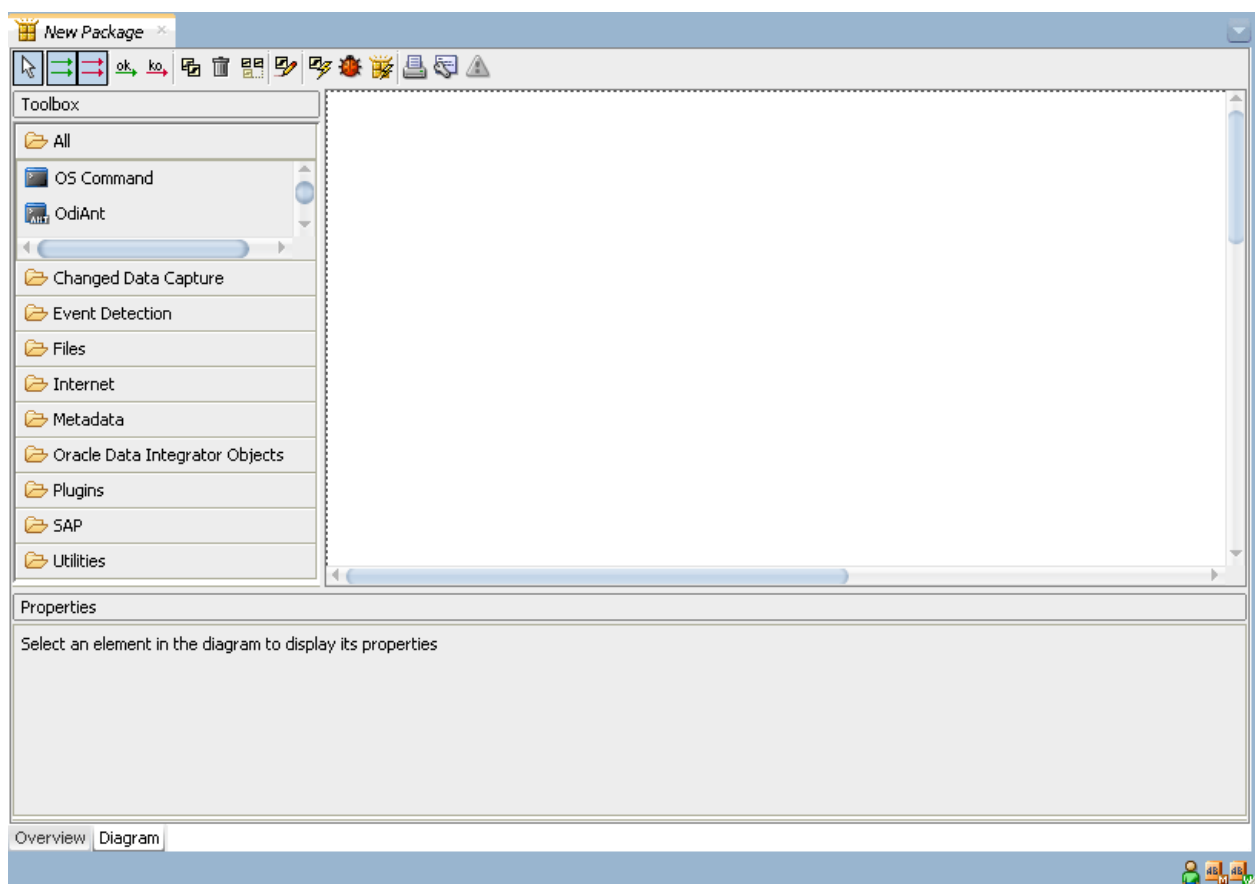


Table 7–2 Package editor Sections

Section	Location in Figure	Description
Package Diagram	Middle	You drag components such as mappings, procedures, datastores, models, sub-models or variables from the Designer Navigator into the Package Diagram for creating steps for these components. You can also define sequence of steps and organize steps in this diagram.
Package Toolbox	Left side of the Package diagram	The Toolbox shows the list of Oracle Data Integrator tools available and that can be added to a Package. These tools are grouped by type.
Package Toolbar	Top of the Package diagram	The Package Toolbar provides tools for organizing and sequencing the steps in the Package.
Properties Panel	Under the Package diagram	This panel displays the properties for the object that is selected in the Package Diagram.

7.2 Creating a new Package

To create a new Package:

1. In the **Project** tree in Designer Navigator, click the **Packages** node in the folder where you want to create the Package.
2. Right-click and select **New Package**.
3. In the New Package dialog, type in the **Name**, and optionally a **Description**, of the Package. Click **OK**.
4. Use the **Overview** tab to set properties for the package.
5. Use the **Diagram** tab to design your package, adding steps as described in "[Working with Steps](#)" on page 7-4.
6. From the **File** menu, click **Save**.

7.3 Working with Steps

Packages are an organized sequence of steps. Designing a Package consists mainly in working with the steps of this Package.

7.3.1 Adding a Step

Adding a step depends on the nature of the steps being inserted. See [Table 7–1, "Step Types"](#) for more information on the different types of steps. The procedures for adding the different type of steps are given below.

7.3.1.1 Adding a Mapping step

To insert a Mapping step:

1. Open the Package editor and go to the **Diagram** tab.
2. In the Designer Navigator, expand the project node and then expand the Mappings node, to show your mappings for this project.
3. Drag and drop a mapping into the diagram. A Flow (Mapping) step icon appears in the diagram.

4. Click the step icon in the diagram. The properties panel shows the mapping's properties.
5. In the properties panel, modify properties of the mapping as needed.
6. From the **File** menu, click **Save**.

7.3.1.2 Adding a Procedure step

To insert a Procedure step:

1. Open the Package editor and go to the **Diagram** tab.
2. In the Designer Navigator, expand the project node and then expand the Procedures node, to show your procedures for this project.
3. Drag and drop a procedure into the diagram. A Procedure step icon appears in the diagram.
4. Click the step icon in the diagram. The properties panel shows the procedure's properties.
5. In the properties panel, modify properties of the procedure as needed.
6. From the **File** menu, click **Save**.

7.3.1.3 Variable Steps

There are different variable step types within Oracle Data Integrator:

- **Declare Variable:** When a variable is used in a Package (or in elements of the topology which are used in the Package), Oracle strongly recommends that you insert a Declare Variable step in the Package. This step explicitly declares the variable in the Package.
- **Refresh Variable:** This variable step refreshes the variable by running the query specified in the variable definition.
- **Set Variable:** There are two functions for this step:
 - **Assign** sets the current value of a variable.
 - **Increment** increases or decreases a numeric value by the specified amount.
- **Evaluate Variable:** This variable step type compares the value of the variable with a given value according to an operator. If the condition is met, then the evaluation step is true, otherwise it is false. This step allows for branching in Packages.

Adding a Variable step

To add a Variable step (of any type):

1. Open the Package editor and go to the **Diagram** tab.
2. In the Designer Navigator, expand the project node and then expand the Variables node, to show your variables for this project. Alternatively, expand the Global Objects node and expand the Variables node, to show global variables.
3. Drag and drop a variable into the diagram. A Variable step icon appears in the diagram.
4. Click the step icon in the diagram. The properties panel shows the variable's properties.
5. In the properties panel, modify properties of the variable as needed. On the **General** tab, select the variable type from the **Type** list.

- For Set Variables, select **Assign**, or **Increment** if the variable is of **Numeric** type. For **Assign**, type into the **Value** field the value to be assigned to the variable (this value may be another variable). For **Increment**, type into the **Increment** field a numeric constant by which to increment the variable.
- For Evaluate Variables, select the **Operator** used to compare the variable value. Type in the **Value** field the value to compare with your variable. This value may be another variable.

Notes:

- You can specify a list of values in the Value field. When using the **IN** operator, use the semicolon character (;) to separate the values of a list.
 - An evaluate variable step can be branched based on the evaluation result. See "[Defining the Sequence of Steps](#)" on page 7-9 for more information on branching steps.
-
-

6. From the **File** menu, click **Save**.

7.3.1.4 Adding Oracle Data Integrator Tool Steps

Oracle Data Integrator provides tools that can be used within Packages for performing simple operations. The tools are either built-in tools or Open Tools that enable you to enrich the data integrator toolbox.

To insert an Oracle Data Integrator Tool step:

1. Open the Package editor and go to the **Diagram** tab.
2. From the Package **Toolbox**, select the tool that you want to use. Note that Open tools appear in the **Plugins** group.
3. Click in the Package diagram. A step corresponding to your tool appears.

Tip: As long as a tool is selected, left-clicking in the diagram will continue to place steps. To stop placing steps, click the **Free Choice** button in the Package Toolbar. The mouse pointer changes to an arrow, indicating you are no longer placing tools.

4. Click the step icon in the diagram. The properties panel shows the tool's properties.
5. Set the values for the parameters of the tool. The parameters descriptions appear when you select one, and are detailed in *Oracle Data Integrator Tool Reference*
6. You can edit the code of this tool call in the **Command** tab.
7. From the **File** menu, click **Save**.

The following tools are frequently used in Oracle Data Integrator Package:

- **OdiStartScen:** starts an Oracle Data Integrator scenario synchronously or asynchronously. To create an OdiStartScen step, you can directly drag and drop the scenario from the Designer Navigator into the diagram.
- **OdiInvokeWebService:** invokes a web service and saves the response in an XML file. OdiInvokeWebService uses the HTTP Analyzer tool to set up and test the tool parameters. For more information, see "[Using HTTP Analyzer](#)" on page 14-3.

- **OS Command:** calls an Operating System command. Using an operating system command may make your Package platform-dependent.

The Oracle Data Integrator tools are listed in *Oracle Data Integrator Tool Reference*

Note: When setting the parameters of a tool using the steps properties panel, graphical helpers allow value selection in a user-friendly manner. For example, if a parameter requires a project *identifier*, the graphical mapping will redesign it and display a list of project *names* for selection. By switching to the **Command** tab, you can review the raw command and see the identifier.

7.3.1.5 Adding a Model, Sub-Model or Datastore

You can perform journalizing, static check or reverse-engineering operations on models, sub-models, and datastores.

To insert a check, reverse engineer, or journalizing step in a Package:

Notes:

- To perform a static check, you must define the CKM in the model.
 - To perform journalizing operations, you must define the JKM in the model.
 - Reverse engineering options set in the model definition are used for performing reverse-engineering processes in a package.
-

1. Open the Package editor and go to the **Diagram** tab.
2. In Designer Navigator, select the model, sub-model or datastore to check from the **Models** tree.
3. Drag and drop this model, sub-model or datastore into the diagram.
4. In the **General** tab of the properties panel, select the Type: Check, Reverse Engineer, or Journalizing.
 - For Check steps, select **Delete Errors from the Checked Tables** if you want this static check to remove erroneous rows from the tables checked in this process.
 - For Journalizing steps, set the journalizing options. See [Chapter 4, "Using Journalizing,"](#) for more information on these options.
5. From the **File** menu, click **Save**.

7.3.2 Deleting a Step

Caution: It is not possible to undo a delete operation in the Package diagram.

To delete a step:

1. In the Package toolbar tab, select the **Free Choice** tool.
2. Select the step to delete in the diagram.

3. Right-click and then select **Delete Step**. Or, hit the Delete key on your keyboard.
4. Click **Yes** to continue.

The step disappears from the diagram.

7.3.3 Duplicating a Step

To duplicate a step:

1. In the Package toolbar tab, select the **Free Choice** tool.
2. Select the step to duplicate in the diagram.
3. Right-click and then select **Duplicate Step**.

A copy of the step appears in the diagram.

7.3.4 Running a Step

To run a step:

1. In the Package toolbar tab, select the **Free Choice** tool.
2. Select the step to run in the diagram.
3. Right-click and then select **Execute Step**.
4. In the **Run** dialog, select the execution parameters:
 - Select the **Context** into which the step must be executed.
 - Select the **Logical Agent** that will run the step.
 - Select a **Log Level**.
 - Optionally, select **Simulation**. This option performs a simulation of the run operation and generates a run report, without actually affecting data.
5. Click **OK**.
6. The **Session Started Window** appears.
7. Click **OK**.

You can review the step execution in the Operator Navigator.

7.3.5 Editing a Step's Linked Object

The step's linked object is the mapping, procedure, variable, or other object from which the step is created. You can edit this object from the Package diagram.

To edit a step's linked object:

1. In the Package toolbar tab, select the **Free Choice** tool.
2. Select the step to edit in the diagram.
3. Right-click and then select **Edit Linked Object**.

The Editor for the linked object opens.

7.3.6 Arranging the Steps Layout

The steps can be rearranged automatically or manually in the diagram in order to make it more readable.

You can use the **Reorganize** button from the toolbar to automatically reorganize all of the steps in your package.

To manually arrange the steps in the diagram:

1. From the Package toolbar menu, select the **Free Choice** tool.
2. Select the steps you wish to arrange using either of the following methods:
 - Keep the CTRL key pressed and select each step.
 - Drag a box around multiple items in the diagram with the left mouse button pressed.
3. To arrange the selected steps, you may either:
 - Drag them to arrange their position into the diagram
 - Right-click, then select a **Vertical Alignment** or **Horizontal Alignment** option from the context menu.

7.4 Defining the Sequence of Steps

Once the steps are created, you must order them into a data processing chain. This chain has the following rules:

- It starts with a unique step defined as the First Step.
- Each step has two termination states: Success or Failure.
- A step in failure or success can be followed by another step, or by the end of the Package.
- In case of failure, it is possible to define a number of retries.

A Package has one entry point, the First Step, but several possible termination steps.

Failure Conditions

The table below details the conditions that lead a step to a Failure state. In other situations, the steps ends in a Success state.

Note: By default, open transactions are not rolled back in a failure state. You can change this behavior using the Physical Agent property "Rollback all open transactions on step failure". Refer to the ODI Studio Online Help for details.

Step Type	Failure conditions
Flow	<ul style="list-style-type: none"> ■ Error in a mapping command. ■ Maximum number or percentage of errors allowed reached.
Procedure	Error in a procedure command.
Refresh Variable	Error while running the refresh query.
Set Variable	Error when setting the variable (invalid value).
Evaluate Variable	The condition defined in the step is not matched.
Declare Variable	This step has no failure condition and always succeeds.

Step Type	Failure conditions
Oracle Data Integrator Tool	Oracle Data Integrator Tool return code is different from zero. If this tool is an OS Command, a failure case is a command return code different from zero.
Journalize Datastore, Model or Sub-Model	Error in a journalizing command.
Check Datastore, Model or Sub-Model	Error in the check process.
Reverse Model	Error in the reverse-engineering process.

Defining the Sequence

To define the first step of the Package:

1. In the Package toolbar tab, select the **Free Choice** tool.
2. Select the step to set as the first one in the diagram.
3. Right-click and then select **First Step**.

The first step symbol appears on the step's icon.

To define the next step upon success:

1. In the Package toolbar tab, select the **Next Step on Success** tool.
2. Drag a line from one step to another, using the mouse.
3. Repeat this operation to link all your steps in a success path sequence. This sequence should start from the step defined as the First Step.

Green arrows representing the success path are shown between the steps, with an *ok* labels on these arrows. In the case of an evaluate variable step, the label is *true*.

To define the next step upon failure:

1. In the Package toolbar tab, select the **Next Step on Failure** tool.
2. Drag a line from one step to another, using the mouse.
3. Repeat this operation to link steps according to your workflow logic.

Red arrows representing the failure path are shown between the steps, with a *ko* labels on these arrows. In the case of an evaluate variable step, the arrow is green and the label is *false*.

To define the end of the Package upon failure:

By default, a step that is linked to no other step after a success or failure condition will terminate the Package when this success or failure condition is met. You can set this behavior by editing the step's behavior.

1. In the Package toolbar tab, select the **Free Choice** tool.
2. Select the step to edit.
3. In the properties panel, select the **Advanced** tab.
4. Select **End in Processing after failure** or **Processing after success**. The links after the step disappear from the diagram.
5. You can optionally set a **Number of attempts** and a **Time between attempts** for the step to retry a number of times with an interval between the retries.

7.5 Running a Package

To run a Package:

1. Use any of the following methods:
 - In the **Projects** node of the Designer Navigator, expand a project and select the Package you want to execute. Right-click and select **Run**, or click the **Run** button in the ODI Studio toolbar, or select **Run** from the **Run** menu of the ODI menu bar.
 - In the package editor, select the package by clicking the tab with the package name at the top of the editor. Click the **Run** button in the ODI Studio toolbar, or select **Run** from the **Run** menu of the ODI menu bar.
2. In the **Run** dialog, select the execution parameters:
 - Select the **Context** into which the package must be executed.
 - Select the **Logical Agent** that will run the package.
 - Select a **Log Level**.
 - Optionally, select **Simulation**. This option performs a simulation of the run operation and generates a run report, without actually affecting data.
3. Click **OK**.
4. The **Session Started Window** appears.
5. Click **OK**.

You can review the Package execution in the Operator Navigator.

Creating and Using Mappings

This chapter describes how to create and use mappings.

This chapter includes the following sections:

- [Introduction to Mappings](#)
- [Creating a Mapping](#)
- [Using Mapping Components](#)
- [Creating a Mapping Using a Dataset](#)
- [Physical Design](#)
- [Reusable Mappings](#)
- [Editing Mappings Using the Property Inspector and the Structure Panel](#)
- [Flow Control and Static Control](#)
- [Designing E-LT and ETL-Style Mappings](#)

8.1 Introduction to Mappings

Mappings are the logical and physical organization of your data sources, targets, and the transformations through which the data flows from source to target. You create and manage mappings using the mapping editor, a new feature of ODI 12c.

The mapping editor opens whenever you open a mapping. Mappings are organized in folders under individual projects, found under Projects in the Designer Navigator.

8.1.1 Parts of a Mapping

A mapping is made up of and defined by the following parts:

- **Datastores**

Data from source datastores is extracted by a mapping, and can be filtered during the loading process. Target datastores are the elements that are loaded by the mapping. Datastores act as [Projector Components](#).

Datastores that will be used as sources and targets of the loading process must exist in data models before you can use them in a mapping. See [Chapter 3, "Creating and Using Data Models and Datastores"](#) for more information.

- **Datasets**

Optionally, you can use datasets within mappings as sources. A Dataset is a logical container organizing datastores by an entity relationship declared as joins and

filters, rather than the flow mechanism used elsewhere in mappings. Datasets operate similarly to ODI 11g interfaces, and if you import 11g interfaces into ODI 12c, ODI will automatically create datasets based on your interface logic. Datasets act as [Selector Components](#).

- **Reusable Mappings**

Reusable mappings are modular, encapsulated flows of components which you can save and re-use. You can place a reusable mapping inside another mapping, or another reusable mapping (that is, reusable mappings may be nested). A reusable mapping can also include datastores as sources and targets itself, like other mapping components. Reusable mappings act as [Projector Components](#).

- **Other Components**

ODI provides additional components that are used in between sources and targets to manipulate the data. These components are available on the component palette in the mapping diagram.

The following are the components available by default in the component palette:

- Expression
- Aggregate
- Distinct
- Set
- Filter
- Join
- Lookup
- Pivot
- Sort
- Split
- Subquery Filter
- Table Function
- Unpivot

- **Connectors**

Connectors create a flow of data between mapping components. Most components can have both input and output connectors. Datastores with only output connectors are considered sources; datastores with only input connectors are considered targets. Some components can support multiple input or output connectors; for example, the split component supports two or more output connectors, allowing you to split data into multiple downstream flows.

- **Connector points** define the connections between components inside a mapping. A connector point is a single pathway for input or output for a component.
- **Connector ports** are the small circles on the left and/or right sides of components displayed in the mapping diagram.

In the mapping diagram, two components connected by a single visible line between their connector ports could have one or more connector points. The diagram only shows a single line to represent all of the connections between two

components. You can select the line to show details about the connection in the property inspector.

- **Staging Schemas**

Optionally, you can specify a staging area for a mapping or for a specific physical mapping design of a mapping. If you want to define a different staging area than any of the source or target datastores, you must define the correct physical and logical schemas in the mapping's execution context before creating a mapping. See [Chapter 2, "Overview of Oracle Data Integrator Topology"](#) for more information.

- **Knowledge Modules**

Knowledge modules define how data will be transferred between data servers and loaded into data targets. Knowledge Modules (IKMs, LKMs, EKMs, and CKMs) that will be selected in the flow must be imported into the project or must be available as global Knowledge Modules.

IKMs allow you to define (or specify) how the actual transformation and loading is performed.

LKMs allow you to specify how the transfer of the data between one data server to another data server is performed.

When used as flow control, CKMs allow you to check for errors in the data flow during the loading of records into a target datastore. When used as static control, CKMs can be used to check for any errors in a table. You can launch static control at any time on a model to see if the data satisfies constraints.

You can select a strategy to perform these tasks by selecting an appropriate KM. For example, you can decide whether to use a JDBC to transfer data between two databases, or use an Oracle database link if the transfer is between two Oracle databases.

See [Chapter 6, "Creating an Integration Project"](#) for more information.

- **Variables, Sequences, and User Functions**

Variables, Sequences, and User Functions that will be used in expressions within your mappings must be created in the project. See [Chapter 11, "Creating and Using Procedures, Variables, Sequences, and User Functions"](#) for more information.

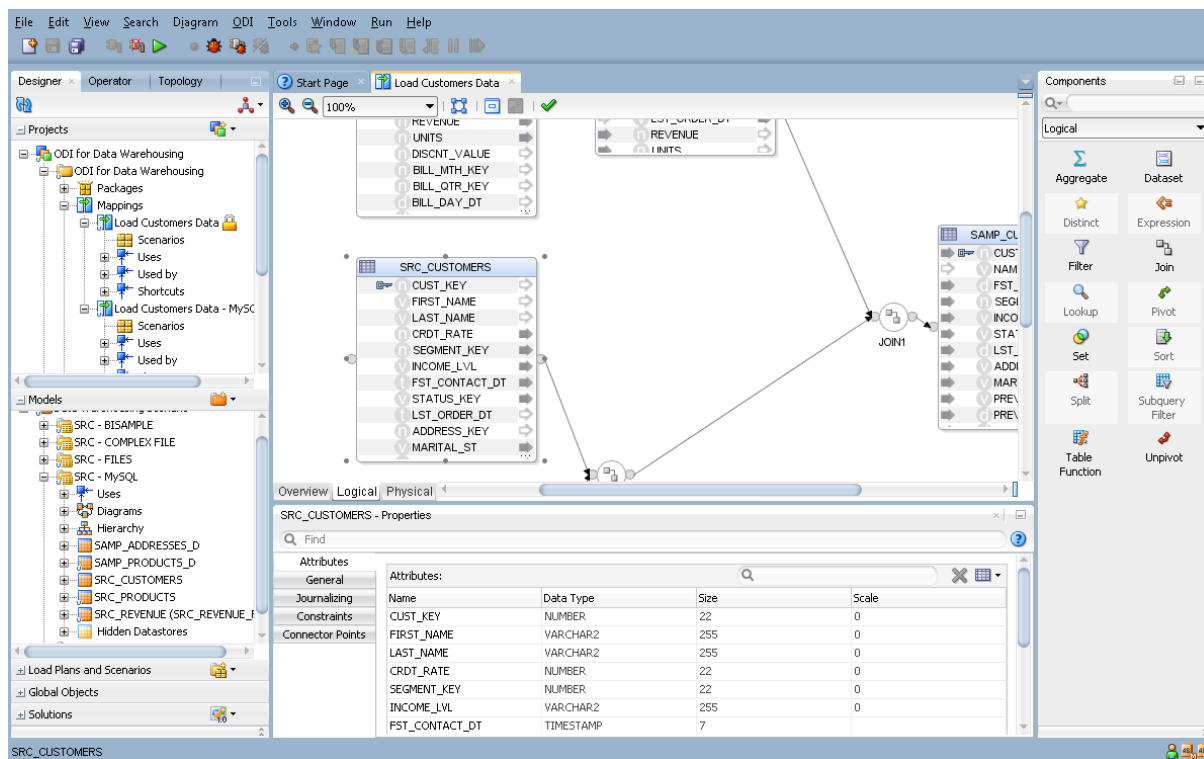
8.1.2 Navigating the Mapping Editor

The mapping editor provides a single environment for designing and editing mappings.

Mappings are organized within folders in a project in the Designer Navigator. Each folder has a mappings node, within which all mappings are listed.

To open the mapping editor, right-click an existing mapping and select **Open**, or double-click the mapping. To create a new mapping, right-click the **Mappings** node and select **New Mapping**. The mapping is opened as a tab on the main pane of ODI Studio. Select the tab corresponding to a mapping to view the mapping editor.

Figure 8–1 Mapping Editor



The mapping editor consists of the sections described in [Table 8–1](#):

Table 8–1 Mapping Editor Sections

Section	Location in Figure 8–1	Description
Mapping Diagram	Middle	<p>The mapping diagram displays an editable logical or physical view of a mapping. These views are sometimes called the logical diagram or the physical diagram.</p> <p>You can drag datastores into the diagram from the Models tree, and Reusable Mappings from the Global Objects or Projects tree, into the mapping diagram. You can also drag components from the component palette to define various data operations.</p>
Mapping Editor tabs	Middle, at the bottom of the mapping diagram	<p>The Mapping Editor tabs are ordered according to the mapping creation process. These tabs are:</p> <ul style="list-style-type: none"> Overview: displays the general properties of the mapping Logical: displays the logical organization of the mapping in the mapping diagram Physical: displays the physical organization of the mapping in the mapping diagram
Property Inspector	Bottom	<p>Displays properties for the selected object.</p> <p>If the Property Inspector does not display, select Properties from the Window menu.</p>

Table 8–1 (Cont.) Mapping Editor Sections

Section	Location in Figure 8–1	Description
Component Palette	Right	<p>Displays the mapping components you can use for creating mappings. You can drag and drop components into the logical mapping diagram from the components palette.</p> <p>If the Component Palette does not display, select Components from the Window menu.</p>
Structure Panel	Not shown	<p>Displays a text-based hierarchical tree view of a mapping, which is navigable using the tab and arrow keys.</p> <p>The Structure Panel does not display by default. To open it, select Structure from the Window menu.</p>
Thumbnail Panel	Not shown	<p>Displays a miniature graphic of a mapping, with a rectangle indicating the portion currently showing in the mapping diagram. This panel is useful for navigating very large or complex mappings.</p> <p>The Thumbnail Panel does not display by default. To open it, select Thumbnail from the Window menu.</p>

8.2 Creating a Mapping

Creating a mapping follows a standard process which can vary depending on the use case.

Using the logical diagram of the mapping editor, you can construct your mapping by dragging components onto the diagram, dragging connections between the components, dragging attributes across those connections, and modifying the properties of the components using the property inspector. When the logical diagram is complete, you can use the physical diagram to define where and how the integration process will run on your physical infrastructure. When the logical and physical design of your mapping is complete, you can run it.

The following step sequence is usually performed when creating a mapping, and can be used as a guideline to design your first mappings:

1. [Creating a New Mapping](#)
2. [Adding and Removing Components](#)
3. [Connecting and Configuring Components](#)
4. [Defining a Physical Configuration](#)
5. [Running Mappings](#)

Note: You can also use the Property Inspector and the Structure Panel to perform the steps 2 to 5. See ["Editing Mappings Using the Property Inspector and the Structure Panel"](#) on page 8-41 for more information.

8.2.1 Creating a New Mapping

To create a new mapping:

1. In Designer Navigator select the **Mappings** node in the folder under the project where you want to create the mapping.
2. Right-click and select **New Mapping**. The New Mapping dialog is displayed.
3. In the New Mapping dialog, fill in the mapping **Name**. Optionally, enter a **Description**. If you want the new mapping to contain a new empty dataset, select **Create Empty Dataset**. Click **OK**.

Note: You can add and remove datasets (including this empty dataset) after you create a mapping. Datasets are entirely optional and all behavior of a dataset can be created using other components in the mapping editor.

In ODI 12c, Datasets offer you the option to create data flows using the entity relationship method familiar to users of previous versions of ODI. In some cases creating an entity relationship diagram may be faster than creating a flow diagram, or make it easier and faster to introduce changes.

When a physical diagram is calculated based on a logical diagram containing a Dataset, the entity relationships in the Dataset are automatically converted by ODI into a flow diagram and merged with the surrounding flow. You do not need to be concerned with how the flow is connected.

Your new mapping opens in a new tab in the main pane of ODI Studio.

Tip: To display the editor of a datastore, a reusable mapping, or a dataset that is used in the Mapping tab, you can right-click the object and select **Open**.

8.2.2 Adding and Removing Components

Add components to the logical diagram by dragging them from the Component Palette. Drag datastores and reusable mappings from the Designer Navigator.

Delete components from a mapping by selecting them, and then either pressing the Delete key, or using the right-click context menu to select Delete. A confirmation dialog is shown.

Source and target datastores are the elements that will be extracted by, and loaded by, the mapping.

Between the source and target datastores are arranged all the other components of a mapping. When the mapping is run, data will flow from the source datastores, through the components you define, and into the target datastores.

Preserving and Removing Downstream Expressions

Where applicable, when you delete a component, a check box in the confirmation dialog allows you to preserve, or remove, downstream expressions; such expressions may have been created when you connected or modified a component. By default ODI preserves these expressions.

This feature allows you to make changes to a mapping without destroying work you have already done. For example, when a source datastore is mapped to a target datastore, the attributes are all mapped. You then realize that you need to filter the source data. To add the filter, one option is to delete the connection between the two

datastores, but preserve the expressions set on the target datastore, and then connect a filter in the middle. None of the mapping expressions are lost.

8.2.3 Connecting and Configuring Components

Create connectors between components by dragging from the originating connector port to the destination connector port. Connectors can also be implicitly created by dragging attributes between components. When creating a connector between two ports, an attribute matching dialog may be shown which allows you to automatically map attributes based on name or position.

8.2.3.1 Attribute Matching

The Attribute Matching Dialog is displayed when a connector is drawn to a projector component (see: "[Projector Components](#)" on page 8-11) in the Mapping Editor. The Attribute Matching Dialog gives you an option to automatically create expressions to map attributes from the source to the target component based on a matching mechanism. It also gives the option to create new attributes on the target based on the source, or new attributes on the source based on the target.

This feature allows you to easily define a set of attributes in a component that are derived from another component. For example, you could drag a connection from a new, empty Set component to a downstream target datastore. If you leave checked the **Create Attributes On Source** option in the Attribute Matching dialog, the Set component will be populated with all of the attributes of the target datastore. When you connect the Set component to upstream components, you will already have the target attributes ready for you to map the upstream attributes to.

8.2.3.2 Connector Points and Connector Ports

Review "[Connectors](#)" on page 8-2 for an introduction to ODI connector terminology.

You can click a connector port on one component and drag a line to another component's connector port to define a connection. If the connection is allowed, ODI will either use an unused existing connector point on each component, or create an additional connector point as needed. The connection is displayed in the mapping diagram with a line drawn between the connector ports of the two connected components. Only a single line is shown even if two components have multiple connections between them.

Most components can use both input and output connectors to other components, which are visible in the mapping diagram as small circles on the sides of the component. The component type may place limitations on how many connectors of each type are allowed, and some components can have only input or only output connections.

Some components allow the addition or deletion of connector points using the property inspector.

For example, a Join component by default has two input connector points and one output connector point. It is allowed to have more than two inputs, though. If you drag a third connection to the input connector port of a join component, ODI creates a third input connector point. You can also select a Join component and, in the property inspector, in the Connector Points section, click the green plus icon to add additional Input Connector Points.

Note: You cannot drag a connection to or from an input port that already has the maximum number of connections. For example, a target datastore can only have one input connector point; if you try to drag another connection to the input connector port, no connection is created.

You can delete a connector by right-clicking the line between two connector points and selecting **Delete**, or by selecting the line and pressing the Delete key.

8.2.3.3 Defining New Attributes

When you add components to a mapping, you may need to create attributes in them in order to move data across the flow from sources, through intermediate components, to targets. Typically you define new attributes to perform transformations of the data.

Use any of the following methods to define new attributes:

- **Attribute Matching Dialog:** This dialog is displayed in certain cases when dragging a connection from a connector port on one component to a connector port on another, when at least one component is a projector component.

The attribute matching dialog includes an option to create attributes on the target. If target already has attributes with matching names, ODI will automatically map to these attributes. If you choose **By Position**, ODI will map the first attributes to existing attributes in the target, and then add the rest (if there are more) below it. For example, if there are three attributes in the target component, and the source has 12, the first three attributes map to the existing attributes, and then the remaining nine are copied over with their existing labels.
- **Drag and drop attributes:** Drag and drop a single (or multi-selected) attribute from a one component into another component (into a blank area of the component graphic, not on top of an existing attribute). ODI creates a connection (if one did not already exist), and also creates the attribute.

Tip: If the graphic for a component is "full", you can hover over the attributes and a scroll bar appears on the right. Scroll to the bottom to expose a blank line. You can then drag attributes to the blank area.

If you drag an attribute onto another attribute, ODI maps it into that attribute, even if the names do not match. This does not create a new attribute on the target component.
- **Add new attributes in the property inspector:** In the property inspector, on the **Attributes** tab, use the green plus icon to create a new attribute. You can select or enter the new attribute's name, data type, and other properties in the **Attributes** table. You can then map to the new attribute by dragging attributes from other components onto the new attribute.

Caution: ODI will allow you to create an illegal data type connection. Therefore, you should always set the appropriate data type when you create a new attribute. For example, if you intend to map an attribute with a DATE data type to a new attribute, you should set the new attribute to have the DATE type as well.

Type-mismatch errors will be caught during execution as a SQL error.

Note: From ODI 12.2.1 onwards, when the DB2 TIME column is mapped to the target column, the target column displays only the time and omits the date.

8.2.3.4 Defining Expressions and Conditions

Expressions and conditions are used to map individual attributes from component to component. Component types determine the default expressions and conditions that will be converted into the underlying code of your mapping.

For example, any target component has an expression for each attribute. A filter, join, or lookup component will use code (such as SQL) to create the expression appropriate to the component type.

Tip: When an expression is set on the target, any source attributes referenced by that expression are highlighted in magenta in the upstream sources. For example, an expression `emp.empno` on the target column `tgt_empno`, when `tgt_empno` is selected (by clicking on it), the attribute `empno` on the source datastore `emp` is highlighted.

This highlighting function is useful for rapidly verifying that each desired target attribute has an expression with valid cross references. If an expression is manually edited incorrectly, such as if a source attribute is misspelled, the cross reference will be invalid, and no source attribute will be highlighted when clicking that target attribute.

You can modify the expressions and conditions of any component by modifying the code displayed in various property fields.

Note: Oracle recommends using the expression editor instead of manually editing expressions in most cases. Selection of a source attribute from the expression editor will always give the expression a valid cross reference, minimizing editing errors. For more information, see "[The Expression Editor](#)" on page 8-12.

Expressions have a result type, such as VARCHAR or NUMERIC. The result type of conditions are boolean, meaning, the result of a condition should always evaluate to TRUE or FALSE. A condition is needed for filter, join, and lookup (selector) components, while an expression is used in datastore, aggregate, and distinct (projector) components, to perform some transformation or create the attribute-level mappings.

Every projector component can have expressions on its attributes. (For most projector components, an attribute has one expression, but the attribute of the Set component can have multiple expressions.) If you modify the expression for an attribute, a small "f" icon appears on the attribute in the logical diagram. This icon provides a visual cue that a function has been placed there.

To define the mapping of a target attribute:

1. In the mapping editor, select an attribute to display the attribute's properties in the Property Inspector.
2. In the **Target** tab (for expressions) or **Condition** tab (for conditions), modify the **Expression** or **Condition** field(s) to create the required logic.

Tip: The attributes from any component in the diagram can be drag-and-dropped into an expression field to automatically add the fully-qualified attribute name to the code.

3. Optionally, select or hover over any field in the property inspector containing an expression, and then click the gear icon that appears to the right of the field, to open the advanced **Expression Editor**.

The attributes on the left are only the ones that are in scope (have already been connected). So if you create a component with no upstream or downstream connection to a component with attributes, no attributes are listed.

4. Optionally, after modifying an expression or condition, consider validating your mapping to check for errors in your SQL code. Click the green check mark icon at the top of the logical diagram. Validation errors, if any, will be displayed in a panel.

8.2.4 Defining a Physical Configuration

In the **Physical** tab of the mapping editor, you define the loading and integration strategies for mapped data. Oracle Data Integrator automatically computes the flow depending on the configuration in the mapping's logical diagram. It proposes default knowledge modules (KMs) for the data flow. The **Physical** tab enables you to view the data flow and select the KMs used to load and integrate data.

For more information about physical design, see "[Physical Design](#)" on page 8-33.

8.2.5 Running Mappings

Once a mapping is created, you can run it. This section briefly summarizes the process of running a mapping. For detailed information about running your integration processes, see: "Running Integration Processes" in *Administering Oracle Data Integrator*.

To run a mapping:

1. From the Projects menu of the Designer Navigator, right-click a mapping and select **Run**.

Or, with the mapping open in the mapping editor, click the run icon in the toolbar.
Or, select **Run** from the **Run** menu.

2. In the **Run** dialog, select the execution parameters:
 - Select the **Context** into which the mapping must be executed. For more information about contexts, see: "[Contexts](#)" on page 2-2.
 - Select the **Physical Mapping Design** you want to run. See: "[Creating and Managing Physical Mapping Designs](#)" on page 8-39.
 - Select the **Logical Agent** that will run the mapping. The object can also be executed using the agent that is built into Oracle Data Integrator Studio, by selecting Local (No Agent). For more information about logical agents, see: "[Agents](#)" on page 2-2.
 - Select a **Log Level** to control the detail of messages that will appear in the validator when the mapping is run. For more information about logging, see: "Managing the Log" in *Administering Oracle Data Integrator*.
 - Check the **Simulation** box if you want to preview the code without actually running it. In this case no data will be changed on the source or target

datastores. For more information, see: "Simulating an Execution" in *Administering Oracle Data Integrator*.

3. Click **OK**.
4. The **Information** dialog appears. If your session started successfully, you will see "Session started."
5. Click **OK**.

Notes:

- When you run a mapping, the Validation Results pane opens. You can review any validation warnings or errors there.
 - You can see your session in the Operator navigator Session List. Expand the Sessions node and then expand the mapping you ran to see your session. The session icon indicates whether the session is still running, completed, or stopped due to errors. For more information about monitoring your sessions, see: "Monitoring Integration Processes" in *Administering Oracle Data Integrator*.
-
-

8.3 Using Mapping Components

In the logical view of the mapping editor, you design a mapping by combining datastores with other components. You can use the mapping diagram to arrange and connect components such as datasets, filters, sorts, and so on. You can form connections between datastores and components by dragging lines between the connector ports displayed on these objects.

Mapping components can be divided into two categories which describe how they are used in a mapping: projector components and selector components.

Projector Components

Projectors are components that influence the attributes present in the data that flows through a mapping. Projector components define their own attributes: attributes from preceding components are mapped through expressions to the projector's attributes. A projector hides attributes originating from preceding components; all succeeding components can only use the attributes from the projector.

Review the following topics to learn how to use the various projector components:

- ["Source and Target Datastores"](#) on page 8-13
- ["Creating Multiple Targets"](#) on page 8-14
- ["Adding a Reusable Mapping"](#) on page 8-15
- ["Creating Aggregates"](#) on page 8-15
- ["Creating Distincts"](#) on page 8-16
- ["Creating Pivots"](#) on page 8-21
- ["Creating Sets"](#) on page 8-23
- ["Creating Subquery Filters"](#) on page 8-25
- ["Creating Table Functions"](#) on page 8-26
- ["Creating Unpivots"](#) on page 8-28
- ["Creating Flatten Components"](#) on page 8-30

- ["Creating Jagged Components"](#) on page 8-31

Selector Components

Selector components reuse attributes from preceding components. Join and Lookup selectors combine attributes from the preceding components. For example, a Filter component following a datastore component reuses all attributes from the datastore component. As a consequence, selector components don't display their own attributes in the diagram and as part of the properties; they are displayed as a round shape. (The Expression component is an exception to this rule.)

When mapping attributes from a selector component to another component in the mapping, you can select and then drag an attribute from the source, across a chain of connected selector components, to a target datastore or next projector component. ODI will automatically create the necessary queries to bring that attribute across the intermediary selector components.

Review the following topics to learn how to use the various selector components:

- ["Creating Expressions"](#) on page 8-17
- ["Creating Filters"](#) on page 8-17
- ["Creating Joins and Lookups"](#) on page 8-18
- ["Creating Sorts"](#) on page 8-24
- ["Creating Splits"](#) on page 8-25
- ["Creating a Dataset in a Mapping"](#) on page 8-33

8.3.1 The Expression Editor

Most of the components you use in a mapping are actually representations of an expression in the code that acts on the data as it flows from your source to your target datastores. When you create or modify these components, you can edit the expression's code directly in the Property Inspector.

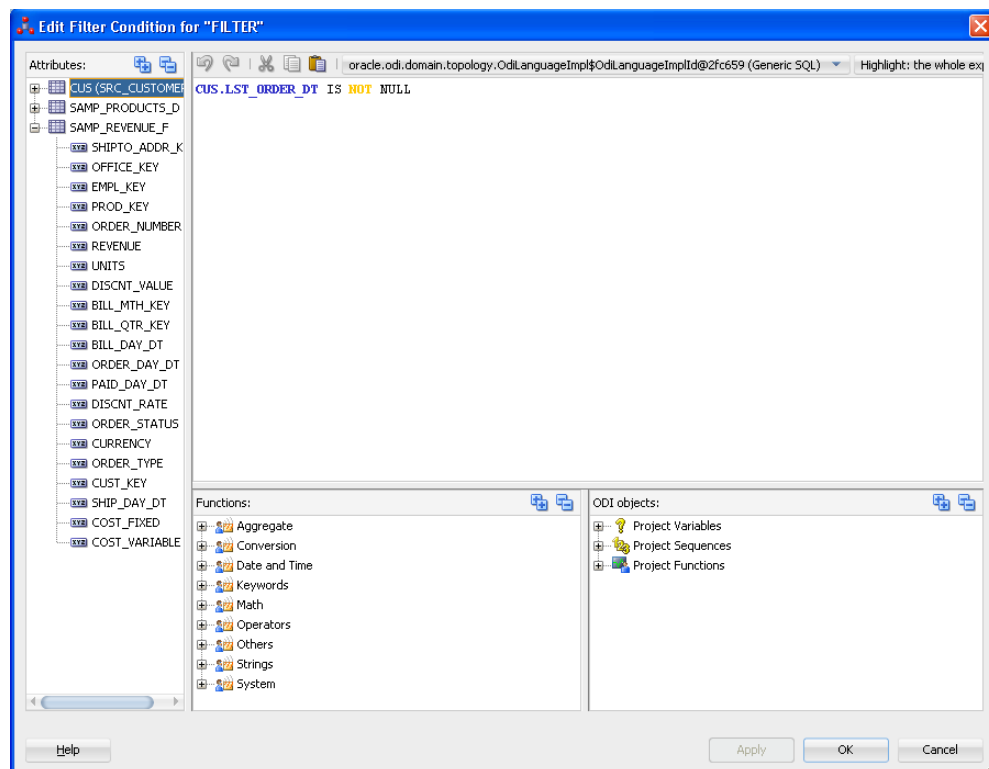
To assist you with more complex expressions, you can also open an advanced editor called the Expression Editor. (In some cases, the editor is labeled according to the type of component; for example, from a Filter component, the editor is called the Filter Condition Advanced Editor. However, the functionality provided is the same.)

To access the Expression Editor, select a component, and in the Property Inspector, select or hover over with the mouse pointer any field containing code. A gear icon appears to the right of the field. Click the gear icon to open the Expression Editor.

For example, to see the gear icon in a Filter component, select or hover over the Filter Condition field on the Condition tab; to see the gear icon in a Datastore component, select or hover over the Journalized Data Filter field of the Journalizing tab.

A typical example view of the Expression Editor is shown in [Figure 8-2](#)

Figure 8–2 Example Expression Editor



The Expression Editor is made up of the following panels:

- **Attributes:** This panel appears on the left of the Expression Editor. When editing an expression for a mapping, this panel contains the names of attributes which are "in scope," meaning, attributes that are currently visible and can be referenced by the expression of the component. For example, if a component is connected to a source datastore, all of the attributes of that datastore are listed.
- **Expression:** This panel appears in the middle of the Expression Editor. It displays the current code of the expression. You can directly type code here, or drag and drop elements from the other panels.
- **Technology functions:** This panel appears below the expression. It lists the language elements and functions appropriate for the given technology.
- **Variables, Sequences, User Functions and odiRef API:** This panel appears to the right of the technology functions and contains:
 - Project and global Variables.
 - Project and global Sequences.
 - Project and global User-Defined Functions.
 - OdiRef Substitution Methods.

Standard editing functions (cut/copy/paste/undo/redo) are available using the tool bar buttons.

8.3.2 Source and Target Datastores

To insert a source or target datastore in a mapping:

1. In the Designer Navigator, expand the **Models** tree and expand the model or sub-model containing the datastore to be inserted as a source or target.
2. Select this datastore, then drag it into the mapping panel. The datastore appears.
3. To make the datastore a source, drag a link from the output (right) connector of the datastore to one or more components. A datastore is not a source until it has at least one outgoing connection.

To make the datastore a target, drag a link from a component to the input (left) connector of the datastore. A datastore is not a target until it has an incoming connection.

Once you have defined a datastore you may wish to view its data.

To display the data of a datastore in a mapping:

1. Right-click the title of the datastore in the mapping diagram.
2. Select **Data...**

The Data Editor opens.

8.3.3 Creating Multiple Targets

In Oracle Data Integrator 12c, creating multiple targets in a mapping is straightforward. Every datastore component which has inputs but no outputs in the logical diagram is considered a target.

ODI allows splitting a component output into multiple flows at any point of a mapping. You can also create a single mapping with multiple independent flows, avoiding the need for a package to coordinate multiple mappings.

The output port of many components can be connected to multiple downstream components, which will cause all rows of the component result to be processed in each of the downstream flows. If rows should be routed or conditionally processed in the downstream flows, consider using a split component to define the split conditions.

See Also: ["Creating Splits"](#) on page 8-25

8.3.3.1 Specifying Target Order

Mappings with multiple targets do not, by default, follow a defined order of loading data to targets. You can define a partial or complete order by using the **Target Load Order** property. Targets which you do not explicitly assign an order will be loaded in an arbitrary order by ODI.

Note: Target load order also applies to reusable mappings. If a reusable mapping contains a source or a target datastore, you can include the reusable mapping component in the target load order property of the parent mapping.

The order of processing multiple targets can be set in the **Target Load Order** property of the mapping:

1. Click the background in the logical diagram to deselect objects in the mapping. The property inspector displays the properties for the mapping.
2. In the property inspector, accept the default target load order, or enter a new target load order, in the **Target Load Order** field.

Note: A default load order is automatically computed based on primary key/foreign key relationships of the target datastores in the mapping. You can modify this default if needed, even if the resultant load order conflicts with the primary key/foreign key relationship. A warning will be displayed when you validate the mapping in this case.

Select or hover over the **Target Load Order** field and click the gear icon to open the **Target Load Order Dialog**. This dialog displays all available datastores (and reusable mappings containing datastores) that can be targets, allowing you to move one or more to the **Ordered Targets** field. In the **Ordered Targets** field, use the icons on the right to rearrange the order of processing.

Tip: Target Order is useful when a mapping has multiple targets and there are foreign key (FK) relationships between the targets. For example, suppose a mapping has two targets called EMP and DEPT, and EMP.DEPTNO is a FK to DEPT.DEPTNO. If the source data contains information about the employee and the department, the information about the department (DEPT) must be loaded first before any rows about the employee can be loaded (EMP). To ensure this happens, the target load order should be set to DEPT, EMP.

8.3.4 Adding a Reusable Mapping

Reusable mappings may be stored within folders in a project, or as global objects within the Global Objects tree, of the Designer Navigator.

To add a reusable mapping to a mapping:

1. To add a reusable mapping stored within the current project:

In the Designer Navigator, expand the **Projects** tree and expand the tree for the project you are working on. Expand the Reusable Mappings node to list all reusable mappings stored within this project.

To add a global reusable mapping:

In the Designer Navigator, expand the Global Objects tree, and expand the Reusable Mappings node to list all global reusable mappings.

2. Select a reusable mapping, and drag it into the mapping diagram. A reusable mapping component is added to the diagram as an interface to the underlying reusable mapping.

8.3.5 Creating Aggregates

The aggregate component is a projector component (see: "[Projector Components](#)" on page 8-11) which groups and combines attributes using aggregate functions, such as average, count, maximum, sum, and so on. ODI will automatically select attributes without aggregation functions to be used as group-by attributes. You can override this by using the **Is Group By** and **Manual Group By Clause** properties.

To create an aggregate component:

1. Drag and drop the aggregate component from the component palette into the logical diagram.
2. Define the attributes of the aggregate if the attributes will be different from the source components. To do this, select the **Attributes** tab in the property inspector,

and click the green plus icon to add attributes. Enter new attribute names in the **Target** column and assign them appropriate values.

If attributes in the aggregate component will be the same as those in a source component, use attribute matching (see Step 4).

3. Create a connection from a source component by dragging a line from the connector port of the source to the connector port of the aggregate component.
4. The **Attribute Matching** dialog will be shown. If attributes in the aggregate component will be the same as those in a source component, check the **Create Attributes on Target** box (see: "[Attribute Matching](#)" on page 8-7).
5. If necessary, map all attributes from source to target that were not mapped through attribute matching, and create transformation expressions as necessary (see: "[Defining Expressions and Conditions](#)" on page 8-9).
6. In the property inspector, the attributes are listed in a table on the **Attributes** tab. Specify aggregation functions for each attribute as needed. By default all attributes not mapped using aggregation functions (such as sum, count, avg, max, min, and so on) will be used as Group By.

You can modify an aggregation expression by clicking the attribute. For example, if you want to calculate average salary per department, you might have two attributes: the first attribute called `AVG_SAL`, which you give the expression `AVG(EMP.SAL)`, while the second attribute called `DEPTNO` has no expression. If **Is Group By** is set to `Auto`, `DEPTNO` will be automatically included in the `GROUP BY` clause of the generated code.

You can override this default by changing the property **Is Group By** on a given attribute from `Auto` to `Yes` or `No`, by double-clicking on the table cell and selecting the desired option from the drop down list.

You can set a different `GROUP BY` clause other than the default for the entire aggregate component. Select the **General** tab in the property inspector, and then set a **Manual Group by Clause**. For example, set the **Manual Group by Clause** to `YEAR(customer.birthdate)` to group by birthday year.

7. Optionally, add a `HAVING` clause by setting the **HAVING** property of the aggregate component: for example, `SUM(order.amount) > 1000`.

8.3.6 Creating Distincts

A distinct is a projector component (see: "[Projector Components](#)" on page 8-11) that projects a subset of attributes in the flow. The values of each row have to be unique; the behavior follows the rules of the SQL `DISTINCT` clause.

To select distinct rows from a source datastore:

1. Drag and drop a Distinct component from the component palette into the logical diagram.
2. Connect the preceding component to the Distinct component by dragging a line from the preceding component to the Distinct component.

The **Attribute Mapping Dialog** will appear: select **Create Attributes On Target** to create all of the attributes in the Distinct component. Alternatively, you can manually map attributes as desired using the **Attributes** tab in the property inspector.

3. The distinct component will now filter all rows that have all projected attributes matching.

8.3.7 Creating Expressions

An expression is a selector component (see: "[Selector Components](#)" on page 8-12) that inherits attributes from a preceding component in the flow and adds additional reusable attributes. An expression can be used to define a number of reusable expressions within a single mapping. Attributes can be renamed and transformed from source attributes using SQL expressions. The behavior follows the rules of the SQL `SELECT` clause.

The best use of an expression component is in cases where intermediate transformations are used multiple times, such as when pre-calculating fields that are used in multiple targets.

If a transformation is used only once, consider performing the transformation in the target datastore or other component.

Tip: If you want to reuse expressions across multiple mappings, consider using reusable mappings or user functions, depending on the complexity. See: "[Reusable Mappings](#)" on page 8-40, and "[Working with User Functions](#)" on page 11-27.

To create an expression component:

1. Drag and drop an Expression component from the component palette into the logical diagram.
2. Connect a preceding component to the Expression component by dragging a line from the preceding component to the Expression component.

The **Attribute Mapping Dialog** will appear; select **Create Attributes On Target** to create all of the attributes in the Expression component.

In some cases you might want the expression component to match the attributes of a downstream component. In this case, connect the expression component with the downstream component first and select **Create Attributes on Source** to populate the Expression component with attributes from the target.

3. Add attributes to the expression component as desired using the **Attributes** tab in the property inspector. It might be useful to add attributes for pre-calculated fields that are used in multiple expressions in downstream components.
4. Edit the expressions of individual attributes as necessary (see: "[Defining Expressions and Conditions](#)" on page 8-9).

8.3.8 Creating Filters

A filter is a selector component (see: "[Selector Components](#)" on page 8-12) that can select a subset of data based on a filter condition. The behavior follows the rules of the SQL `WHERE` clause.

Filters can be located in a dataset or directly in a mapping as a flow component.

When used in a dataset, a filter is connected to one datastore or reusable mapping to filter all projections of this component out of the dataset. For more information, see [Creating a Mapping Using a Dataset](#).

To define a filter in a mapping:

1. Drag and drop a Filter component from the component palette into the logical diagram.

2. Drag an attribute from the preceding component onto the filter component. A connector will be drawn from the preceding component to the filter, and the attribute will be referenced in the filter condition.

In the **Condition** tab of the Property Inspector, edit the **Filter Condition** and complete the expression. For example, if you want to select from the CUSTOMER table (that is the source datastore with the CUSTOMER alias) only those records with a NAME that is not null, an expression could be `CUSTOMER.NAME IS NOT NULL`.

Tip: Click the gear icon to the right of the **Filter Condition** field to open the **Filter Condition Advanced Editor**. The gear icon is only shown when you have selected or are hovering over the Filter Condition field with your mouse pointer. For more information about the Filter Condition Advanced Editor, see: "[The Expression Editor](#)" on page 8-12.

3. Optionally, on the **General** tab of the Property Inspector, enter a new name in the **Name** field. Using a unique name is useful if you have multiple filters in your mapping.
4. Optionally, set an **Execute on Hint**, to indicate your preferred execution location: No hint, Source, Staging, or Target. The physical diagram will locate the execution of the filter according to your hint, if possible. For more information, see "[Configuring Execution Locations](#)" on page 8-36.

8.3.9 Creating Joins and Lookups

This section contains the following topics:

- [About Joins](#)
- [About Lookups](#)
- [Creating a Join or Lookup](#)

About Joins

A Join is a selector component (see: "[Selector Components](#)" on page 8-12) that creates a join between multiple flows. The attributes from upstream components are combined as attributes of the Join component.

A Join can be located in a dataset or directly in a mapping as a flow component. A join combines data from two or more data flows, which may be datastores, datasets, reusable mappings, or combinations of various components.

When used in a dataset, a join combines the data of the datastores using the selected join type. For more information, see [Creating a Mapping Using a Dataset](#).

A join used as a flow component can join two or more sources of attributes, such as datastores or other upstream components. A join condition can be formed by dragging attributes from two or more components successively onto a join component in the mapping diagram; by default the join condition will be an equi-join between the two attributes.

About Lookups

A Lookup is a selector component (see: "[Selector Components](#)" on page 8-12) that returns data from a lookup flow being given a value from a driving flow. The attributes of both flows are combined, similarly to a join component.

Lookups can be located in a dataset or directly in a mapping as a flow component.

When used in a dataset, a Lookup is connected to two datastores or reusable mappings combining the data of the datastores using the selected join type. For more information, see [Creating a Mapping Using a Dataset](#).

Lookups used as flow components (that is, not in a dataset) can join two flows. A lookup condition can be created by dragging an attribute from the driving flow and then the lookup flow onto the lookup component; the lookup condition will be an equi-join between the two attributes.

The **Multiple Match Rows** property defines which row from the lookup result must be selected as the lookup result if the lookup returns multiple results. Multiple rows are returned when the lookup condition specified matches multiple records.

You can select one of the following options to specify the action to perform when multiple rows are returned by the lookup operation:

- **Error: multiple rows will cause a mapping failure**

This option indicates that when the lookup operation returns multiple rows, the mapping execution fails.

Note: In ODI 12.1.3, the **Deprecated - Error: multiple rows will cause a mapping failure** option with the `EXPRESSION_IN_SELECT` option value is deprecated. It is included for backward compatibility with certain patched versions of ODI 12.1.2.

This option is replaced with the `ERROR_WHEN_MULTIPLE_ROW` option of **Error: multiple rows will cause a mapping failure**.

- **All Rows (number of result rows may differ from the number of input rows)**

This option indicates that when the lookup operation returns multiple rows, all the rows should be returned as the lookup result.

Note: In ODI 12.1.3, the **Deprecated - All rows (number of result rows may differ from the number of input rows)** option with the `LEFT_OUTER` option value is deprecated. It is included for backward compatibility with certain patched versions of ODI 12.1.2.

This option is replaced with the `ALL_ROWS` option of **All rows (number of result rows may differ from the number of input rows)**.

- **Select any single row**

This option indicates that when the lookup operation returns multiple rows, any one row from the returned rows must be selected as the lookup result.

- **Select first single row**

This option indicates that when the lookup operation returns multiple rows, the first row from the returned rows must be selected as the lookup result.

- **Select nth single row**

This option indicates that when the lookup operation returns multiple rows, the nth row from the result rows must be selected as the lookup result. When you select this option, the **Nth Row Number** field appears, where you can specify the value of n.

- **Select last single row**

This option indicates that when the lookup operation returns multiple rows, the last row from the returned rows must be selected as the lookup result.

Use the **Lookup Attributes Default Value & Order By** table to specify how the result set that contains multiple rows should be ordered, and what the default value should be if no matches are found for the input attribute in the lookup flow through the lookup condition. Ensure that the attributes are listed in the same order (from top to bottom) in which you want the result set to be ordered. For example, to implement an ordering such as ORDER BY attr2, attr3, and then attr1, the attributes should be listed in the same order. You can use the arrow buttons to change the position of the attributes to specify the order.

The **No-Match Rows** property indicates the action to be performed when there are no rows that satisfy the lookup condition. You can select one of the following options to perform when no rows are returned by the lookup operation:

- **Return no row**

This option does not return any row when no row in the lookup results satisfies the lookup condition.

- **Return a row with the following default values**

This option returns a row that contains default values when no row in the lookup results satisfies the lookup condition. Use the **Lookup Attributes Default Value & Order By**: table below this option to specify the default values for each lookup attribute.

Creating a Join or Lookup

To create a join or a lookup between two upstream components:

1. Drag a join or lookup from the component palette into the logical diagram.
2. Drag the attributes participating in the join or lookup condition from the preceding components onto the join or lookup component. For example, if attribute ID from source datastore CUSTOMER and then CUSTID from source datastore ORDER are dragged onto a join, then the join condition CUSTOMER.ID = ORDER.CUSTID is created.

Note: When more than two attributes are dragged into a join or lookup, ODI compares and combines attributes with an AND operator. For example, if you dragged attributes from sources A and B into a Join component in the following order:

```
A.FIRSTNAME  
B.FIRSTNAME  
A.LASTNAME  
B.LASTNAME
```

The following join condition would be created:

```
A.FIRSTNAME=B.FIRSTNAME AND A.LASTNAME=B.LASTNAME
```

You can continue with additional pairs of attributes in the same way.

You can edit the condition after it is created, as necessary.

3. In the **Condition** tab of the Property Inspector, edit the **Join Condition** or **Lookup Condition** and complete the expression.

Tip: Click the gear icon to the right of the **Join Condition** or **Lookup Condition** field to open the Expression Editor. The gear icon is only shown when you have selected or are hovering over the condition field with your mouse pointer. For more information about the Expression Editor, see: "[The Expression Editor](#)" on page 8-12.

4. Optionally, set an **Execute on Hint**, to indicate your preferred execution location: No hint, Source, Staging, or Target. The physical diagram will locate the execution of the filter according to your hint, if possible.
5. For a join:

Select the **Join Type** by checking the various boxes (**Cross**, **Natural**, **Left Outer**, **Right Outer**, **Full Outer** (by checking both left and right boxes), or (by leaving all boxes empty) **Inner Join**). The text describing which rows are retrieved by the join is updated.

For a lookup:

Select the **Multiple Match Rows** by selecting an option from the drop down list. The Technical Description field is updated with the SQL code representing the lookup, using fully-qualified attribute names.

If applicable, use the **Lookup Attributes Default Value & Order By** table to specify how a result set that contains multiple rows should be ordered.

Select a value for the **No-Match Rows** property to indicate the action to be performed when there are no rows that satisfy the lookup condition.

6. Optionally, for joins, if you want to use an ordered join syntax for this join, check the **Generate ANSI Syntax** box.

The Join Order box will be checked if you enable **Generate ANSI Syntax**, and the join will be automatically assigned an order number.

7. For joins inside of datasets, define the join order. Check the **Join Order** check box, and then in the **User Defined** field, enter an integer. A join component with a smaller join order number means that particular join will be processed first among other joins. The join order number determines how the joins are ordered in the FROM clause. A smaller join order number means that the join will be performed earlier than other joins. This is important when there are outer joins in the dataset.

For example: A mapping has two joins, *JOIN1* and *JOIN2*. *JOIN1* connects *A* and *B*, and its join type is `LEFT OUTER JOIN`. *JOIN2* connects *B* and *C*, and its join type is `RIGHT OUTER JOIN`.

To generate `(A LEFT OUTER JOIN B) RIGHT OUTER JOIN C`, assign a join order 10 for *JOIN1* and 20 for *JOIN2*.

To generate `A LEFT OUTER JOIN (B RIGHT OUTER JOIN C)`, assign a join order 20 for *JOIN1* and 10 for *JOIN2*.

8.3.10 Creating Pivots

A pivot component is a projector component (see: "[Projector Components](#)" on page 8-11) that lets you transform data that is contained in multiple input rows into a single output row. The pivot component lets you extract data from a source once, and

produce one row from a set of source rows that are grouped by attributes in the source data. The pivot component can be placed anywhere in the data flow of a mapping.

8.3.10.1 Example: Pivoting Sales Data

Table 8–2 shows a sample of data from the SALES relational table. The QUARTER attribute has 4 possible character values, one for each quarter of the year. All the sales figures are contained in one attribute, SALES.

Table 8–2 SALES

YEAR	QUARTER	SALES
2010	Q1	10.5
2010	Q2	11.4
2010	Q3	9.5
2010	Q4	8.7
2011	Q1	9.5
2011	Q2	10.5
2011	Q3	10.3
2011	Q4	7.6

Table 8–3 depicts data from the relational table SALES after pivoting the table. The data that was formerly contained in the QUARTER attribute (Q1, Q2, Q3, and Q4) corresponds to 4 separate attributes (Q1_Sales, Q2_Sales, Q3_Sales, and Q4_Sales). The sales figures formerly contained in the SALES attribute are distributed across the 4 attributes for each quarter.

Table 8–3 PIVOTED DATA

Year	Q1_Sales	Q2_Sales	Q3_Sales	Q4_Sales
2010	10.5	11.4	9.5	8.7
2011	9.5	10.5	10.3	7.6

8.3.10.2 The Row Locator

When you use the pivot component, multiple input rows are transformed into a single row based on the row locator. The row locator is an attribute that you must select from the source to correspond with the set of output attributes that you define. It is necessary to specify a row locator to perform the pivot operation.

In this example, the row locator is the attribute QUARTER from the SALES table and it corresponds to the attributes Q1_Sales, Q2_Sales, Q3_Sales, and Q4_Sales attributes in the pivoted output data.

8.3.10.3 Using the Pivot Component

To use a pivot component in a mapping:

1. Drag and drop the source datastore into the logical diagram.
2. Drag and drop a Pivot component from the component palette into the logical diagram.
3. From the source datastore drag and drop the appropriate attributes on the pivot component. In this example, the YEAR attribute.

Note: Do not drag the row locator attribute or the attributes that contain the data values that correspond to the output attributes. In this example, QUARTER is the row locator attribute and SALES is the attribute that contain the data values (sales figures) that correspond to the Q1_Sales, Q2_Sales, Q3_Sales, and Q4_Sales output attributes.

4. Select the pivot component. The properties of the pivot component are displayed in the Property Inspector.
5. Enter a name and description for the pivot component.
6. If required, change the Aggregate Function for the pivot component. The default is MIN.
7. Type in the expression or use the Expression Editor to specify the row locator. In this example, since the QUARTER attribute in the SALES table is the row locator, the expression will be SALES.QUARTER.
8. Under Row Locator Values, click the + sign to add the row locator values. In this example, the possible values for the row locator attribute QUARTER are Q1, Q2, Q3, and Q4.
9. Under Attributes, add output attributes to correspond to each input row. If required, you can add new attributes or rename the listed attributes.
In this example, add 4 new attributes, Q1_Sales, Q2_Sales, Q3_Sales, and Q4_Sales that will correspond to 4 input rows Q1, Q2, Q3, and Q4 respectively.
10. If required, change the expression for each attribute to pick up the sales figures from the source and select a matching row for each attribute.
In this example, set the expressions for each attribute to SALES.SALES and set the matching rows to Q1, Q2, Q3, and Q4 respectively.
11. Drag and drop the target datastore into the logical diagram.
12. Connect the pivot component to the target datastore by dragging a link from the output (right) connector of the pivot component to the input (left) connector of the target datastore.
13. Drag and drop the appropriate attributes of the pivot component on to the target datastore. In this example, YEAR, Q1_Sales, Q2_Sales, Q3_Sales, and Q4_Sales.
14. Go to the physical diagram and assign new KMs if you want to.
Save and execute the mapping to perform the pivot operation.

8.3.11 Creating Sets

A set component is a projector component (see: "[Projector Components](#)" on page 8-11) that combines multiple input flows into one using set operation such as UNION, INTERSECT, EXCEPT, MINUS and others. The behavior reflects the SQL operators.

Note: PigSetCmd does not support the EXCEPT set operation.

Additional input flows can be added to the set component by connecting new flows to it. The number of input flows is shown in the list of Input Connector Points in the

Operators tab. If an input flow is removed, the input connector point needs to be removed as well.

To create a set from two or more sources:

1. Drag and drop a Set component from the component palette into the logical diagram.
2. Define the attributes of the set if the attributes will be different from the source components. To do this, select the **Attributes** tab in the property inspector, and click the green plus icon to add attributes. Select the new attribute names in the **Target** column and assign them appropriate values.

If Attributes will be the same as those in a source component, use attribute matching (see step 4).
3. Create a connection from the first source by dragging a line from the connector port of the source to the connector port of the Set component.
4. The **Attribute Matching** dialog will be shown. If attributes of the set should be the same as the source component, check the **Create Attributes on Target** box (see: "[Attribute Matching](#)" on page 8-7).
5. If necessary, map all attributes from source to target that were not mapped through attribute matching, and create transformation expressions as necessary (see: "[Defining Expressions and Conditions](#)" on page 8-9).
6. All mapped attributes will be marked by a yellow arrow in the logical diagram. This shows that not all sources have been mapped for this attribute; a set has at least two sources.
7. Repeat the connection and attribute mapping steps for all sources to be connected to this set component. After completion, no yellow arrows should remain.
8. In the property inspector, select the **Operators** tab and select cells in the **Operator** column to choose the appropriate set operators (`UNION`, `EXCEPT`, `INTERSECT`, and so on). `UNION` is chosen by default. You can also change the order of the connected sources to change the set behavior.

Note: You can set **Execute On Hint** on the attributes of the set component, but there is also an **Execute On Hint** property for the set component itself. The hint on the component indicates the preferred location where the actual set operation (`UNION`, `EXCEPT`, and so on) is performed, while the hint on an attribute indicates where the preferred location of the expression is performed.

A common use case is that the set operation is performed on a staging execution unit, but some of its expressions can be done on the source execution unit. For more information about execution units, see "[Configuring Execution Locations](#)" on page 8-36.

8.3.12 Creating Sorts

A Sort is a projector component (see: "[Projector Components](#)" on page 8-11) that will apply a sort order to the rows of the processed dataset, using the SQL `ORDER BY` statement.

To create a sort on a source datastore:

1. Drag and drop a Sort component from the component palette into the logical diagram.

2. Drag the attribute to be sorted on from a preceding component onto the sort component. If the rows should be sorted based on multiple attributes, they can be dragged in desired order onto the sort component.
3. Select the sort component and select the **Condition** tab in the property inspector. The **Sorter Condition** field follows the syntax of the SQL `ORDER BY` statement of the underlying database; multiple fields can be listed separated by commas, and `ASC` or `DESC` can be appended after each field to define if the sort will be ascending or descending.

8.3.13 Creating Splits

A Split is a selector component (see: "[Selector Components](#)" on page 8-12) that divides a flow into two or more flows based on specified conditions. Split conditions are not necessarily mutually exclusive: a source row is evaluated against all split conditions and may be valid for multiple output flows.

If a flow is divided unconditionally into multiple flows, no split component is necessary: you can connect multiple downstream components to a single outgoing connector port of any preceding component, and the data output by that preceding component will be routed to all downstream components.

A split component is used to conditionally route rows to multiple preceding flows and targets.

To create a split to multiple targets in a mapping:

1. Drag and drop a Split component from the component palette into the logical diagram.
2. Connect the split component to the preceding component by dragging a line from the preceding component to the split component.
3. Connect the split component to each following component. If either of the upstream or downstream components contain attributes, the Attribute Mapping Dialog will appear. In the **Connection Path** section of the dialog, it will default to the first unmapped connector point and will add connector points as needed. Change this selection if a specific connector point should be used.
4. In the property inspector, open the **Split Conditions** tab. In the **Output Connector Points** table, enter expressions to select rows for each target. If an expression is left empty, all rows will be mapped to the selected target. Check the **Remainder** box to map all rows that have not been selected by any of the other targets.

8.3.14 Creating Subquery Filters

A subquery filter component is a projector component (see: "[Projector Components](#)" on page 8-11) that lets you to filter rows based on the results of a subquery. The conditions that you can use to filter rows are `EXISTS`, `NOT EXISTS`, `IN`, and `NOT IN`.

For example, the EMP datastore contains employee data and the DEPT datastore contains department data. You can use a subquery to fetch a set of records from the DEPT datastore and then filter rows from the EMP datastore by using one of the subquery conditions.

A subquery filter component has two input connector points and one output connector point. The two input connector points are Driver Input connector point and Subquery Filter Input connector point. The Driver Input connector point is where the main datastore is set, which drives the whole query. The Subquery Filter Input connector point is where the datastore that is used in the sub-query is set. In the

example, EMP is the Driver Input connector point and DEPT is the Subquery Filter Input connector point.

To filter rows using a subquery filter component:

1. Drag and drop a subquery filter component from the component palette into the logical diagram.
2. Connect the subquery filter component with the source datastores and the target datastore.
3. Drag and drop the input attributes from the source datastores on the subquery filter component.
4. Drag and drop the output attributes of the subquery filter component on the target datastore.
5. Go to the Connector Points tab and select the input datastores for the driver input connector point and the subquery filter input connector point.
6. Click the subquery filter component. The properties of the subquery filter component are displayed in the Property Inspector.
7. Go to the Attributes tab. The output connector point attributes are listed. Set the expressions for the driver input connector point and the subquery filter connector point.

Note: You are required to set an expression for the subquery filter input connector point only if the subquery filter input role is set to one of the following:

IN, NOT IN, =, >, <, >=, <=, !=, <>, ^=

8. Go to the Condition tab.
9. Type an expression in the Subquery Filter Condition field. It is necessary to specify a subquery filter condition if the subquery filter input role is set to EXISTS or NOT EXISTS.
10. Select a subquery filter input role from the **Subquery Filter Input Role** drop-down list.
11. Select a group comparison condition from the **Group Comparison Condition** drop-down list. A group comparison condition can be used only with the following subquery input roles:
=, >, <, >=, <=, !=, <>, ^=
12. Save and then execute the mapping.

8.3.15 Creating Table Functions

A table function component is a projector component (see: "[Projector Components](#)" on page 8-11) that represents a table function in a mapping. Table function components enable you to manipulate a set of input rows and return another set of output rows of the same or different cardinality. The set of output rows can be queried like a physical table. A table function component can be placed anywhere in a mapping, as a source, a target, or a data flow component.

A table function component can have multiple input connector points and one output connector point. The input connector point attributes act as the input parameters for

the table function, while the output connector point attributes are used to store the return values.

For each input connector, you can define the parameter type, REF_CURSOR or SCALAR, depending on the type of attributes the input connector point will hold.

To use a table function component in a mapping:

1. Create a table function in the database if it does not exist.
2. Right-click the **Mappings** node and select **New Mapping**.
3. Drag and drop the source datastore into the logical diagram.
4. Drag and drop a table function component from the component palette into the logical diagram. A table function component is created with no input connector points and one default output connector point.
5. Click the table function component. The properties of the table function component are displayed in the Property Inspector.
6. In the property inspector, go to the Attributes tab.
7. Type the name of the table function in the **Name** field. If the table function is in a different schema, type the function name as SCHEMA_NAME.FUNCTION_NAME.
8. Go to the Connector Points tab and click the + sign to add new input connector points. Do not forget to set the appropriate parameter type for each input connector.

Note: Each REF_CURSOR attribute must be held by a separate input connector point with its parameter type set to REF_CURSOR. Multiple SCALAR attributes can be held by a single input connector point with its parameter type set to SCALAR.

9. Go to the Attributes tab and add attributes for the input connector points (created in previous step) and the output connector point. The input connector point attributes act as the input parameters for the table function, while the output connector point attributes are used to store the return values.
10. Drag and drop the required attributes from the source datastore on the appropriate attributes for the input connector points of the table function component. A connection between the source datastore and the table function component is created.
11. Drag and drop the target datastore into the logical diagram.
12. Drag and drop the output attributes of the table function component on the attributes of the target datastore.
13. Go to the physical diagram of the mapping and ensure that the table function component is in the correct execution unit. If it is not, move the table function to the correct execution unit.
14. Assign new KMs if you want to.
15. Save and then execute the mapping.

8.3.16 Creating Unpivots

An unpivot component is a projector component (see: "[Projector Components](#)" on page 8-11) that lets you transform data that is contained across attributes into multiple rows.

The unpivot component does the reverse of what the pivot component does. Similar to the pivot component, an unpivot component can be placed anywhere in the flow of a mapping.

The unpivot component is specifically useful in situations when you extract data from non-relational data sources such as a flat file, which contains data across attributes rather than rows.

8.3.16.1 Example: Unpivoting Sales Data

The external table, QUARTERLY_SALES_DATA, shown in [Table 8-4](#), contains data from a flat file. There is a row for each year and separate attributes for sales in each quarter.

Table 8-4 QUARTERLY_SALES_DATA

Year	Q1_Sales	Q2_Sales	Q3_Sales	Q4_Sales
2010	10.5	11.4	9.5	8.7
2011	9.5	10.5	10.3	7.6

[Table 8-5](#) shows a sample of the data after an unpivot operation is performed. The data that was formerly contained across multiple attributes (Q1_Sales, Q2_Sales, Q3_Sales, and Q4_Sales) is now contained in a single attribute (SALES). The unpivot component breaks the data in a single attribute (Q1_Sales) into two attributes (QUARTER and SALES). A single row in QUARTERLY_SALES_DATA corresponds to 4 rows (one for sales in each quarter) in the unpivoted data.

Table 8-5 UNPIVOTED DATA

YEAR	QUARTER	SALES
2010	Q1	10.5
2010	Q2	11.4
2010	Q3	9.5
2010	Q4	8.7
2011	Q1	9.5
2011	Q2	10.5
2011	Q3	10.3
2011	Q4	7.6

8.3.16.2 The Row Locator

The row locator is an output attribute that corresponds to the repeated set of data from the source. The unpivot component transforms a single input attribute into multiple rows and generates values for a row locator. The other attributes that correspond to the data from the source are referred as value locators. In this example, the attribute QUARTER is the row locator and the attribute SALES is the value locator.

Note: To use the unpivot component, you are required to create the row locator and the value locator attributes for the unpivot component.

The **Value Locator** field in the **Unpivot Transforms** table can be populated with an arbitrary expression. For example:

```
UNPIVOT_EMP_SALES.Q1_SALES + 100
```

8.3.16.3 Using the Unpivot Component

To use an unpivot component in a mapping:

1. Drag and drop the source data store into the logical diagram.
2. Drag and drop an unpivot component from the component palette into the logical diagram.
3. From the source datastore drag and drop the appropriate attributes on the unpivot component. In this example, the YEAR attribute.

Note: Do not drag the attributes that contain the data that corresponds to the value locator. In this example, Q1_Sales, Q2_Sales, Q3_Sales, and Q4_Sales.

4. Select the unpivot component. The properties of the unpivot component are displayed in the Property Inspector.
5. Enter a name and description for the unpivot component.
6. Create the row locator and value locator attributes using the Attribute Editor. In this example, you need to create two attributes named QUARTER and SALES.

Note: Do not forget to define the appropriate data types and constraints (if required) for the attributes.

7. In the Property Inspector, under UNPIVOT, select the row locator attribute from the **Row Locator** drop-down list. In this example, QUARTER.

Now that the row locator is selected, the other attributes can act as value locators. In this example, SALES.

8. Under UNPIVOT TRANSFORMS, click + to add transform rules for each output attribute. Edit the default values of the transform rules and specify the appropriate expressions to create the required logic.

In this example, you need to add 4 transform rules, one for each quarter. The transform rules define the values that will be populated in the row locator attribute QUARTER and the value locator attribute SALES. The QUARTER attribute must be populated with constant values (Q1, Q2, Q3, and Q4), while the SALES attribute must be populated with the values from source datastore attributes (Q1_Sales, Q2_Sales, Q3_Sales, and Q4_Sales).

9. Leave the INCLUDE NULLS check box selected to generate rows with no data for the attributes that are defined as NULL.
10. Drag and drop the target datastore into the logical diagram.

11. Connect the unpivot component to the target datastore by dragging a link from the output (right) connector of the unpivot component to the input (left) connector of the target datastore.
12. Drag and drop the appropriate attributes of the unpivot component on to the target datastore. In this example, YEAR, QUARTER, and SALES.
13. Go to the physical diagram and assign new KMs if you want to.
14. Click **Save** and then execute the mapping to perform the unpivot operation.

8.3.17 Creating Flatten Components

The flatten component is a Projector component (see: "[Projector Components](#)" on page 8-11) that processes input data with complex structure and produces a flattened representation of the same data using standard datatypes.

Flatten produces a cross product of the nested structure with the enclosing structure, so that every row of the nested structure produces a new row in the result.

The flatten component has one input connector point and one output connector point.

Example: Flatten Complex Data

[Table 8–6](#) shows an example of a datastore `movie_ratings`, which has a complex type attribute `ratings`. The complex type attribute `ratings` is repeating and has child attribute `rating`.

Table 8–6 *movie_ratings*

movie_id	year	title	ratings		
			rating	rating	rating
1	1970	Nexus	2	5	3

[Table 8–7](#) shows the resulting records after flatten.

Table 8–7 *movie_ratings*

movie_id	year	title	rating
1	1970	Nexus	2
1	1970	Nexus	5
1	1970	Nexus	3

8.3.17.1 Using a Flatten Component in a Mapping

To use a flatten component in a mapping:

1. Drag and drop the source data store into the logical diagram.
2. Drag and drop a flatten component from the component palette into the logical diagram.
3. Choose one attribute from the source component to be flattened and enter it into the property **Complex Type Attribute** of the Flatten component. This attribute should be a complex type in the source data.
4. Manually enter all attributes of the complex type in the **Attributes** properties of the Flatten component. These attributes should have no expression.
5. Map any other source attributes into the Flatten component.

6. Check the property **Include Nulls** if the complex type attribute can be null or an empty array.
7. Connect the Flatten component to a target datastore or any other downstream components.
8. Go to the physical diagram and assign new KMs if you want to.
9. Click **Save** and then execute the mapping to perform the flatten operation.

8.3.17.2 Considerations for using Flatten component with JSON Source

When you use Flatten component and your source is JSON data, you must consider the following points:

- Flatten does not support multiple child objects and nested objects within a JSON file. The source JSON data must be plain, as shown in the following example:

```
Example: {"POSTAL_AREA":1, "POSTAL_AREA_DETAIL":[{"STATE":"CA", "POSTAL_CODE":"200001"}]}
```

- When a JSON file and a flatten component is used in a mapping with Pig as the staging area, you must set the **Storage Function** option and the **Schema for Complex Fields** option for LKM File to Pig.

These options must be set as shown in the following example:

Storage Function: `JsonLoader`

Schema for Complex Fields: `POSTAL_AREA_DETAIL: { (STATE:chararray, POSTAL_CODE:chararray) }`

8.3.18 Creating Jagged Components

The jagged component is a Projector component that processes unstructured data using meta pivoting. With the jagged component, you can transform data into structured entities that can be loaded into database tables.

The jagged data component has one input group and multiple output groups, based on the configuration of the component.

The input group has two mandatory attributes: one each for name and value part of the incoming data set. A third, optional, attribute is used for row identifier sequences to delineate row sets.

To use a jagged component in a mapping:

1. Drag and drop the source data store into the logical diagram.
2. Drag and drop an jagged component from the component palette into the logical diagram.
3. The jagged input group requires two attribute level mappings; one for **name** and one for **value**.
4. Map one or more of the output groups to the downstream components.

Note: In some cases, you may not need any additional attributes other than the default group: **others**.

5. Go to the physical diagram and assign new KMs if you want to.
6. Click **Save** and then execute the mapping to perform the flatten operation.

8.4 Creating a Mapping Using a Dataset

A dataset component is a container component that allows you to group multiple data sources and join them through relationship joins. A dataset can contain the following components:

- Datastores
- Joins
- Lookups
- Filters
- Reusable Mappings: Only reusable mappings with no input signature and one output signature are allowed.

Create Joins and lookups by dragging an attribute from one datastore to another inside the dataset. A dialog is shown to select if the relationship will be a join or lookup.

Note: A driving table will have the key to look up, while the lookup table has additional information to add to the result.

In a dataset, drag an attribute from the driving table to the lookup table. An arrow will point from the driving table to the lookup table in the diagram.

By comparison, in a flow-based lookup (a lookup in a mapping that is not inside a dataset), the driving and lookup sources are determined by the order in which connections are created. The first connection is called `DRIVER_INPUT1`, the second connection `LOOKUP_INPUT1`.

Create a filter by dragging a datastore or reusable mapping attribute onto the dataset background. Joins, lookups, and filters cannot be dragged from the component palette into the dataset.

This section contains the following topics:

- [Differences Between Flow and Dataset Modeling](#)
- [Creating a Dataset in a Mapping](#)
- [Converting a Dataset to Flow-Based Mapping](#)

8.4.1 Differences Between Flow and Dataset Modeling

Datasets are container components which contain one or more source datastores, which are related using filters and joins. To other components in a mapping, a dataset is indistinguishable from any other projector component (like a datastore); the results of filters and joins inside the dataset are represented on its output port.

Within a dataset, data sources are related using relationships instead of a flow. This is displayed using an entity relationship diagram. When you switch to the physical tab of the mapping editor, datasets disappear: ODI models the physical flow of data exactly the same as if a flow diagram had been defined in the logical tab of the mapping editor.

Datasets mimic the ODI 11g way of organizing data sources, as opposed to the flow metaphor used in an ODI 12c mapping. If you import projects from ODI 11g, interfaces converted into mappings will contain datasets containing your source datastores.

When you create a new, empty mapping, you are prompted whether you would like to include an empty dataset. You can delete this empty dataset without harm, and you can always add an empty dataset to any mapping. The option to include an empty dataset is purely for your convenience.

A dataset exists only within a mapping or reusable mapping, and cannot be independently designed as a separate object.

8.4.2 Creating a Dataset in a Mapping

To create a dataset in a mapping, drag a dataset from the component palette into the logical diagram. You can then drag datastores into the dataset from the Models section of the Designer Navigator. Drag attributes from one datastore to another within a dataset to define join and lookup relationships.

Drag a connection from the dataset's output connector point to the input connector point on other components in your mapping, to integrate it into your data flow.

See Also: To create a Join or Lookup inside a Dataset, see: "[Creating a Join or Lookup](#)" on page 8-20

8.4.3 Converting a Dataset to Flow-Based Mapping

You can individually convert datasets into a flow-based mapping diagram, which is merged with the parent mapping flow diagram.

The effect of conversion of a dataset into a flow is the permanent removal of the dataset, together with the entity relationship design. It is replaced by an equivalent flow-based design. The effect of the conversion is irreversible.

To convert a dataset into a flow-based mapping:

1. Select the dataset in the mapping diagram.
2. Right click on the title and select **Convert to Flow** from the context menu.
3. A warning and confirmation dialog is displayed. Click **Yes** to perform the conversion, or click **No** to cancel the conversion.

The dataset is converted into flow-based mapping components.

8.5 Physical Design

The physical tab shows the distribution of execution among different execution units that represent physical servers. ODI computes a default physical mapping design containing execution units and groups based on the logical design, the topology of those items and any rules you have defined.

You can also customize this design by using the physical diagram. You can use the diagram to move components between execution units, or onto the diagram background, which creates a separate execution unit. Multiple execution units can be grouped into execution groups, which enable parallel execution of the contained execution units.

A mapping can have multiple physical mapping designs; they are listed in tabs under the diagram. By having multiple physical mapping designs you can create different execution strategies for the same mapping.

To create new physical mapping tabs, click the **Create New** tab.

To delete physical mapping designs, right-click on the physical mapping design tab you want to delete, and select **Delete** from the context menu.

Physical components define how a mapping is executed at runtime; they are the physical representation of logical components. Depending on the logical component a physical component might have a different set of properties.

This section contains the following topics:

- [About the Physical Mapping Diagram](#)
- [Selecting LKMs, IKMs and CKMs](#)
- [Configuring Execution Locations](#)
- [Adding Commands to be Executed Before and After a Mapping](#)
- [Configuring In-Session Parallelism](#)
- [Configuring Parallel Target Table Load](#)
- [Configuring Temporary Indexes](#)
- [Configuring Journalizing](#)
- [Configuring Extraction Options](#)
- [Creating and Managing Physical Mapping Designs](#)

8.5.1 About the Physical Mapping Diagram

In the physical diagram, the following items appear:

- **Physical Mapping Design:** The entire physical diagram represents one physical mapping design. Click the background or select the white tab with the physical mapping design label to display the physical mapping properties. By default, the staging location is colocated on the target, but you can explicitly select a different staging location to cause ODI to automatically move staging to a different host.

You can define additional physical mapping designs by clicking the small tab at the bottom of the physical diagram, next to the current physical mapping design tab. A new physical mapping design is created automatically from the logical design of the mapping.

- **Execution Groups:** Yellow boxes display groups of objects called execution units, which are executed in parallel within the same execution group. These are usually Source Groups and Target Groups:
 - **Source Execution Group(s):** Source Datastores that are within the same dataset or are located on the same physical data server are grouped in a single source execution group in the physical diagram. A source execution group represents a group of datastores that can be extracted at the same time.
 - **Target Execution Group(s):** Target Datastores that are located on the same physical data server are grouped in a single target execution group in the physical diagram. A target execution group represents a group of datastores that can be written to at the same time.
- **Execution Units:** Within the yellow execution groups are blue boxes called execution units. Execution units within a single execution group are on the same physical data server, but may be different structures.
- **Access Points:** In the target execution group, whenever the flow of data goes from one execution unit to another there is an access point (shown with a round icon).

Loading Knowledge Modules (LKMs) control how data is transferred from one execution unit to another.

An access point is created on the target side of a pair of execution units, when data moves from the source side to the target side (unless you use Execute On Hint in the logical diagram to suggest a different execution location). You cannot move an access point node to the source side. However, you can drag an access point node to the empty diagram area and a new execution unit will be created, between the original source and target execution units in the diagram.

- **Components:** mapping components such as joins, filters, and so on are also shown on the physical diagram.

You use the following knowledge modules (KMs) in the physical tab:

- **Loading Knowledge Modules (LKMs):** LKMs define how data is moved. One LKM is selected for each access point for moving data from the sources to a staging area. An LKM can be also selected to move data from a staging area not located within a target execution unit, to a target, when a single technology IKM is selected for the staging area. Select an access point to define or change its LKM in the property inspector.
- **Integration Knowledge Modules (IKMs) and Check Knowledge Modules (CKMs):** IKMs and CKMs define how data is integrated into the target. One IKM and one CKM is typically selected on a target datastore. When the staging area is different from the target, the selected IKM can be a multi-technology IKM that moves and integrates data from the staging area into the target. Select a target datastore to define or change its IKM and CKM in the property inspector.

Notes:

- Only built-in KMs, or KMs that have already been imported into the project or the global KM list, can be selected in the mapping. Make sure that you have imported the appropriate KMs in the project before proceeding.
 - For more information on the KMs and their options, refer to the KM description and to the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator*.
-
-

8.5.2 Selecting LKMs, IKMs and CKMs

ODI automatically selects knowledge modules in the physical diagram as you create your logical diagram.

Note: The **Integration Type** property of a target datastore (which can have the values `Control Append`, `Incremental Update`, or `Slowly Changing Dimension`) is referenced by ODI when it selects a KM. This property is also used to restrict the IKM selection shown, so you will only see IKMs listed that are applicable.

You can use the physical diagram to change the KMs in use.

To change the LKM in use:

1. In the physical diagram, select an access point. The Property Inspector opens for this object.

2. Select the **Loading Knowledge Module** tab, and then select a different LKM from the **Loading Knowledge Module** list.
3. KMs are set with default options that work in most use cases. You can optionally modify the KM Options.

Note: If an identically-named option exists, when switching from one KM to another KM options of the previous KM are retained. However, options that are not duplicated in the new KM are lost.

To change the IKM in use:

Note: In order to use a multi-connect IKM on the target node, you must select LKM SQL Multi-Connect, or no LKM, for the access point of that execution unit. If another LKM is selected, only mono-connect IKMs are selectable.

1. In the physical diagram, select a target datastore by clicking its title. The Property Inspector opens for this object.
2. In the Property Inspector, select the **Integration Knowledge Module** tab, and then select an IKM from the **Integration Knowledge Module** list.
3. KMs are set with default options that work in most use cases. You can optionally modify the KM Options.

Note: If an identically-named option exists, when switching from one KM to another KM options of the previous KM are retained. However, options that are not duplicated in the new KM are lost.

To change the CKM in use:

1. In the physical diagram, select a target datastore by clicking its title. The Property Inspector opens for this object.
2. In the Property Inspector, select the **Check Knowledge Module** tab, and then select a CKM from the **Check Knowledge Module** list.
3. KMs are set with default options that work in most use cases. You can optionally modify the KM Options.

Note: If an identically-named option exists, when switching from one KM to another KM options of the previous KM are retained. However, options that are not duplicated in the new KM are lost.

8.5.3 Configuring Execution Locations

In the physical tab of the mapping editor, you can change the staging area and determine where components will be executed. When you designed the mapping using components in the logical diagram, you optionally set preferred execution locations using the Execute On Hint property. In the physical diagram, ODI attempts to follow these hints where possible.

You can further manipulate execution locations in the physical tab. See the following topics for details:

- [Moving Physical Nodes](#)
- [Moving Expressions](#)
- [Defining New Execution Units](#)

8.5.3.1 Moving Physical Nodes

You can move the execution location of a physical node. Select the node and drag it from one Execution Group into another Execution Group. Or, drag it to a blank area of the physical diagram, and ODI will automatically create a new Execution Group for the component.

You can change the order of execution of certain components only. The following components can be reordered on the physical diagram:

- Expressions
- Filters
- Joins
- Lookups

Note: An inner input Connector Point of an outer join is an input whose data source contributes only matched rows to the output data set.

For example -

In ANSI SQL, 'A LEFT OUTER JOIN B' signifies that B corresponds to the inner input.

In Oracle outer join syntax, 'WHERE A.col1 = B.col2 (+)' signifies that B corresponds to the inner input.

A physical node can be reordered around an outer join only if it does not cause a change in the nodes that are connected to the inner input Connector Points of the outer join.

8.5.3.2 Moving Expressions

You can move expressions in the physical diagram. Select the Execution Unit and in the property inspector, select the **Expressions** tab. The execution location of the expression is shown in the **Execute on** property. Double-click the property to alter the execution location.

8.5.3.3 Defining New Execution Units

You can define a new execution unit by dragging a component from its current execution unit onto a blank area of the physical diagram. A new execution unit and group is created. Select the execution unit to modify its properties using the property inspector.

8.5.4 Adding Commands to be Executed Before and After a Mapping

ODI allows the addition of commands to be executed before and after a mapping. These commands can be in ODI-supported languages such as SQL, Jython, Groovy, and others. In the SQL language the Begin Mapping and End Mapping commands are executed in the same transaction as the mapping. The physical design of a mapping has the following properties to control this behavior:

Property	Description
Begin Mapping Command	Command to be executed at the beginning of the mapping.
Technology for Begin Mapping Command	Technology that this command will be executed with.
Location for Begin Mapping Command	Logical Schema that this command will be executed in.
End Mapping Command	Command to be executed at the end of the mapping.
Technology for End Mapping Command	Technology that this command will be executed with.
Location for End Mapping Command	Logical Schema that this command will be executed in.

You can view and set these properties from the Property Inspector by selecting a Physical Mapping Design.

8.5.5 Configuring In-Session Parallelism

ODI agent is the scheduler that runs an entire ODI mapping job on a given host. If you have two or more loads, it will either run them one after another (serialized), or simultaneously (parallelized, using separate processor threads).

Execution units in the same execution group are parallelized. If you move an execution unit into its own group, it is no longer parallelized with other execution units: it is now serialized. The system will select the order in which separate execution groups are run.

You might choose to run loads serially to reduce instantaneous system resource usage, while you might choose to run loads in parallel to reduce the longevity of system resource usage.

8.5.6 Configuring Parallel Target Table Load

You can enable parallel target table loading in a physical mapping design. Select the physical mapping design (by clicking on the tab at the bottom of the physical diagram, or clicking an empty area of the diagram) and in the property inspector, check the box for the property **Use Unique Temporary Object Names**.

This option allows multiple instances of the same mapping to be executed concurrently. To load data from source to staging area, C\$ tables are created in the staging database.

Note: In ODI 11g, C\$ table names were derived from the target table of the interface. As a result, when multiple instances of the same mapping were executed at the same time, data from different sessions could load into the same C\$ table and cause conflicts.

In ODI 12c, if the option **Use Unique Temporary Object Names** is set to true, the system generates a globally-unique name for C\$ tables for each mapping execution. This prevents any conflict from occurring.

8.5.7 Configuring Temporary Indexes

If you want ODI to automatically generate a temporary index to optimize the execution of a filter, join, or datastore, select the node in the physical diagram. In the property inspector, select the **Temporary Indexes** tab. You can double-click the **Index Type** field to select a temporary index type.

Note: The creation of temporary indexes may be a time consuming operation in the overall flow. Oracle recommends reviewing execution statistics and comparing the execution time saved by the indexes to the time spent creating them.

8.5.8 Configuring Journalizing

A source datastore can be configured in the physical diagram to use journalized data only. This is done by enabling **Journalized Data Only** in the **General** properties of a source datastore. The check box is only available if the referenced datastore is added to CDC in the model navigator.

Only one datastore per mapping can have journalizing enabled.

For more information about journalizing, see [Chapter 4, "Using Journalizing."](#)

8.5.9 Configuring Extraction Options

Each component in the physical diagram, excluding access points and target datastores, has an **Extraction Options** tab in the property inspector. Extraction options influence the way that SQL is generated for the given component. Most components have an empty list of extraction options, meaning that no further configuration of the SQL generation is supported.

Extraction options are driven by the Extract Knowledge Module (XKM) selected in the **Advanced** sub-tab of the **Extract Options** tab. XKMs are part of ODI and cannot be created or modified by the user.

8.5.10 Creating and Managing Physical Mapping Designs

The entire physical diagram represents one physical mapping design. Click the background or select the white tab with the physical mapping design label to display the physical mapping properties for the displayed physical mapping design.

You can define additional physical mapping designs by clicking the small tab at the bottom of the physical diagram, next to the current physical mapping design tab(s). A new physical mapping design is created automatically, generated from the logical design of the mapping. You can modify this physical mapping design, and save it as part of the mapping.

For example, you could use one physical mapping design for your initial load, and another physical mapping design for incremental load using changed data capture (CDC). The two physical mapping designs would have different journalizing and knowledge module settings.

As another example, you could use different optimization contexts for each physical mapping design. Each optimization context represents a slightly different users' topology. One optimization context can represent a development environment, and another context represents a testing environment. You could select different KMs appropriate for these two different topologies.

8.6 Reusable Mappings

Reusable mappings allow you to encapsulate a multi-step integration (or portion of an integration) into a single component, which you can save and use just as any other components in your mappings. Reusable mappings are a convenient way to avoid the labor of creating a similar or identical subroutine of data manipulation that you will use many times in your mappings.

For example, you could load data from two tables in a join component, pass it through a filter component, and then a distinct component, and then output to a target datastore. You could then save this procedure as a reusable mapping, and place it into future mappings that you create or modify.

After you place a reusable mapping component in a mapping, you can select it and make modifications to it that only affect the current mapping.

Reusable mappings consist of the following:

- **Input Signature and Output Signature components:** These components describe the attributes that will be used to map into and out of the reusable mapping. When the reusable mapping is used in a mapping, these are the attributes that can be matched by other mapping components.
- **Regular mapping components:** Reusable mappings can include all of the regular mapping components, including datastores, projector components, and selector components. You can use these exactly as in regular mappings, creating a logical flow.

By combining regular mapping components with signature components, you can create a reusable mapping intended to serve as a data source, as a data target, or as an intermediate step in a mapping flow. When you work on a regular mapping, you can use a reusable mapping as if it were a single component.

8.6.1 Creating a Reusable Mapping

You can create a reusable mapping within a project, or as a global object. To create a reusable mapping, perform the following steps:

1. From the designer navigator:
Open a project, right-click Reusable Mappings, and select New Reusable Mapping.
Or, expand the Global Objects tree, right click Global Reusable Mappings, and select New Reusable Mapping.
2. Enter a name and, optionally, a description for the new reusable mapping.
Optionally, select Create Default Input Signature and/or Create Default Output Signature. These options add empty input and output signatures to your reusable mapping; you can add or remove input and output signatures later while editing your reusable mapping.

Note: In order to make use of these signatures, you will need to connect them to your reusable mapping flow.

3. Drag components from the component palette into the reusable mapping diagram, and drag datastores and other reusable mappings from the designer navigator, to assemble your reusable mapping logic. Follow all of the same processes as for creating a normal mapping.

Note: When you have a reusable mapping open for editing, the component palette contains the Input Signature and Output Signature components in addition to the regular mapping components.

4. Validate your reusable mapping by clicking the **Validate the Mapping** button (a green check mark icon). Any errors will be displayed in a new error pane.

When you are finished creating your reusable mapping, click **File** and select **Save**, or click the **Save** button, to save your reusable mapping. You can now use your reusable mapping in your mapping projects.

8.7 Editing Mappings Using the Property Inspector and the Structure Panel

You can use the Property Inspector with the Structure Panel to perform the same actions as on the logical and physical diagrams of the mapping editor, in a non-graphical form.

Using the Structure Panel

When creating and editing mappings without using the logical and physical diagrams, you will need to open the Structure Panel. The Structure Panel provides an expandable tree view of a mapping, which you can traverse using the tab keys, allowing you to select the components of your mapping. When you select a component or attribute in the Structure Panel, its properties are shown in the Property Inspector exactly the same as if you had selected the component in the logical or physical diagram.

The Structure Panel is useful for accessibility requirements, such as when using a screen reader.

To open the structure panel, select **Window** from the main menu and then click **Structure**. You can also open the Structure Panel using the hotkey **Ctrl+Shift-S**.

This section contains the following topics:

- [Adding and Removing Components](#)
- [Editing a Component](#)
- [Customizing Tables](#)
- [Using Keyboard Navigation for Common Tasks](#)

8.7.1 Adding and Removing Components

With the Property Inspector, the Component Palette, and the Structure Panel, you can add or remove components of a mapping.

8.7.1.1 Adding Components

To add a component to a mapping with the Component Palette and the Structure Panel:

1. With the mapping open in the Mapping Editor, open the Component Palette.
2. Select the desired component using the Tab key, and hit Enter to add the selected component to the mapping diagram and the Structure Panel.

8.7.1.2 Removing Components

To remove a component with the Structure Panel:

1. In the Structure Panel, select the component you want to remove.
2. While holding down Ctrl+Shift, hit Tab to open a pop-up dialog. Keep holding down Ctrl+Shift, and use the arrow keys to navigate to the left column and select the mapping. You can then use the right arrow key to select the logical or physical diagram. Release the Ctrl+Shift keys after you select the logical diagram.

Alternatively, select **Windows > Documents...** from the main menu bar. Select the mapping from the list of document windows, and click **Switch to Document**.

3. The component you selected in the Structure Panel in step 1 is now highlighted in the mapping diagram. Hit Delete to delete the component. A dialog box confirms the deletion.

8.7.2 Editing a Component

To edit a component of a mapping using the Structure Panel and the Property Inspector:

1. In the Structure Panel, select a component. The component's properties are shown in the Property Inspector.
2. In the Property Inspector, modify properties as needed. Use the Attributes tab to add or remove attributes. Use the Connector Points tab to add connections to other components in your mapping.
3. Expand any component in the Structure Panel to list individual attributes. You can then select individual attributes to show their properties in the Property Inspector.

8.7.3 Customizing Tables

There are two ways to customize the tables in the Property Inspector to affect which columns are shown. In each case, open the Structure Panel and select a component to display its properties in the Property Inspector. Then, select a tab containing a table and use one of the following methods:

- From the table toolbar, click the **Select Columns...** icon (on the top right corner of the table) and then, from the drop down menu, select the columns to display in the table. Currently displayed columns are marked with a check mark.
- Use the Customize Table Dialog:
 1. From the table toolbar, click **Select Columns...**
 2. From the drop down menu, select **Select Columns...**
 3. In the **Customize Table Dialog**, select the columns to display in the table.
 4. Click **OK**.

8.7.4 Using Keyboard Navigation for Common Tasks

This section describes the keyboard navigation in the Property Inspector.

[Table 8–8](#) shows the common tasks and the keyboard navigation used in the Property Inspector.

Table 8–8 Keyboard Navigation for Common Tasks

Navigation	Task
Arrow keys	Navigate: move one cell up, down, left, or right
TAB	Move to next cell
SHIFT+TAB	Move to previous cell
SPACEBAR	Start editing a text, display items of a list, or change value of a checkbox
CTRL+C	Copy the selection
CTRL+V	Paste the selection
ESC	Cancel an entry in the cell
ENTER	Complete a cell entry and move to the next cell or activate a button
DELETE	Clear the content of the selection (for text fields only)
BACKSPACE	Delete the content of the selection or delete the preceding character in the active cell (for text fields only)
HOME	Move to the first cell of the row
END	Move to the last cell of the row
PAGE UP	Move up to the first cell of the column
PAGE DOWN	Move down to the last cell of the column

8.8 Flow Control and Static Control

In a mapping, it is possible to set two points of control. Flow Control checks the data in the incoming flow before it gets integrated into a target, and Static Control checks constraints on the target datastore after integration.

IKMs can have options to run `FLOW_CONTROL` and to run `STATIC_CONTROL`. If you want to enable either of these you must set the option in the IKM, which is a property set on the target datastore. In the physical diagram, select the datastore, and select the **Integration Knowledge Module** tab in the property inspector. If flow control options are available, they are listed in the Options table. Double-click an option to change it.

Notes:

- Flow control is not supported for component KMs like IKM Oracle Insert. For more information, see "Knowledge Modules" in *Connectivity and Knowledge Modules Guide for Oracle Data Integrator*. The description of each IKM indicates if it supports flow control.
 - In ODI 11g the CKM to be used when flow or static control is invoked was defined on the interface. ODI 12c supports multiple targets on different technologies within the same mapping, so the CKM is now defined on each target datastore
-
-

This section contains the following topics:

- [Setting up Flow Control](#)
- [Setting up Static Control](#)
- [Defining the Update Key](#)

8.8.1 Setting up Flow Control

The flow control strategy defines how data is checked against the constraints defined on a target datastore before being integrated into this datastore. It is defined by a Check Knowledge Module (CKM). The CKM can be selected on the target datastore physical node. The constraints that checked by a CKM are specified in the properties of the datastore component on the logical tab.

To define the CKM used in a mapping, see: "[Selecting LKMs, IKMs and CKMs](#)" on page 8-35.

8.8.2 Setting up Static Control

The post-integration control strategy defines how data is checked against the constraints defined on the target datastore. This check takes place once the data is integrated into the target datastore. It is defined by a CKM. In order to have the post-integration control running, you must set the `STATIC_CONTROL` option in the IKM to `true`. Post-integration control requires that a primary key is defined in the data model for the target datastore of your mapping.

The settings **Maximum Number of Errors Allowed** and **Integration Errors as Percentage** can be set on the target datastore component. Select the datastore in the logical diagram, and in the property inspector, select the **Target** tab.

Post-integration control uses the same CKM as flow control.

8.8.3 Defining the Update Key

If you want to use update or flow control features in your mapping, it is necessary to define an update key on the target datastore.

The update key of a target datastore component contains one or more attributes. It can be the unique key of the datastore that it is bound to, or a group of attributes that are marked as the key attribute. The update key identifies each record to update or check before insertion into the target.

To define the update key from a unique key:

1. In the mapping diagram, select the header of a target datastore component. The component's properties will be displayed in the Property Inspector.
2. In the **Target** properties, select an **Update Key** from the drop down list.

Notes:

- The Target properties are only shown for datastores which are the target of incoming data. If you do not see the Target properties, your datastore does not have an incoming connection defined.
 - Only unique keys defined in the model for this datastore appear in this list.
-
-

You can also define an update key from the attributes if:

- You don't have a unique key on your datastore.
- You want to specify the key regardless of already defined keys.

When you define an update key from the attributes, you select manually individual attributes to be part of the update key.

To define the update key from the attributes:

1. Unselect the update key, if it is selected.
2. In the Target Datastore panel, select one of the attributes that is part of the update key to display the Property Inspector.
3. In the Property Inspector, under **Target** properties, check the **Key** box. A key symbol appears in front of the key attribute(s) in the datastore component displayed in the mapping editor logical diagram.
4. Repeat the operation for each attribute that is part of the update key.

8.9 Designing E-LT and ETL-Style Mappings

See Also: E-LT and ETL are defined and described in "What is E-LT" in *Understanding Oracle Data Integrator*.

In an E-LT-style integration mapping, ODI processes the data in a staging area, which is located on the target. Staging area and target are located on the same RDBMS. The data is loaded from the source(s) to the target. To create an E-LT-style integration mapping, follow the standard procedure described in "Creating a Mapping" on page 8-5.

In an ETL-style mapping, ODI processes the data in a staging area, which is different from the target. The data is first extracted from the source(s) and then loaded to the staging area. The data transformations take place in the staging area and the intermediate results are stored in temporary tables in the staging area. The data loading and transformation tasks are performed with the standard ELT KMs.

Oracle Data Integrator provides two ways for loading the data from the staging area to the target:

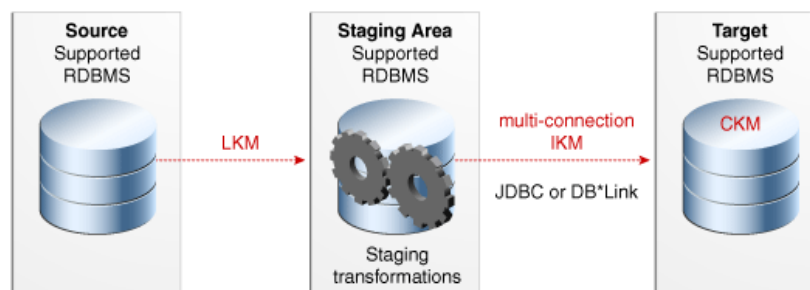
- [Using a Multi-connection IKM](#)
- [Using an LKM and a mono-connection IKM](#)

Depending on the KM strategy that is used, flow and static control are supported. See "Designing an ETL-Style Mapping" in the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator* for more information.

Using a Multi-connection IKM

A multi-connection IKM allows updating a target where the staging area and sources are on different data servers. [Figure 8-3](#) shows the configuration of an integration mapping using a multi-connection IKM to update the target data.

Figure 8-3 ETL-Mapping with Multi-connection IKM



See the chapter in the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator* that corresponds to the technology of your staging area for more information on when to use a multi-connection IKM.

To use a multi-connection IKM in an ETL-style mapping:

1. Create a mapping using the standard procedure as described in "[Creating a Mapping](#)" on page 8-5. This section describes only the ETL-style specific steps.
2. In the Physical tab of the Mapping Editor, select a physical mapping design by clicking the desired physical mapping design tab and clicking on the diagram background. In the property inspector, the field **Preset Staging Location** defines the staging location. The empty entry specifies the target schema as staging location. Select a different schema as a staging location other than the target.
3. Select an Access Point component in the physical schema and go to the property inspector. For more information about Access Points, see: "[About the Physical Mapping Diagram](#)" on page 8-34.
4. Select an LKM from the LKM Selector list to load from the source(s) to the staging area. See the chapter in the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator* that corresponds to the technology of your staging area to determine the LKM you can use.
5. Optionally, modify the KM options.
6. In the Physical diagram, select a target datastore. The property inspector opens for this target object.

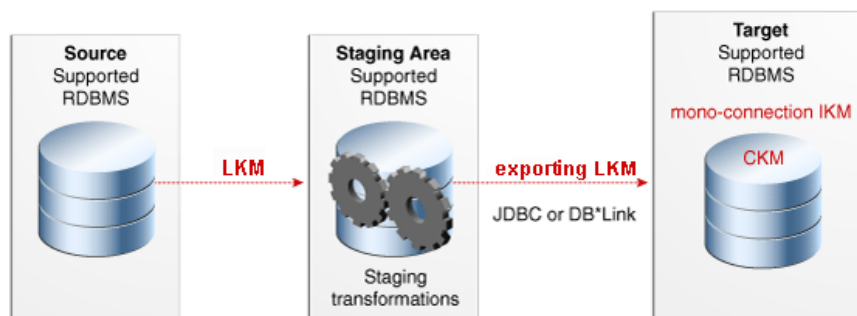
In the Property Inspector, select an ETL multi-connection IKM from the IKM Selector list to load the data from the staging area to the target. See the chapter in the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator* that corresponds to the technology of your staging area to determine the IKM you can use.

7. Optionally, modify the KM options.

Using an LKM and a mono-connection IKM

If there is no dedicated multi-connection IKM, use a standard exporting LKM in combination with a standard mono-connection IKM. [Figure 8-4](#) shows the configuration of an integration mapping using an exporting LKM and a mono-connection IKM to update the target data. The exporting LKM is used to load the flow table from the staging area to the target. The mono-connection IKM is used to integrate the data flow into the target table.

Figure 8-4 ETL-Mapping with an LKM and a Mono-connection IKM



Note that this configuration (LKM + exporting LKM + mono-connection IKM) has the following limitations:

- Neither simple CDC nor consistent CDC are supported when the source is on the same data server as the staging area (explicitly chosen in the Mapping Editor)
- Temporary Indexes are not supported

See the chapter in the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator* that corresponds to the technology of your staging area for more information on when to use the combination of a standard LKM and a mono-connection IKM.

To use an LKM and a mono-connection IKM in an ETL-style mapping:

1. Create a mapping using the standard procedure as described in "[Creating a Mapping](#)" on page 8-5. This section describes only the ETL-style specific steps.
2. In the Physical tab of the Mapping Editor, select a physical mapping design by clicking the desired physical mapping design tab and clicking on the diagram background. In the property inspector, the field **Preset Staging Location** defines the staging location. The empty entry specifies the target schema as staging location. Select a different schema as a staging location other than the target.
3. Select an Access Point component in the physical schema and go to the property inspector. For more information about Access Points, see: "[About the Physical Mapping Diagram](#)" on page 8-34.
4. In the Property Inspector, in the **Loading Knowledge Module** tab, select an LKM from the **Loading Knowledge Module** drop-down list to load from the source(s) to the staging area. See the chapter in the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator* that corresponds to the technology of your staging area to determine the LKM you can use.
5. Optionally, modify the KM options. Double-click a cell in the **Value** column of the options table to change the value.
6. Select the access point node of a target execution unit. In the Property Inspector, in the **Loading Knowledge Module** tab, select an LKM from the **Loading Knowledge Module** drop-down list to load from the staging area to the target. See the chapter in the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator* that corresponds to the technology of your staging area to determine the LKM you can use.
7. Optionally, modify the options.
8. Select the Target by clicking its title. The Property Inspector opens for this object.
In the Property Inspector, in the **Integration Knowledge Module** tab, select a standard mono-connection IKM from the **Integration Knowledge Module** drop-down list to update the target. See the chapter in the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator* that corresponds to the technology of your staging area to determine the IKM you can use.
9. Optionally, modify the KM options.

Creating and Using Dimensions and Cubes

This chapter describes how to create and use dimensional objects in ODI. Dimensional objects are logical structures to help identify and categorize data. Oracle Data Integrator enables you to design, create, edit and load two types of dimensional objects - dimensions and cubes.

Note: The Dimensions and Cubes feature only support Oracle Technology.

This chapter includes the following topics in detail:

- [Overview of Dimensional Objects](#)
- [Creating Dimensional Objects through ODI](#)
- [Using Dimensional Components in Mappings](#)
- [Expanding Dimensional Components](#)

9.1 Overview of Dimensional Objects

Dimensional objects are logical construct that are used to create a model representing the logical design of a data warehouse. The physical design and implementation of the dimensional model will translate the logical design into database via SQL statements. This section provides a general overview on dimension and cube objects along with their physical implementation.

For more information on Data Warehousing, refer to <https://docs.oracle.com/database/121/DWHSG/toc.htm>

9.1.1 Overview of Dimensions

A dimension is a structure that organizes data. For example, a products dimension organizes data about products including product information, product categories and its sub-categories. A dimension consists of a set of levels and a set of hierarchies defined over these levels.

To create a dimension, you must define the following:

- Levels
- Level Attributes
- Hierarchies

For example, the products dimension can have levels category and subcategory. It can have hierarchies to help drill from product to sub-category or to category.

Using dimensions improves query performance as users often analyze data by drilling down on known hierarchies. An example of a hierarchy is the Time hierarchy of year, quarter, month, day.

1. Level

Level represents a collection of dimension values that share similar characteristics. For example, there can be a State level that has state name, state population and state capital.

2. Level Attribute

Level Attribute includes one surrogate and business identifiers for levels. A level attribute is a descriptive characteristic of a level value. For example, an ITEM level can have an attribute called COLOR. For example, item1 has value green as COLOR and item2 has value blue as COLOR. Attributes represent logical groupings that enable end users to select data based on like characteristics.

Some level attributes are natural keys or surrogate identifiers. A Natural key uniquely identifies the record from the source system. It can be composed of composite keys.

A surrogate identifier uniquely identifies each level record across all the levels of the dimension. It must be composed of a single attribute. Surrogate identifiers enable you to hook facts to any dimension level as opposed to the lowest dimension level only. You must use a surrogate key if:

- your dimension is a Type 2 slowly changing dimension (SCD). In these cases, you can have multiple dimension records loaded for each Natural key value, so you need an extra unique key to track these records
- your dimension contains more than one level and is implemented using a star schema. Thus, any cube that references such a dimension references multiple dimension level.

If no surrogate key is defined, then only the leaf-level dimension records are saved in the dimension table, the parent level information is stored in extra columns in the leaf-level records. But there is no unique way to reference the upper level in that case.

You do not need a surrogate key for any Type 1 dimensions, implemented by star, where only the leaf level(s) are referenced by a cube.

3. Hierarchy

A structure that uses ordered levels as a means of organizing data. A level hierarchy defines hierarchical relationships between adjacent levels. A hierarchical relationship is a functional dependency from one level of a hierarchy to a more abstract level in the hierarchy.

A hierarchy can be used to define data aggregation; for example in a time dimension. A hierarchy might be used to aggregate data from the "Months" level to the "Quarter" level to the "Year" level.

4. Dimension Role

A dimension can perform multiple dimension roles in a data warehouse.

Within a data warehouse, a cube can refer to the same dimension multiple times. For each such reference, the dimension performs a different dimension role in the cube.

For example, in a wholesale company, each sales record can have three time values:

- First time value records when the order is received
- Second time value records when the product is shipped
- Third time value records when the payment is received.

Different departments in the wholesales company are interested to summarize the sales data in different ways:- by order time, by product shipment time and by product payment time.

To model such scenario, the warehouse designer has the following choices:

- Model and populate three time dimensions separately and let the wholesales cube refer to each time dimension.
- Model one time dimension. Create three roles for the time dimension. First one is "order time". Second one is "ship time". Third one is "payment time". Let the sales cube refer to "order time", "ship time" and "payment time".

The second choice has the advantage of storing the time data only once.

5. Overview of Slowly Changing Dimensions

During loading of a dimension, you may require to preserve existing data when new data comes in. For example, let's say that the warehouse records initially that a city has population 5000. Now new data comes in which indicate the city has a new population of 6000. Instead of simply overriding the old population with the new number, you may need to retain the old population somewhere in the warehouse so that you can compare the two populations at some point of time.

A Slowly Changing Dimension (SCD) is a dimension that stores and manages both current and historical data over time in a data warehouse. The strategy of how to keep historical data is called slowly changing dimension strategy. In data warehousing, there are three commonly recognized types of SCDs. They are: Type 1, Type 2, and Type 3.

Types of Slow Changing Dimensions

- **Type I** slowly changing dimension doesn't store any history. In a Type 1 Slowly Changing Dimension (SCD), the new data overwrites the existing data. Typically, this type is not considered an SCD and most dimensions are of this type. Thus the historic data is lost as it is not stored else where. This is the default type of dimension you create. You need not specify any additional information to create a Type 1 SCD, unless there are specific business reasons, you must assume that a Type 1 SCD is sufficient.

An attribute can play only one of these roles. For example, an attribute cannot be a regular attribute and an effective date attribute.

- **Type II** slowly changing dimension stores all versions of histories. They store the entire history of values of every attribute and every parent level relationship that is marked as SCD2 Trigger. In the population example, a type two slowly changing dimension stores all population numbers that each city can ever hold.

Define the following parameters, to define a type two SCD:

- For the level attribute that will trigger the SCD type 2, you have to select the settings as **Trigger History**. You can choose one or more level attributes so that changes in the value of chosen attributes will trigger a version of history to be saved.

- To use SCD Type 2, Surrogate Keys attributes should be created for all the levels. Surrogate Key attribute should not be set as SCD2 Trigger History.
- Level that has any of attributes defined as Type 2 Trigger History must have date/timestamp level attribute set as Type 2 Setting **Start Date**.
- Level that has any of attributes defined as Type 2 Trigger History must have date/timestamp level attribute set as Type 2 Setting **End Date**.
- A SCD type 2 can also be triggered by the Parent Level reference member, if it has been set to **Trigger History**.

To create a Type 2 SCD or a Type 3 SCD, in addition to the regular dimension attributes, you need additional attributes that perform the following roles:

- **Trigger History:** These are attributes for which historical values must be stored. For example, in the PRODUCTS dimension, the attribute PACKAGE_TYPE of the Product level can be a triggering attribute. When the value of the attribute changes, the old value must be stored.
 - **Start Date:** This attribute stores the start date of the record's life span.
 - **End Date:** This attribute stores the end date of the record's life span.
 - **Previous Attribute:** For Type 3 SCDs only, this attribute stores the previous value of a attribute, that has a version.
- **Type III** slowly changing dimension store two versions of values for the chosen level of attributes, that is the current value and the previous value. In the population example, a type three slowly changing dimension will keep the current population of each city as well as the previous population.

Define the following parameters, to define a type three SCD:

- For each level attribute, that you want to store previous value, you need to create and assign another level attribute as Type 3 Previous Attribute. Also, date/timestamp level attribute must be created and assigned to level attribute as Type 3 Start Date.
- There are two restrictions on which attribute can have a previous value. They are:
 1. Previous value attribute cannot have further previous values. For example, once you indicate an attribute as "old_email" as the previous value of "email", they cannot choose yet another attribute "previous-previous email" to be used as the previous value of "old_email".
 2. The Surrogate Key attribute cannot have previous values.

6. Dimension Implementation

Along with defining logical structure of dimension, such as its levels, attributes and hierarchies, you need to specify how the dimension data is physically stored.

Note: ODI supports only Star schema for the current release.

In a star schema, the data of each dimension is stored on a single datastore.

We call this specification of data storage of dimension as dimension implementation.

9.1.2 Overview of Cubes

A cube is a set of measures grouped together that have similar dimensionality. The axes of the cube contain dimension values and the body of the cube contains measure values.

For example, sales data can be organized into a cube, whose edges contain values from the Time, Product, and Customer dimensions and whose body contains values from the Volume Sales, and Dollar Sales measures.

Each cube must hold data related to one or more dimensions. Dimensions can be shared across cubes. The same dimension may be reused under different dimension roles by the same cube. Each cube must have a **fact table**.

9.1.2.1 Understanding Measure (Fact)

A measure is data, usually numeric and additive, that can be examined and analyzed. Examples include Sales, Cost and Profit. Fact and measure are synonymous; Fact is more commonly used with relational environments whereas Measure is more commonly used with multidimensional environments.

9.1.2.2 Cube Implementation

After specifying the logical structure of a cube, such as measures and dimension references, it is necessary for users to specify how the cube will be physically stored. Typically a cube is stored on a single table called fact table. Each measure usually corresponds to one column and each dimension reference corresponds to a column on the fact table and optionally a foreign key from fact table to dimension table.

9.2 Creating Dimensional Objects through ODI

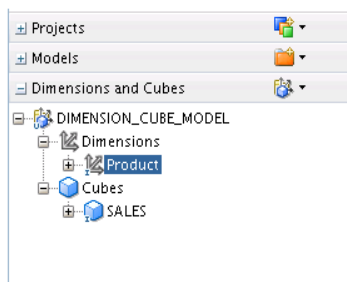
In ODI, a new type of model called Dimension and Cube Model is added to support creation of dimension and cube objects. The dimensional objects can be used in mapping for creating the physical data warehouse.

This section elaborates on the following:

- [Dimension and Cube Accordion](#)
- [Using Dimensions in ODI](#)
- [Using Cubes in ODI](#)
- [Creating New Dimensional Models](#)
- [Creating and Editing Dimensional Objects using the Editor](#)

9.2.1 Dimension and Cube Accordion

Dimensional objects are added as a separate accordion in the Navigator tab. A dimension model serves as a folder to contain dimension and cube objects as shown below:



9.2.2 Using Dimensions in ODI

This section elaborates on the following:

Generic Properties

- Dimension is modeled separately from the underlying datastore objects.
- It supports type one, type two and type three slowly changing dimensions
- It supports dimension roles
- Supports multiple hierarchies

To define a dimension, you need to specify its levels and hierarchies.

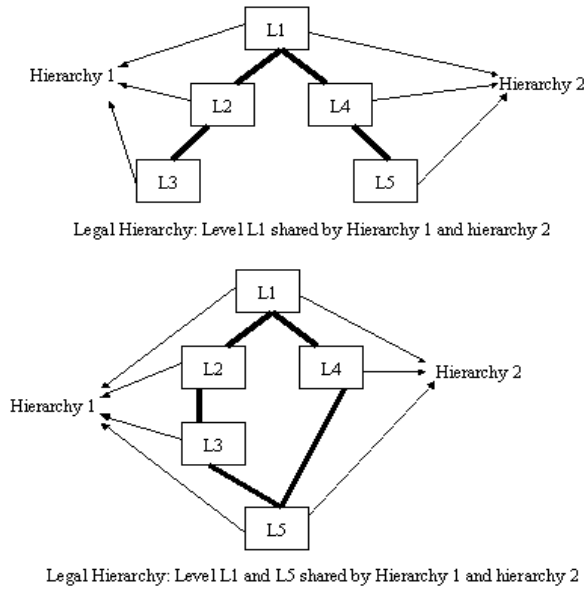
1. Level

- A level may determine one or more dependent attributes.
- All columns of a level must come from the same table.
- Levels could be shared across hierarchies.
- Levels must be ordered within a hierarchy
- A level can uniquely determine some attributes. These attribute columns must come from the same table as level columns.
- The columns of a hierarchy level may not be associated with more than one dimension.
- No two levels may have an identical set of columns. The columns of each hierarchy level are not null.
- Each level may appear in a hierarchy at most once.

2. Hierarchy

- A dimension must have one or more hierarchies.
- Each hierarchy must have one or more levels.
- Levels of a dimension may be shared across hierarchies. The hierarchies of a dimension may overlap or be disconnected from each other. The following diagrams illustrate the kind of hierarchies ODI will be able to support.

Figure 9–1 Hierarchies Supported by ODI



- All hierarchies must strictly have 1:n relationships. Considering two adjacent levels in a hierarchy, one record in a parent level corresponds to multiple records in a child level within a hierarchy. Further, a record in a child level can correspond to only one parent record within a hierarchy.
- Ability to skip levels within a hierarchy.
- It supports the selection of a default hierarchy.
- No support for inverted hierarchy.

Table 9–1 Examples of Inverted Hierarchy

Hierarchy One	Hierarchy Two
Total US	Total US
Region	District
District	Region
Account	Account

In hierarchy one, Districts roll into regions, and in hierarchy two regions roll into Districts.

9.2.3 Using Cubes in ODI

This section elaborates on the following:

9.2.3.1 Generic Properties

- Each cube must have one and only one fact table. A cube contains a list of measures.
- The fact table contains measure columns.
- Each cube must be defined by a set of dimensions.
- A cube may reuse the same dimension multiple times under different aliases.

- A cube can refer to a non-lowest level in a dimension.
- A fact table may contain multiple measures.
- The 1:n relationships from the fact tables to the dimension tables must be enforced.

9.2.3.2 Cube Measures

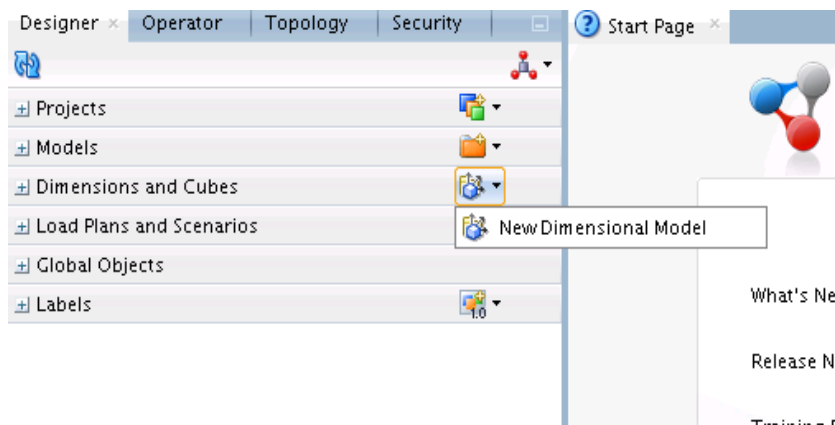
Please note:

- Only simple measures are supported. That is, a measure is mapped to a single column of a fact table.
- Measures need be explicitly defined.

9.2.4 Creating New Dimensional Models

To create a new dimensional model,

- In the **Designer** tab, click the **New Dimensional Model** icon, present beside the **Dimensions and Cubes** node, as show below:

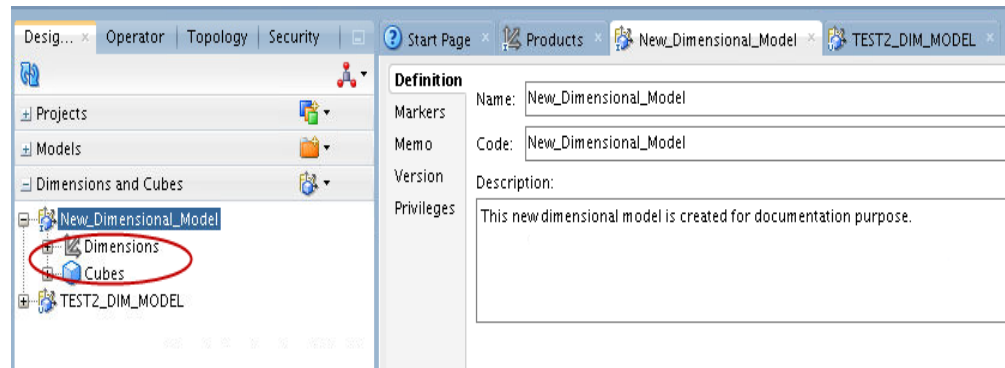


Definition tab appears, allowing you to configure the following details of the newly created dimensional model:

1. **Name** - Enter the name of the newly created dimensional model
2. **Code**- Enter the code of the newly created dimensional model
3. **Description** - Enter a description for the newly created dimensional model

Click the **Save** icon.

A new dimensional model folder with the specified name is created with two nodes - Dimensions and Cubes, as shown below:



9.2.5 Creating and Editing Dimensional Objects using the Editor

To create new dimensional objects,

- Right-click the **Dimensions** node and select **New Dimension** option, to create a new dimension object
- Right-click the **Cubes** node and select **New Cube** option, to create a new cube object

New dimensional objects are created.

To edit the properties of existing dimensional objects,

- Double click the required dimension or cube object that you wish to edit or else right-click and select Open.

The selected dimensional object opens in the Dimension or Cube editor. The Dimension or Cube Editor gives you full control over all the aspects of dimension or cube definition and implementation. This provides maximum flexibility to perform various operations related to the created dimensional objects.

9.2.5.1 Using Dimension Editor

Dimension Editor has three essential tabs for easy and effective administration of a dimension object. They are:

- [Definition Tab](#)
- [Levels Tab](#)
- [Hierarchies Tab](#)

9.2.5.1.1 Definition Tab Definition tab of the Dimension Editor allows you to define the following properties of a newly created or existing dimension:

1. **Name**- It represents the name of the created dimension.
In the **Name** text box, enter the required name of the created dimension.
2. **Description**- It represents a short description of the created dimension.
In the **Description** text box, enter a short description for the created dimension.
3. **Pattern Name** - It denotes the type of pattern that has to be applied to the created dimension.

Note: ODI has a default built-in pattern called Dimension Pattern.

4. **Binding Datastore** - It denotes the datastores to which the created dimension is bound to.
5. **Surrogate Key Sequence** - represents a sequence from a project tree (where dimension mappings are created), that is used to generate surrogate keys for the dimension if Surrogate Keys are used.

Note: Project ODI Sequence of type **Native Sequence** must be created in the same logical schema where dimension datastore is present.

9.2.5.1.2 Levels Tab Levels tab of the Dimension Editor allows you to define the levels and level attributes for each level of a newly created or existing dimension. It contains the following tables:

- Levels table
- Levels Attributes table
- Parent Level References

1. Levels table

The list of levels available for a dimension are displayed in the **Levels** table, present at the top of the Levels tab. You can create levels from top to bottom and by this it becomes easier to define level relationships.

For creating or removing levels click the Add + or cross x icons present in the header of the Levels table.

This Levels table contains the following parameters:

- **Name** - It represents the name of the created level.
In the **Name** text box, enter the name of the newly created level.
- **Description** - It represents a short description for the level creation.
In the **Description** text box, enter a short description for the newly created level.
- **Binding Datastore** - It represents the datastore that is used to store the dimension data.
- **Staging Datastore** - It represents the datastore that is used to stage the incoming data for this level before loading them into the actual binding datastore.

Please note, Binding and Staging Datastores need to already exist and they should be present in the same data model. Along with these, the bound datastore must also be created in the database. The staging tables are created and truncated during the execution of mapping in which the dimensions are used. These datastores should have all the attributes to bind to Levels attributes, Natural Key Members attributes and Parent Level reference attributes for all levels.

2. Level Attributes Table

This table displays all the level attributes of the level that is selected in the Levels table.

For creating or removing level attributes click the Add + or cross x icons and to reorder the level, click the **up or down arrow** icons, present in the header of the Level Attributes table.

This Levels table contains the following parameters:

- **Name** - It represents the name of the created level attribute.

In the **Name** text box, enter the name of the created level attribute.
- **Is Surrogate Key** - It represents the nature of the selected attribute i.e.whether it can be treated as the surrogate key or not.

Click the **Is Surrogate Key** check box, to use the selected attribute as a surrogate key.
- **Description** - It represents a short description for the newly created level attribute.

In the **Description** text box, enter a short description for the newly created level attribute.
- **Data Type** - It represents the data type of the newly created level attribute.

From the **Data Type** drop-down box, select the required data type for the newly created level attribute. The values of the **Data Type** drop-down box are the data types in the "Generic SQL" technology defined in ODI repository.
- **SCD2 Setting** - It contains the following values:

 - **None** - Select this option, if you do not wish to select any of the below listed attributes.
 - **Start Date** - This attribute stores the start date of the record's life span.
 - **End Date** - This attribute stores the end date of the record's life span.
 - **Trigger History** - These are attributes for which historical values must be stored. For example, in the PRODUCTS dimension, the attribute PACKAGE_TYPE of the Product level can be a triggering attribute. When the value of the attribute changes, the old value must be stored. Select this option for an attribute, if the attribute should be versioned.

From the **SCD2 Setting** drop-down box, select the required value.
- **SCD 3 Previous Attribute** -It represents the list of level attributes for the current level. If this parameter is set, then SCD3 enabled is for the newly created level attribute. When the incoming data contains any change to this attribute, its value is first moved to the SCD3 previous attribute before being overridden by the incoming data.
- **SCD 3 Effective Date** - This drop-down box lists all the level attributes of the current level. When SCD3 is enabled for the newly created level attribute, the timestamp with which the previous value is set will be stored in the level attribute designated as SCD 3 effective date.
- **Attribute** - You can select the datastore attribute that is used to store the data for the newly created level attribute from the binding attribute drop-down box. The valid values of the binding attributes constitute the attributes of the dimension table (for star dimension implementation).
- **Staging Attribute** - You can select the datastore attribute that is used to store the data for this level attribute from the drop-down box. The valid values of the staging attribute constitutes the attributes of the staging datastore selected as the binding for the current level.

Note: Within the same level, only one level attribute can act as either Surrogate Key or SCD2 Start Date or SCD2 End Date or SCD3 Previous Attribute or SCD3 Start Date. You will get an error message, when you try to set multiple level attributes with a particular role.

- **Natural Key Members** - Natural Key Members table displays the key members for the Natural key of the level that is selected in the Levels table. The Natural key is also commonly known as "Business Key" or "Application Key", which is usually used to store the key value(s) that can uniquely identify a particular record in the source systems.

For creating or removing natural key members, click the Add + or cross x icons and to reorder the level, click the **up or down arrow** icons, present in the header of the Natural Key Members table. This table contains the Level Attribute column. The level attribute constitutes the natural key of the selected level. The valid values of this column are the list of level attributes of the current level are listed in this column.

3. **Parent Level References Table** - This table defines the list of references from the selected level in the Level table to some other parent level. This table is driven from the Level table

For creating or removing parent level references, click the Add + or cross x icons, present in the header of the Parent Level Attributes table.

Functionally, a parent level reference defines how to retrieve the parent level record from the current record. One or more attributes in the current record are used as a foreign key to match the corresponding natural key attributes in the parent level. The Parent Level References table contains following columns:

- **Name** - It represents the name of the created parent level reference.
In the **Name** text box, enter the name of the newly created parent level reference.
- **Description** - It represents a short description for the newly created parent level reference.
In the **Description** text box, enter a short description for the newly created parent level reference.
- **Parent Level** - Select the required parent level that this reference is pointing to, from the parent level drop-down box. The valid values of the parent level column constitute the list of levels in the dimension that is higher than the currently selected level.

Note: You can select the same parent level in multiple parent level references. It means that there are multiple paths from the current record to that parent level.

- **SCD 2 Setting** - This parameter has two values - None or Trigger History. If you select trigger history as SCD 2 setting, a new row is created during dimension loading if any one of the parent level reference key member columns is not the same as the incoming data.

Parent Level Reference Key Members table is driven from the parent level reference that is selected in the Parent Level References table. If the parent level has a surrogate

key then the key member is surrogate key, if not it is a natural key. Rows in this table are automatically populated with the key members of the parent level. This table contains the following columns:

- **Parent Key Attribute** - The level attribute that is part of the natural key of the parent level. The Dimension component is used in a mapping to load data into dimensions defined in ODI.
- **Foreign Key Attribute** - This parameter represents the datastore column of the current level that is used to match the corresponding natural key member of the parent level. The valid values of the foreign key attribute drop-down box are all the columns of the datastore of the currently selected level.
- **Foreign Key Staging Attribute** - This parameter represents the datastore column of the current level that is used to match the corresponding natural key member of the parent level. The valid values of the foreign key staging attribute drop-down box are all the columns of the staging datastore of the currently selected level.

Note: For a star dimension, you need not specify the key members of a parent level reference, as a dimension record is stored in the same database row, which includes all information of the parent level.

9.2.5.1.3 Hierarchies Tab The Hierarchies tab displays the list of hierarchies that are defined for the newly created dimension. Hierarchy defines how dimension data is summarized and rolled up in BI reporting tools.

For creating or removing hierarchies click the Add + or cross x icons and to reorder the level, click the **up or down arrow** icons, present in the header of the Hierarchies table. The Hierarchies table contains the following columns:

- **Name** - It represents the name of the created hierarchy.
In the **Name** text box, enter the name of the newly created hierarchy.
- **Description** - It represents a short description for the newly created hierarchy.
In the **Description** text box, enter a short description for the newly created hierarchy.
- **Default** - When a dimension has multiple hierarchy, query tools show the default hierarchy. Only one hierarchy can be defined as default.

The Hierarchy Members table displays the list of level members in the hierarchy selected in the Hierarchies table. For adding or removing hierarchy Members click the Add + or cross x icons and to reorder the level, click the **up or down arrow** icons present in the header of the Hierarchies table. The hierarchy members are sorted based on the natural order of the levels in the dimension. This table contains the following columns:

- **Level** - Select the required level for the newly created hierarchy from the Level drop-down box. The valid values listed in the drop-down box are the list of levels in the dimension that are not a member of the currently selected hierarchy which are of lower level than any preceding hierarchy members.

For example, if we have a dimension with three levels - Brand, Category and Product, if the first hierarchy member is Brand, then the valid levels of the second hierarchy members are Category and Product.

- **Parent Level Reference** - The parent level reference that is used to navigate from the selected level to the level of the immediately preceding hierarchy members.

The column is a drop-down box and its valid values are the parent level references between the current level and the level of the immediately preceding hierarchy member.

Skip Level Members table displays the list of skip level members in the hierarchy member selected in the Hierarchy Members table. This allows the data present in the level, to have multiple paths to roll up to different parent levels, wherein some paths may skip one or more parent levels in the same hierarchy.

For example, assume that we have a hierarchy **Store** in the order- City -> Region -> State -> County in a dimension called Store of a retail company. In this company, some stores are more important and they report directly to the regional office. As a result, a store can have two parent levels, city and region, where City is optional. To describe this hierarchy, we have a hierarchy member for Store level and its parent level is City. This hierarchy member also has a skip level and its parent level reference is pointing to Region level.

Since parent levels can skip multiple paths, click + or cross x buttons to describe these different roll up paths. The Skip Levels table contains the following columns:

Parent Level Reference: The parent level reference that is used to skip to some preceding level in the hierarchy. This column is a drop-down box and its valid values are the parent level references defined for the level of the currently selected hierarchy member, and including those parent level references that do not point to the immediately preceding level.

9.2.5.2 Using the Cube Editor

Cube Editor has two essential tabs for easy and effective administration of a cube object. They are:

- [Definition Tab](#)
- [Details Tab](#)

9.2.5.2.1 Definition Tab

Definition tab of the Cube Editor allows you to define the following properties of a newly created or existing cube objects:

- **Name** - It represents the name of the created cube object.
In the **Name** text box, enter the required name of the created cube object.
- **Description** - It represents a short description of the created cube object.
In the **Description** text box, enter a short description for the created cube object.
- **Binding Datastore**- The datastore that is used to store the cube data can be selected using the wheel. It displays all the datastores present in all the models of ODI repository. Click the Search icon, present beside the **Cube Table** text box, to select the required cube table.

9.2.5.2.2 Details Tab

Details tab of the Cube Editor allows you to define the following properties of a newly created or existing cube objects:

Note: ODI supports only single datastore to store cube information.

- **Dimensions Table** - The Dimensions table displays the list of dimensions being used by the cube. You can add or delete a dimension using the Add "+" or Cross "X" buttons present in the header of the Dimensions Table. The Dimensions Table has the following columns:
 - **Level** - It denotes the level of the dimension that is referenced to the cube. When you click the browse icon, present beside any Level parameter in the table, it displays all the levels present in all the dimensions of ODI repository. When you select the level, the qualified name of the level (in the form of <dimension_name>.<level_name>) is displayed in the column.
 - **Role** - If you plan to use the same dimension multiple times in a cube then you have to specify an alternate name to uniquely identify the same dimension. This can be done by using the role column.
- **Key Binding Table** -The Key binding table is driven by the dimension table. It has the following columns:
 - **Dimension Key** - If any dimension is selected in the Dimension Table then the Key binding datastore displays the keys present in the Dimension Key column. If the dimension uses a surrogate key then the respective surrogate key is displayed, if surrogate keys are not used then Natural Keys are displayed.
 - **Binding Attribute** - This parameter displays all the attributes present in the cube binding datastore. If any attribute is already bound to a Dimension key or a measure then those attributes are not listed.
- **Measure Tables** - Measure Tables are used to add or delete new measures. Measure Table comprises of the following columns:
 - **Name** - It denotes the name of the created measure.
In the **Name** column, enter the name of the newly added measure.
 - **Description** - It denotes a short description of the measure creation.
In the **Description** column, enter a short description of then newly created measure.
 - **Data Type** - You can select the data type of the level attribute from the Data Type drop-down box. The valid values of the column are all the data types in the "Generic SQL" technology defined in ODI repository.
 - **Size** - It denotes the length or precision of the measure.
 - **Scale** - It denotes the scale of this measure (for numeric data type).
 - **Binding Column** - This column denotes the datastore that is used to store the data for a measure. The valid values of this binding column are the columns of the cube table. If any attribute is already bound to a dimension key or a measure then those attributes are not listed in this column.

9.3 Using Dimensional Components in Mappings

Dimensional Objects can be used extensively in mappings.

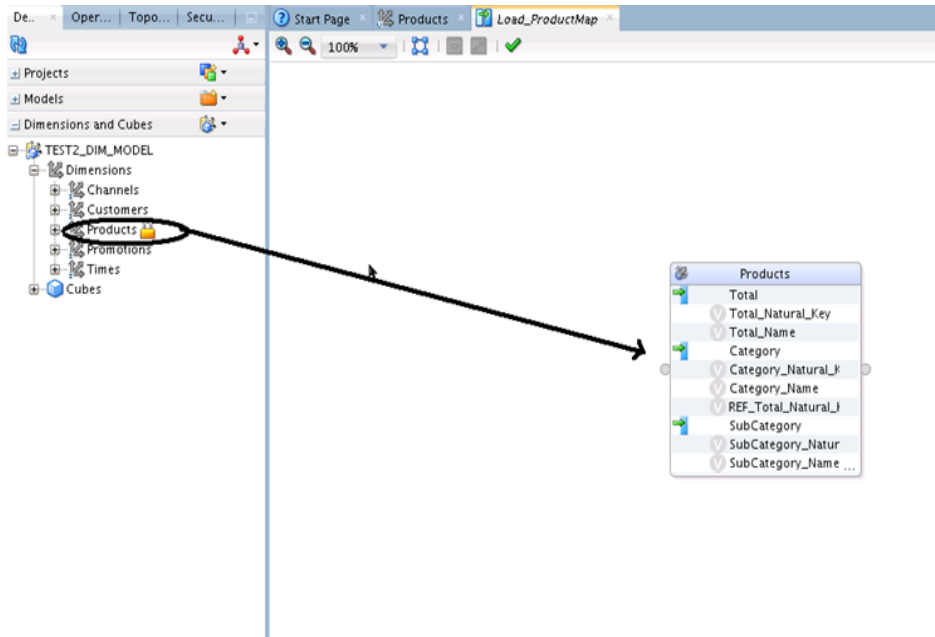
9.3.1 Using Dimension Component in Mapping

The Dimension component is used in a mapping to load data into dimensions and slowly changing dimensions.

The Dimension component contains one group for each level in the dimension. The groups use the same name as the dimension levels. The level attributes of each level are listed under the group that represents the level.

You cannot map a data flow to the surrogate key attribute or the parent reference key attribute of any dimension level.

To use a dimension in a Mapping, drag the dimension from the Dimensions and Cubes Model and drop it in the Mapping Editor Canvas, as shown below:



9.3.1.1 Dimension Component Properties Editor

The Dimension component has the following properties:

9.3.1.1.1 Attributes

Attributes are fixed and provided by the base dimension. It is a data field within a level.

9.3.1.1.2 General Properties

The general properties of the dimension component includes:

- **Name**- It denotes the name of the created dimension component
- **Description** - It denotes a short description provided during dimension component creation
- **Dimension** - It is a read only property which provides information on which base dimension it is related to.
- **Datastore** - It is a read only property, and this value is derived from the base dimension and it is the name of the datastore to which the base dimension is bound to.
- **Sequence** - It refers to the sequence that is used by the base dimension.
- **Storage Type** - The default value for storage type is **Star**.
- **Component Type** - The default value for component type is **Dimension**.

- **Pattern** - The default value for pattern is **Dimension Pattern** and if you create your own pattern they get listed here, thereby enabling you to select the required pattern.

9.3.1.1.3 Connector Points

Connector points define the connections between components inside a mapping. The Dimension Component has only input connector points. The output connector points are invalid. The properties of input connector points are:

Table 9–2 Properties of Input Connector Points

Property	Description
Name	It denotes the name of the connector point. This field can be edited.
Description	It denotes the description of the connector point.
Bound Object	It denotes the name of the object to which the connector point is bound to. This field remains blank, if the component type does not support connector point binding.
Connected From	It denotes the name of the preceding components to which this component connects from.

9.3.1.1.4 History Properties

The History properties of the dimension component includes the following:

Note: These properties are applicable only for Type 2 SCDs.

- **Default Effective Time of Initial Record**- It represents the default value assigned as the effective time for the initial load of a particular dimension record.
- **Default Effective Time of Open Record** - It represents the default value set for the effective time of the open records, after the initial record. This value should not be modified.
- **Default Expiration Time of Open Record** - It represents a date value that is used as the expiration time of a newly created open record for all the levels in the dimension.
- **Slowly Changing Type**- This is a read only property and this value is set based on the type of SCD settings.
- **Type 2 Gap** - It represents the time interval between the expiration time of an old record that is versioned and the start time of the current record that has just been versioned.

When the value of a triggering attribute is updated, the current record is closed and a new record is created with the updated values. Because the closing of the old record and opening of the current record occur simultaneously, it is useful to have a time interval between the end time of the old record and the start time of the open record, instead of using the same value for both.

- **Type 2 Gap Unit** - It represents the unit of time used to measure the gap interval represented in the Type2 Gap property. Available options are: Seconds, Minutes, Hours, Days, and Weeks. The default value is Seconds.

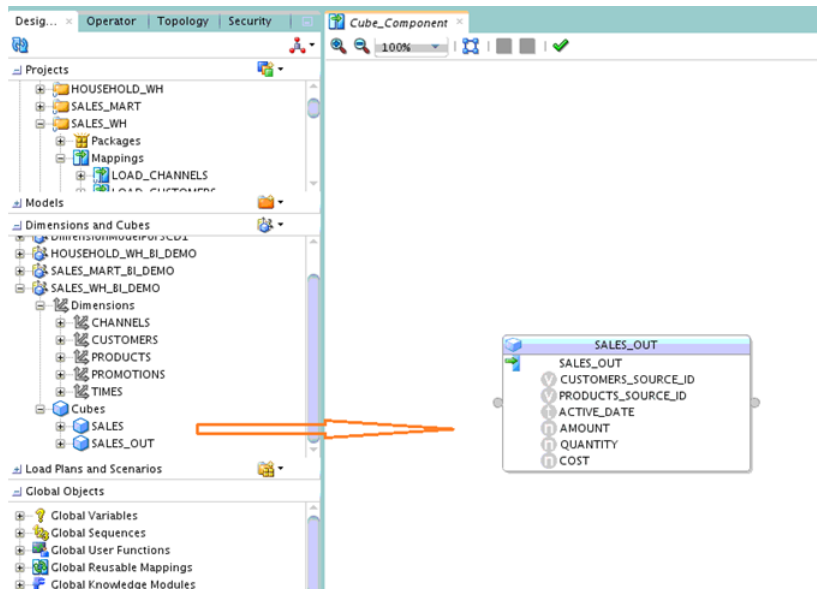
9.3.1.1.5 Target Properties

Select the **Enable Source De-duplicate** check box present under target properties tab, to make sure that no duplicate records are processed from the source.

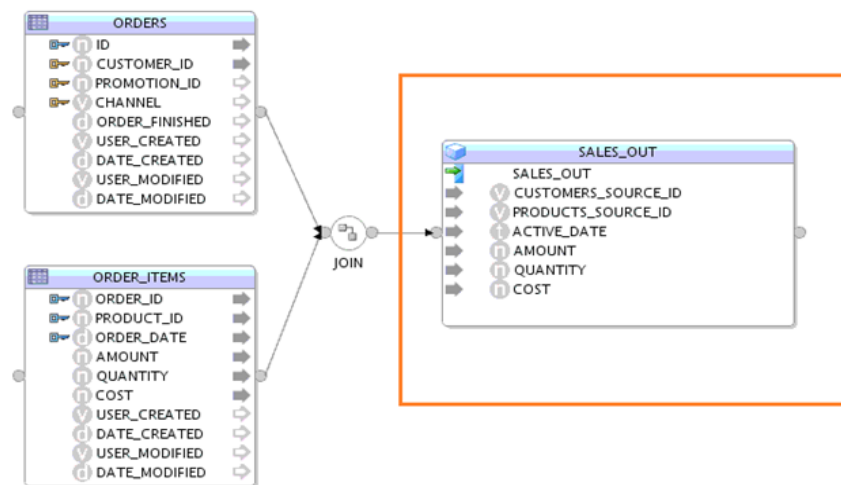
9.3.2 Using Cube Component in Mappings

The cube component is based on a cube object and has a set of attributes according to the base cube.

When you drag and drop a cube object onto to a mapping editor, a cube component gets created based on that cube.



Note: The cube component can be used only as a target and cannot be used as a source for any other component.



9.3.2.1 Cube Component Properties Editor

The Cube component has the following properties:

9.3.2.1.1 Attributes Cube component has a set of input map attributes derived from the base cube. They are:

- **Measure Attributes** - Each measure in base cube of the cube component has a corresponding map attribute in cube component. The map attribute is bound to the cube measure. The measure name is used as the map attribute name.
- **Dimension Natural key Attributes** - Each natural key attribute of the referenced level has a corresponding map attribute in cube component. The map attribute is bound to the natural key level attribute of the referenced level.
 - If the dimension reference of the cube has role qualifier set, then the map attribute name is in the form of <dimension name>_<role name>_<attribute name>.
 - If the dimension reference of the cube has no role qualifier, then the map attribute name is in the form of <dimension name>_<attribute name>.
- **Active Date** - It is a special map attribute which is created in the case that the base cube of the cube component references at least one SCD2 dimension.

If the map attribute in cube component is represented as a natural key identifier of the referenced dimension, it has the following properties:

- **Target**
 - Expression
 - Execute on Hint
 - Fixed Execution Location
- **General**
 - Name
 - Description
 - Data Type
 - Size
 - Scale
 - Attribute Role - It is a read-only property, which indicates the attribute, represented as a natural key identifier.
 - Bound Object - It is a read-only property. The attribute is bound to a natural key level attribute

If the map attribute in cube component is represented as a cube measure, it has the following properties.

- **Target**
 - Expression
 - Execute on Hint
 - Fixed Execution Location
- **General**
 - Name
 - Description
 - Data Type

- Size
- Scale
- Attribute Role - It is a read-only property, which indicates the attribute, represented as a cube measure.
- Bound Object - It is a read-only property. The attribute is bound to a cube measure.
- Source Aggregation Function - It denotes the source loading aggregation function for the measure. This property is worked together with property "Enable Source Aggregation" on cube component. In the user interface, a combo box lists all the aggregate functions of the generic technology.

The active date attribute in cube component has following properties:

- **Target**
 - Expression
 - Execute on Hint
 - Fixed Execution Location
- **General**
 - Name
 - Description
 - Data Type - It denotes the default data type for Active Date attribute - "TIMESTAMP".
 - Size
 - Scale
 - Attribute Role - It is a read-only property, which is set to Active Date. Active Date attribute has a default expression which is set to "FUNC_SYSDATE". "FUNC_SYSDATE" is a global user function. This function is also used in Dimension Component.

9.3.2.1.2 General Properties Cube component has the following general properties:

- **Name** - It denotes the name of the cube component.
- **Description** - It denotes a short description of the mapping.
- **Cube** - It is a read-only property that provides the base cube of the cube component.
- **Datastore** - It is a read-only property and this value is derived from the base cube. It is the bound datastore of the base cube of this component.
- **Component Type** - It is a ready only property and its value is **Cube**, by default.
- **Pattern** - At present, you can select only pattern - **Cube Pattern**.

9.3.2.1.3 Target Properties Cube component has following target properties:

- **Integration Type** - Integration Type is of three types. They are:
 - None
 - Incremental Update
 - Control Append

The default value is Incremental Update. The integration type ultimately helps to determine the default IKM and limit the set of IKM choices.

- Enable Source Aggregation** - If this property is enabled, an aggregate component is added in the expanded map of the cube component before loading the fact table. The source row set is grouped by the dimension reference attributes. Measure aggregation functions are determined by the SOURCE_AGGREGATION_FUNCTION attribute properties.

9.3.2.1.4 Connector Points The cube component can be used only as a target. Hence it has only one input connector point and has no output connector points.

9.4 Expanding Dimensional Components

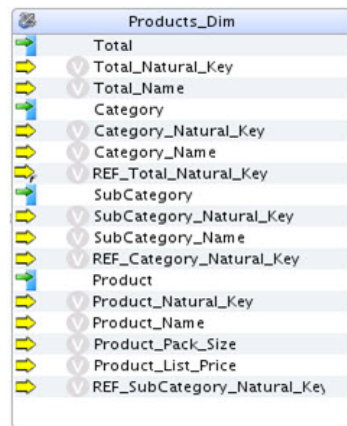
This section explains briefly about expanding the dimension and cube components.

9.4.1 Expanding Dimension Component

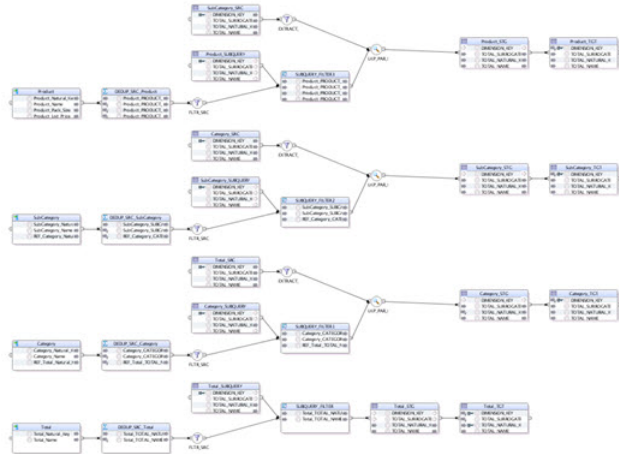
Dimension component is an expandable component and it can be expanded like a reusable Mapping.

To expand a dimension component, right-click the dimension component and from the pop-up menu select **Open**.

It switches to the expanded mapping of the dimension component. The expanded mapping is determined by the selected pattern.



A typical expanded mapping of a dimension component looks like:

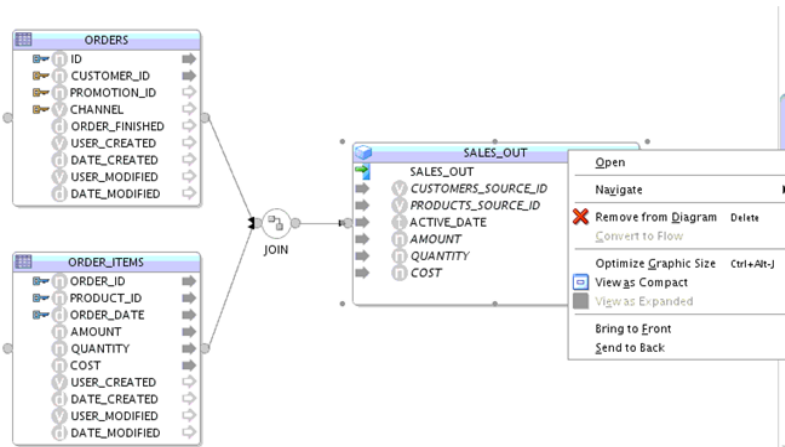


9.4.2 Expanding Cube Component

Cube component is an expandable component and it can be expanded like a reusable Mapping.

To expand a cube component, right-click the cube component and from the pop-up menu select **Open**.

It switches to the expanded mapping of the cube component. The expanded mapping is determined by the selected pattern.



A typical expanded mapping of a cube component looks like:



Using Compatibility Mode

This chapter describes how to use Oracle Data Integrator in compatibility mode. Compatibility mode allows you to import and run Interfaces built in earlier versions of ODI.

This chapter includes the following sections:

- [About Compatibility Mode](#)
- [Creating Compatible Mappings](#)
- [About Internal Identifiers \(IDs\)](#)
- [Renumbering Repositories](#)

10.1 About Compatibility Mode

Oracle Data Integrator 12c provides a backward-compatibility feature that allows you to import and run Interfaces created in ODI 11g. You can create these 11g-compatible mappings in the following two ways:

- When upgrading to ODI 12c using the Upgrade Assistant, disable the **Upgrade interfaces to 12c mappings - losing 11g SDK compatibility** option
- Create mappings using the ODI 11g SDK

De-selecting the 11g SDK compatibility flag in the Upgrade Assistant is only required if you want to *modify* your 11g-compatible mappings later, using the 11g SDK. 11g SDK-compatible mappings are read-only when viewed in ODI Studio 12c. The 11g SDK is the only way to modify an upgraded 11g-compatible mapping. If you do not want to modify your 11g-compatible mappings, you do not need to enable 11g SDK compatibility using the Upgrade Assistant.

11g-compatible mappings use a special mapping component, the "11g Compatible Dataset," to emulate 11g Interface behavior.

You can run an 11g SDK-compatible mapping in ODI 12c in exactly the same way as running an ODI 12c mapping.

You can convert an 11g SDK-compatible mapping to a true ODI 12c mapping, using a context menu command.

ODI 12c mappings cannot be converted to 11g compatibility.

Maintaining 11g SDK Compatibility with Imported Interfaces

When importing an 11g-compatible mapping, you must set a user preference to determine whether it is imported as an 11g SDK-compatible mapping, or converted to 12c mapping:

1. In ODI Studio, select **ODI** from the menu bar, select **User Interface**, and then select **Designer**.
2. Enable the option **Maintain 11g SDK compatibility for migrated interfaces**.
When importing an 11g Interface export file, this option will determine the imported interface's compatibility as either 11g SDK, or 12c.

10.2 Creating Compatible Mappings

You have two options for creating 11g SDK-compatible mappings:

- [Creating Mappings using Upgrade Assistant](#)
- [Creating Mappings with the 11g SDK in ODI 12c](#)

10.2.1 Creating Mappings using Upgrade Assistant

You can upgrade an earlier version of ODI to ODI 12c, using the Oracle Upgrade Assistant. While upgrading ODI, 11g Interfaces are converted to 12c mappings by default.

The Upgrade Assistant gives you the option of enabling 11g SDK Compatibility Mode, by de-selecting an option. 11g SDK Compatibility Mode allows you to modify upgraded 11g-compatible mappings using the ODI 11g SDK.

To convert ODI 11g Interfaces to 11g-compatible mappings, de-select the **Upgrade interfaces to 12c mappings - losing 11g SDK compatibility** option in "Task 6 Selecting the ODI Upgrade Option" of Chapter 4, "Upgrading Your Oracle Data Integrator Environment", in *Upgrading Oracle Data Integrator*.

10.2.2 Creating Mappings with the 11g SDK in ODI 12c

In any upgraded or standard ODI 12c environment, you can create 11g-compatible mappings using the ODI 11g SDK. In ODI 12c, these mappings are read-only.

If you upgraded to ODI 12c using the Oracle Upgrade Assistant, and you enabled 11g compatibility mode, you can modify 11g-compatible mappings using the 11g SDK.

To create an 11g-compatible mapping using the 11g SDK, review the *Java API Reference for Oracle Data Integrator*.

10.3 About Internal Identifiers (IDs)

To ensure object uniqueness across several work repositories, ODI 11g used a mechanism to generate unique IDs for objects (such as technologies, data servers, Models, Projects, Mappings, KMs, etc.). Every object in Oracle Data Integrator 11g is identified by an internal ID. The internal ID appears on the Version tab of each object.

ODI 11g Master and Work Repositories are identified by their unique 3-digit internal IDs. The internal ID of an 11g object is calculated by appending the value of the RepositoryID to an automatically incremented number:

<UniqueNumber><RepositoryID>

If the Repository ID is shorter than 3 digits, the missing digits are completed with "0". For example, if a repository has the ID 5, possible internal IDs of the objects in this repository could be: 1005, 2005, 3005, ..., 1234567005. Note that all objects created within the same repository have the same three last digits, in this example 005.

This internal ID is unique for the object type within the repository and also unique between repositories for the object type because it contains the repository unique ID.

Important Export/Import Rules and Guidelines

Due to the structure of the 11g object IDs, these guidelines should be followed:

- ODI 11g Work repositories must always have different internal IDs. Work repositories with the same ID are considered to contain the same objects.
- When importing ODI 11g objects from an 11g repository, you must define an Upgrade Key. The Upgrade Key should uniquely identify the set of repositories that are working together.

10.4 Renumbering Repositories

Renumbering a master or work repository consists of changing the repository ID and the internal ID of the objects stored in the repository. This operation is only available on repositories that are in 11g Compatibility Mode. In ODI 12c, repositories have unique global identifiers, which cannot be (and do not need to be) renumbered.

Renumbering a repository is advised when two repositories have been created with the same ID. Renumbering one of these repositories allows object import/export between these repositories without object conflicts.

WARNING: Renumbering a repository is an administrative operation that requires you to perform a backup of the repository that will be renumbered on the database.

Renumbering an 11g Compatible Repository

1. In the Topology Navigator, expand the **Repositories** panel.
2. Expand the **Master Repositories** or **Work Repositories** node and right-click the repository you want to renumber.
3. Select **Renumber...**
4. In the Renumbering the Repository - Step 1 dialog click **Yes**.
5. In the Renumbering the Repository - Step 2 dialog enter a new and unique ID for the repository and click **OK**.
6. The repository and all the details stored in it such as topology, security, and version management details are renumbered.

Creating and Using Procedures, Variables, Sequences, and User Functions

This chapter describes how to work with procedures, variables, sequences, and user functions. An overview of these components and how to work with them is provided.

This chapter includes the following sections:

- [Working with Procedures](#)
- [Working with Variables](#)
- [Working with Sequences](#)
- [Working with User Functions](#)

11.1 Working with Procedures

This section provides an introduction to procedures and describes how to create and use procedures in Oracle Data Integrator.

The following sections describe how to create and use procedures:

- [Introduction to Procedures](#)
- [Creating Procedures](#)
- [Using Procedures](#)
- [Encrypting and Decrypting Procedures](#)

11.1.1 Introduction to Procedures

A **Procedure** is a set of commands that can be executed by an agent. These commands concern all technologies accessible by Oracle Data Integrator (OS, JDBC, JMS commands, etc).

A Procedure is a reusable component that allows you to group actions that do not fit in the mapping framework. Procedures should be considered only when what you need to do can't be achieved in a mapping. In this case, rather than writing an external program or script, you would include the code in Oracle Data Integrator and execute it from your packages. Procedures require you to develop all your code manually, as opposed to mappings.

A procedure is composed of tasks, which have properties. Each task has two commands; On Source and On Target. The commands are scripts, possibly mixing different languages. The tasks are executed sequentially. Some commands may be

skipped if they are controlled by an option. These options parameterize whether or not a command should be executed, as well as the code of the commands.

The code within a procedure can be made generic by using options and the ODI Substitution API.

Before creating a procedure, note the following:

- Although you can perform data transformations in procedures, using them for this purpose is not recommended; use mappings instead.
- If you start writing a complex procedure to automate a particular recurring task for data manipulation, you should consider converting it into a Knowledge Module. Refer to the *Developing Knowledge Modules with Oracle Data Integrator* for more information.
- Whenever possible, try to avoid operating-system-specific commands. Using them makes your code dependent on the operating system that runs the agent. The same procedure executed by agents on two different operating systems (such as UNIX and Windows) will not work properly.

11.1.2 Creating Procedures

Creating a procedure follows a standard process which can vary depending on the use case. The following step sequence is usually performed when creating a procedure:

1. [Create a New Procedure](#)
2. [Define the Procedure's Options](#)
3. [Create and Manage the Procedure's Tasks](#).

When creating procedures, it is important to understand the following coding guidelines:

- [Writing Code in Procedures](#)
- [Using the Substitution API](#)
- [Handling RDBMS Transactions](#)
- [Binding Source and Target Data](#)

11.1.2.1 Create a New Procedure

To create a new procedure:

1. In Designer Navigator select the **Procedures** node in the folder under the project where you want to create the procedure.
2. Right-click and select **New Procedure**.
3. On the **Definition** tab fill in the procedure **Name**.
4. Check **Multi-Connections** if you want the procedure to manage more than one connection at a time.

Multi-Connections: It is useful to choose a multi-connection procedure if you wish to use data that is retrieved by a command sent on a source connection in a command sent to another (target) connection. This data will pass through the execution agent. By enabling Multi-Connections, you can use both Target and Source fields in the Tasks (see "[Create and Manage the Procedure's Tasks](#)" on page 11-4).

If you access one connection at a time (which enables you to access different connections, but only one at a time) leave the Multi-Connections box unchecked. Only Target tasks will be used.

5. Select the **Target Technology**, and if the **Multi-Connections** box is checked, also select the **Source Technology**. Each new Procedure line will be based on this technology. You can also leave these fields empty and specify the technologies in each procedure command.

Caution: Source and target technologies are not mandatory for saving the Procedure. However, the execution of the Procedure might fail, if the related commands require to be associated with certain technologies and logical schemas.

6. Optionally, select **Use Unique Temporary Object Names** and **Remove Temporary Objects On Error**:
 - **Use Unique Temporary Object Names:** If this procedure can be run concurrently, enable this option to create non-conflicting temporary object names.
 - **Remove Temporary Objects On Error:** Enable this option to run cleanup tasks even when a session encounters an error.
7. Optionally, enter a **Description** of this procedure.
8. From the **File** menu, click **Save**.

A new procedure is created, and appears in the Procedures list in the tree under your Project.

11.1.2.2 Define the Procedure's Options

Procedure options act like parameters for your tasks and improve the code reusability.

There are three types of options:

- **Boolean** options. Their value can be used to determine whether individual command are executed or not. They act like an "if" statement.
- **Value** and **Text** options used to pass in short or long textual information respectively. The values of these options can only be recovered in the code of the procedure's commands, using the **getOption()** substitution method. When using your procedure in a package, its values can be set on the step.

To create procedure's options:

1. In Designer Navigator, double-click the procedure you want to configure. The Procedure Editor opens.
2. Select the Options tab. In the Options table, click the Add Option button to add a row to the table. Each row represents one option.
3. Edit the following fields of an option:
 - **Name:** Name of the option as it appears in the graphical interface
 - **Type:** Type of the option.
 - **Boolean:** The option is boolean: True = 1/False = 0. These are the only options used for procedures and KMs to determine if such tasks should be executed or not.

- **Text:** It is an alphanumerical option. Maximum size is not limited. Accessing this type of option is slower than for value options.
- **Value:** It is an alphanumerical option. Maximum size is 250 characters.
- **Default Value:** Value that the option will take, if no value has been specified by the user of the procedure or the KM.
- **Direct Execution Value:** Use in cases when you want to execute or test a procedure with different option values. The default for the Direct Execution Value is the corresponding Default Value for this option. You can overwrite a Default Value, then a Direct Execution Value will get a new default value if it was not changed before. Using the **Reset Direct Execution Value to Default** context menu item of an option will reset it to the default value.
- **Description:** Optional, short description of the option. For boolean options, this description is displayed in the Options tab of the Property Inspector when a row in the Tasks table is selected.
- **Help:** Optional, descriptive help on the option. For procedures, this text is displayed in the properties pane when the procedure is selected in a mapping.

Note: Right-click an option to access the context menu. From this menu you can **Duplicate** the option, or **Reset Direct Execution Value to Default**.

4. If there are multiple Options, set their order in the table. Select a row, and use the up and down buttons in the menu bar to change its position in the list. Options are checked in order from top to bottom.
5. Repeat these operations for each option that is required for the procedure.
6. From the **File** menu, click **Save**.

11.1.2.3 Create and Manage the Procedure's Tasks

Most procedures only execute commands on a target. In some cases, your procedure may require reading data and performing actions using this data. In these cases, specify the command to read the data in the **Source** fields and the actions performed with this data in the **Target** fields of the Tasks tab. Refer to "[Binding Source and Target Data](#)" on page 11-8 for more information. You can leave the Source fields blank if you are not performing commands on source datastores.

Create and manage the tasks in a procedure using the following options:

- [Creating a Procedure's Tasks](#)
- [Duplicating Tasks](#)
- [Deleting Tasks](#)
- [Changing the Order of Tasks](#)

Creating a Procedure's Tasks

1. In Designer Navigator double-click the procedure for which you want to create a command. The Procedure Editor opens.
2. In the Procedure Editor, go to the Tasks tab. Any tasks already in the procedure are listed in a table.

3. Click **Add**. A new task row is created in the table. Edit the fields in the row to configure your task.

The following fields are available:

Notes:

- If you do not see a field, use the Select Columns button to show hidden columns in the table. Alternatively, open the property inspector and select a task row to see some of the following fields in the property inspector. The **Always Execute** option is *only* visible in the property inspector.
 - The transaction, commit, and transaction isolation options work only for technologies supporting transactions.
-
-

- **Name:** Enter a name for this task.
- **Cleanup:** Mark a task as cleanup task if you would like it to be executed even when the procedure results in error. For example, use cleanup tasks to remove temporary objects.
- **Ignore Errors** must be checked if you do not want the procedure to stop if this command returns an error. If this box is checked, the procedure command will generate a "warning" message instead of an "error," and the procedure will not be stopped.
- **Source/Target Transaction:** Transaction where the command will be executed.
The Transaction and Commit options allow you to run commands within transactions. Refer to "[Handling RDBMS Transactions](#)" on page 11-8 for more information.
- **Source/Target Commit:** Indicates the commit mode of the command in the transaction.
The Transaction and Commit options allow you to run commands within transactions. Refer to "[Handling RDBMS Transactions](#)" on page 11-8 for more information.
- **Source/Target Technology:** Technology used for this command. If it is not set, the technology specified on the Definition tab of the Procedure editor is used.
- **Source/Target Command:** Text of the command to execute. You can open the Expression Editor by clicking ... in the command field.
The command must be entered in a language appropriate for the selected technology. Refer to "[Writing Code in Procedures](#)" on page 11-7 for more information.
Oracle recommends using substitution methods to make the code generic and dependent on the topology information. Refer to "[Using the Substitution API](#)" on page 11-7.
- **Source/Target Context:** Forced Context for the execution. If it is left undefined, the execution context will be used. You can leave it undefined to ensure the portability of the code in any context.
- **Source/Target Logical Schema:** Logical schema for execution of the command.
- **Source/Target Transaction Isolation:** The transaction isolation level for the command.

- **Log Counter:** Shows which counter (Insert, Update, Delete or Errors) will record the number of rows processed by this command. Note that the Log Counter works only for Insert, Update, Delete, and Errors rows resulting from an Insert or Update SQL statement.

Tip: After executing a Procedure, you can view the counter results in Operator Navigator. They are displayed in the Step or Task editor, on the Definition tab, in the Record Statistics section.

- **Log level:** Logging level of the command. At execution time, the task generated for this command will be kept in the Session log based on this value and the log level defined in the execution parameters. Refer to "Execution Parameters" in *Administering Oracle Data Integrator* for more details on the execution parameters.
- **Log Final Command:** The ODI execution logs normally write out the task code before the final task code processing. Enable this flag if you would like to log the final processed command in addition to the pre-processed command.
- **Options:** In the **Options** node in the property inspector is a table with the list of all available options. These options are only visible from the property inspector:
 - **Always Execute:** Enable if you want this command to be executed all the time regardless of the other option values.
 - Other options are listed in the table. The available options differ depending on the procedure. If you did not select Always Execute, you can select individual options which you want to be executed for the selected task.

Note: The options are only visible in the property inspector. Select a task row, and then in the property inspector, select the **Options** tab, to see the task options.

4. From the **File** menu, click **Save**.

You can make a copy of existing tasks in the tasks list:

Duplicating Tasks

1. Go to the **Tasks** tab of the Procedure.
2. Select the command to duplicate.
3. Right-click then select **Duplicate**. A new row is added to the list of tasks. It is a copy of the selected command.
4. Make the necessary modifications and from the **File** menu, click **Save**.

You can delete a task from the list:

Deleting Tasks

1. Go to the **Tasks** tab of the Procedure.
2. Select the command line to delete.
3. From the Editor toolbar, click **Delete**, or right-click on the row and select **Delete** from the context menu.

The command line will disappear from the list.

You can change the order in which tasks are executed.

Tasks are executed in the order displayed in the **Tasks** tab of the Procedure Editor. It may be necessary to reorder them.

Changing the Order of Tasks

1. Go to the **Tasks** tab of the Procedure.
2. Click on the command line you wish to move.
3. From the tasks table toolbar, click the arrows to move the command line to the appropriate position.

Writing Code in Procedures

You can open the expression editor to write and modify the code in a procedure.

Commands within a procedure can be written in several languages. These include:

- **SQL:** or any language supported by the targeted RDBMS such as PL/SQL, Transact SQL etc. Usually these commands can contain Data Manipulation Language (DML) or Data Description Language (DDL) statements. Using SELECT statements or stored procedures that return a result set is subject to some restrictions. To write a SQL command, you need to select:
 - A valid RDBMS technology that supports your SQL statement, such as Teradata or Oracle etc.
 - A logical schema that indicates where it should be executed. At runtime, this logical schema will be converted to the physical data server location selected to execute this statement.
 - Additional information for transaction handling as described further in section Handling RDBMS Transactions.
- **Operating System Commands:** Useful when you want to run an external program. In this case, your command should be the same as if you wanted to execute it from the command interpreter of the operating system of the Agent in charge of the execution. When doing so, your objects become dependent on the platform on which the agent is running. To write an operating system command, select "Operating System" from the list of technologies of your current step. It is recommended to use for these kind of operations the OdiOSCommand tool as this tool prevents you from calling and setting the OS command interpreter.
- **ODI Tools:** ODI offers a broad range of built-in tools that you can use in procedures to perform some specific tasks. These tools include functions for file manipulation, email alerts, event handling, etc. They are described in detail in the online documentation. To use an ODI Tool, select **ODITools** from the list of technologies of your current step.
- **Scripting Language:** You can write a command in any scripting language supported by Oracle Data Integrator. By default, ODI includes support for the following scripting languages that you can access from the technology list box of the current step: Jython, Groovy, NetRexx, and Java BeanShell.

Using the Substitution API

Oracle recommends that you use the ODI substitution API when writing commands in a procedure, to keep it independent of the context of execution. You can refer to the online documentation for information about this API. Common uses of the substitution API are given below:

- Use `getObjectName()` to obtain the qualified name of an object in the current logical schema regardless of the execution context, rather than hard coding it.
- Use `getInfo()` to obtain general information such as driver, URL, user etc. about the current step
- Use `getSession()` to obtain information about the current session
- Use `getOption()` to retrieve the value of a particular option of your procedure
- Use `getUser()` to obtain information about the ODI user executing your procedure.

When accessing an object properties through Oracle Data Integrator' substitution methods, specify the flexfield Code and Oracle Data Integrator will substitute the Code by the flexfield value for the object instance. See "Using Flexfields" in the *Developing Knowledge Modules with Oracle Data Integrator* for more information on how to create and use flexfields.

Handling RDBMS Transactions

Oracle Data Integrator procedures include an advanced mechanism for transaction handling across multiple steps or even multiple procedures. Transaction handling applies only for RDBMS steps and often depends on the transaction capabilities of the underlying database. Within procedures, you can define for example a set of steps that would be committed or rolled back in case of an error. You can also define up to 10 (from 0 to 9) independent sets of transactions for your steps on the same server. Using transaction handling is of course recommended when your underlying database supports transactions. Note that each transaction opens a connection to the database.

However, use caution when using this mechanism as it can lead to deadlocks across sessions in a parallel environment.

Binding Source and Target Data

Data binding in Oracle Data Integrator is a mechanism in procedures that allows performing an action for every row returned by a SQL SELECT statement.

To bind source and target data:

1. In the Task properties, open the Command Editor by hovering the mouse pointer over the Source or Target Command field and then clicking the gear icon that displays on the right.
2. In the **Source Command** editor, specify the SELECT statement.
3. In the **Target Command** editor, specify the action code. The action code can itself be an INSERT, UPDATE or DELETE SQL statement or any other code such as an ODI Tool call, Jython or Groovy. Refer to *Oracle Data Integrator Tool Reference* for details about the ODI Tools syntax.

The values returned by the source result set can be referred to in the action code using the column names returned by the SELECT statement. They should be prefixed by colons ":" whenever used in a target INSERT, UPDATE or DELETE SQL statement and will act as "bind variables". If the target statement is not a DML statement, then they should be prefixed by a hash "#" sign and will act as substituted variables. Note also that if the resultset of the Source tab is passed to the Target tab using a hash "#" sign, the target command is executed as many times as there are values returned from the Source tab command.

The following examples give you common uses for this mechanism. There are, of course, many other applications for this powerful mechanism.

Example 11–1 Loading Data from a Remote SQL Database

Suppose you want to insert data into the Teradata PARTS table from an Oracle PRODUCT table. [Table 11–1](#) gives details on how to implement this in a procedure step.

Table 11–1 Procedure Details for Loading Data from a Remote SQL Database

Source Technology	Oracle
Source Logical Schema	ORACLE_INVENTORY
Source Command	<pre>select PRD_ID MY_PRODUCT_ID, PRD_NAME PRODUCT_NAME, from <%=odiRef.getObjectName("L", "PRODUCT", "D")%></pre>
Target Technology	Teradata
Target Logical Schema	TERADATA_DWH
Target Command	<pre>insert into PARTS (PART_ID, PART_ORIGIN, PART_NAME) values (:MY_PRODUCT_ID, 'Oracle Inventory', :PRODUCT_NAME)</pre>

ODI will implicitly loop over every record returned by the SELECT statement and bind its values to ":MY_PRODUCT_ID" and ":PRODUCT_NAME" bind variables. It then triggers the INSERT statement with these values after performing the appropriate data type translations.

When batch update and array fetch are supported by the target and source technologies respectively, ODI prepares arrays in memory for every batch, making the overall transaction more efficient.

Note: This mechanism is known to be far less efficient than a fast or multi load in the target table. You should only consider it for very small volumes of data.

The section Using the Agent in the Loading Strategies further discusses this mechanism.

Example 11–2 Sending Multiple Emails

Suppose you have a table that contains information about all the people that need to be warned by email in case of a problem during the loading of your Data Warehouse. You can do it using a single procedure task as described in [Table 11–2](#).

Table 11–2 Procedure Details for Sending Multiple Emails

Source Technology	Oracle
Source Logical Schema	ORACLE_DWH_ADMIN
Source Command	<pre>Select FirstName FNAME, EMailaddress EMAIL From <%=odiRef.getObjectName("L", "Operators", "D")%> Where RequireWarning = 'Yes'</pre>
Target Technology	ODITools
Target Logical Schema	None

Table 11–2 (Cont.) Procedure Details for Sending Multiple Emails

Target Command	<pre>OdiSendMail -MAILHOST=my.smtp.com -FROM=admin@mycompany.com "-TO=#EMAIL" "-SUBJECT=Job Failure" Dear #FNAME, I'm afraid you'll have to take a look at ODI Operator, because session <%=snpRef.getSession("SESS_NO")%> has just failed! -Admin</pre>
----------------	---

The "-TO" parameter will be substituted by the value coming from the "Email" column of your source SELECT statement. The "OdiSendMail" command will therefore be triggered for every operator registered in the "Operators" table.

11.1.3 Using Procedures

A procedure can be used in the following ways:

- [Executing the Procedure](#) directly in Designer Navigator for testing its execution.
- [Using a Procedure in a Package](#) along with mappings and other development artifacts for building a data integration workflow.
- [Generating a Scenario for a Procedure](#) for launching only this procedure in a run-time environment.

11.1.3.1 Executing the Procedure

To run a procedure:

1. In the **Project** view of the Designer Navigator, select the procedure you want to execute.
2. Right-click and select **Run**.
3. In the **Run** dialog, set the execution parameters. Refer to "Execution Parameters" in *Administering Oracle Data Integrator* for more information.
4. Click **OK**.
5. The **Session Started Window** appears.
6. Click **OK**.

Note: During this execution the Procedure uses the option values set on the Options tab of the Procedure editor.

11.1.3.2 Using a Procedure in a Package

Procedures can be used as package steps. Refer to ["Adding a Procedure step"](#) on page 7-5 for more information on how to execute a procedure in a package step. Note that if you use a procedure in a package step, the procedure is not a copy of the procedure you created but a link to it. If this procedure is modified outside of the package, the package using the procedure will be changed, too.

Note: If you don't want to use the option values set on the Options tab of the Procedure, set the new options values directly in the Options tab of the Procedure step.

11.1.3.3 Generating a Scenario for a Procedure

It is possible to generate a scenario to run a procedure in production environment, or to schedule its execution without having to create a package using this procedure. The generated scenario will be a scenario with a single step running this procedure. How to generate a scenario for a procedure is covered in ["Generating a Scenario"](#) on page 12-2.

11.1.4 Encrypting and Decrypting Procedures

Encrypting a Knowledge Module (KM) or a procedure allows you to protect valuable code. An encrypted KM or procedure can neither be read nor modified if it is not decrypted. The commands generated in the log by an Encrypted KM or procedure are also unreadable.

Oracle Data Integrator uses a DES Encryption algorithm based on a personal encryption key. This key can be saved in a file and reused to perform encryption or decryption operations.

WARNING: There is no way to decrypt an encrypted KM or procedure without the encryption key. It is therefore strongly advised to keep this key in a safe location. It is also advised to use a unique key for all the developments.

The steps for encrypting and decrypting procedures are identical to the steps for encrypting and decrypting knowledge modules. Follow the instructions in ["Encrypting and Decrypting a Knowledge Module"](#) on page 6-9

11.2 Working with Variables

This section provides an introduction to variables and describes how to create and use variables in Oracle Data Integrator. This section contains the following topics:

- [Introduction to Variables](#)
- [Creating Variables](#)
- [Using Variables](#)

11.2.1 Introduction to Variables

A **variable** is an object that stores a single value. This value can be a string, a number or a date. The variable value is stored in Oracle Data Integrator. It can be used in several places in your projects, and its value can be updated at run-time.

Depending on the variable type, a variable can have the following characteristics:

- It has a default value defined at creation time.
- Its value can be passed as a parameter when running a scenario using the variable.
- Its value can be refreshed with the result of a statement executed on one of your data servers. For example, it can retrieve the current date and time from a database.
- Its value can be set or incremented in package steps.

- Its value can be tracked from the initial value to the value after executing each step of a session. See "[Tracking Variables and Sequences](#)" on page 11-21 for more information.
- It can be evaluated to create conditions and branches in packages.
- It can be used in the expressions and code of mappings, procedures, steps,...

Variables can be used in any expression (SQL or others), as well as within the metadata of the repository. A variable is resolved when the command containing it is executed by the agent or the graphical interface.

A variable can be created as a **global** variable or in a **project**. This defines the variable scope. Global variables can be used in all projects, while project variables can only be used within the project in which they are defined.

The variable scope is detailed in "[Using Variables](#)" on page 11-14.

The following section describes how to create and use variables.

11.2.2 Creating Variables

To create a variable:

1. In Designer Navigator select the **Variables** node in a project or the **Global Variables** node in the **Global Objects** view.
2. Right-click and select **New Variable**. The Variable Editor opens.
3. Specify the following variable parameters:

Properties	Description
Name	Name of the variable, in the form it will be used. This name should not contain characters that could be interpreted as word separators (blanks, etc.) by the technologies the variable will be used on. Variable names are case-sensitive. That is, "YEAR" and "year" are considered to be two different variables. The variable name is limited to a length of 400 characters.
Datatype	Type of variable: <ul style="list-style-type: none"> ■ Alphanumeric (Text of max 255 char, including text representing an integer or decimal value) ■ Date (This format is the standard ISO date and time format: <code>YYYY-MM-DDThh:mm:ssZ</code> where the capital letter T is used to separate the date and time components. For example: <code>2011-12-30T13:49:02</code> represents 49 minutes and two seconds after one o'clock in the afternoon of 2011-12-30.) ■ Numeric (Integer, Maximum 10 digits (if variable refreshed as decimal, decimal part will be truncated)) ■ Text (Unlimited length)
Keep History	This parameter shows the length of time the value of a variable is kept for: <ul style="list-style-type: none"> ■ No History: The value of the variable is kept in memory for a whole session. ■ Latest value: Oracle Data Integrator stores in its repository the latest value held by the variable. ■ All values: Oracle Data Integrator keeps a history of all the values held by this variable.

Properties	Description
Secure Value	Select Secure Value if you do not want the variable to be recorded. This is useful when the variable contains passwords or other sensitive data. If Secure Value is selected: <ul style="list-style-type: none"> ■ The variable will never be tracked: it will be displayed unresolved in the source or target code, it will not be tracked in the repository, and it will not be historized. ■ The Keep History parameter is automatically set to No History and cannot be edited.
Default Value	The value assigned to the variable by default.
Description	Detailed description of the variable

4. If you want the variable's value to be set by a query:
 - a. Select the **Refreshing** tab.
 - b. Select the logical **Schema** where the command will be executed, then edit the command text in the language of the schema's technology. You can use the Expression Editor for editing the command text. It is recommended to use Substitution methods such as *getObjectName* in the syntax of your query expression.
 - c. Click **Testing query on the DBMS** to check the syntax of your expression.
 - d. Click **Refresh** to test the variable by executing the query immediately. If the Keep History parameter is set to *All Values* or *Latest Value*, you can view the returned value on the History tab of the Variable editor. See "[Notes on Refreshing a Variable Value](#)" for more information on how the value of the variable is calculated.
5. From the **File** menu, click **Save**.

The variable appears in the **Projects** or **Global Objects** sections in Designer Navigator.

Tip: It is advised to use the Expression Editor when you refer to variables. By using the Expression Editor, you can avoid the most common syntax errors. For example, when selecting a variable in the Expression Editor, the variable name will be automatically prefixed with the correct code depending on the variable scope. Refer to "[Variable scope](#)" on page 11-14 for more information on how to refer to your variables.

Notes on Refreshing a Variable Value

- A *numeric session variable* may be defined with no default value. If the session variable also does not have any prior value persisted in the repository, the variable value is considered to be undefined. When such a numeric session variable is queried for its value, for example during a refresh, ODI returns 0 as the result.
- A *non-numeric session variable* (for example: date, alphanumeric, or text) that is defined with no default value will generate an ODI-17506: Variable has no value: <var_name> error when such a variable is queried for its value.
- *Load Plan variables* do not have a default or persisted value. At startup, Load Plans do not take into account the default value of a variable, or the historized/latest value of a variable in the execution context. The value of the variable is either the one specified when starting the Load Plan, or the value set/refreshed within the Load Plan. If a Load Plan variable is not passed as a start up value, the Load Plan

variable's start up value is considered undefined. And if the variable is not refreshed or overwritten in a Load Plan step, the variable's value in the step is also undefined. A numeric Load Plan variable with an undefined value behaves the same as a numeric session variable, for example 0 will be returned when it is queried for its value. See ["Working with Variables in Load Plans"](#) on page 13-15 for more information.

- For *non-numeric Load Plan variables*, there is a limitation in the current ODI repository design that they cannot be distinguished between having an undefined value and a null value. Therefore, non-numeric Load Plan variables with undefined value are currently treated by ODI as having a null value.
- If a *session variable* or a *Load Plan variable* having a null value is referenced in a command or in an expression, for example a SQL text, an empty string (" "), a string with 0 length without the double quotes) will be used as the value for the variable reference in the text.

11.2.3 Using Variables

Using Variables is highly recommended to create reusable packages or packages with a complex conditional logic, mappings and procedures. Variables can be used everywhere within ODI. Their value can be stored persistently in the ODI Repository if their *Keep History* parameter is set to **All values** or **Latest value**. Otherwise, if their *Keep History* parameter is set to **No History**, their value will only be kept in the memory of the agent during the execution of the current session.

This section provides an overview of how to use variables in Oracle Data Integrator. Variables can be used in the following cases:

- [Using Variables in Packages](#)
- [Using Variables in Mappings](#)
- [Using Variables in Object Properties](#)
- [Using Variables in Procedures](#)
- [Using Variables within Variables](#)
- [Using Variables in the Resource Name of a Datastore](#)
- [Passing a Variable to a Scenario](#)
- [Generating a Scenario for a Variable](#)
- [Tracking Variables and Sequences](#)

Variable scope

Use the Expression Editor to refer to your variables in Packages, mappings, and procedures. When you use the Expression Editor the variables are retrieved directly from the repository.

You should only manually prefix variable names with GLOBAL or the PROJECT_CODE, when the Expression Editor is not available.

Referring to variable MY_VAR in your objects should be done as follows:

- #MY_VAR: With this syntax, the variable must be in the same project as the object referring to it. Its value will be substituted. To avoid ambiguity, consider using fully qualified syntax by prefixing the variable name with the project code.
- #MY_PROJECT_CODE.MY_VAR: Using this syntax allows you to use variables by explicitly stating the project that contains the variable. It prevents ambiguity when

2 variables with the same name exist for example at global and project level. The value of the variable will be substituted at runtime.

- **#GLOBAL.MY_VAR:** This syntax allows you to refer to a global variable. Its value will be substituted in your code. Refer to section Global Objects for details.
- Using ":" instead of "#": You can use the variable as a SQL bind variable by prefixing it with a colon rather than a hash. However this syntax is subject to restrictions as it only applies to SQL DML statements, not for OS commands or ODI API calls and using the bind variable may result in performance loss. It is advised to use ODI variables prefixed with the '#' character to ensure optimal performance at runtime.
 - When you reference an ODI Variable prefixed with the ':' character, the name of the Variable is NOT substituted when the RDBMS engine determines the execution plan. The variable is substituted when the RDBMS executes the request. This mechanism is called **Binding**. If using the binding mechanism, it is not necessary to enclose the variables which store strings between delimiters (such as quotes) because the RDBMS is expecting the same type of data as specified by the definition of the column for which the variable is used.

For example, if you use the variable `TOWN_NAME = :GLOBAL.VAR_TOWN_NAME` the VARCHAR type is expected.
 - When you reference an ODI variable prefixed with the "#" character, ODI substitutes the name of the variable by the value before the code is executed by the technology. The variable reference needs to be enclosed in single quote characters, for example `TOWN = '#GLOBAL.VAR_TOWN'`. This reference mode of the variable works for OS commands, SQL, and ODI API calls.

11.2.3.1 Using Variables in Packages

Variables can be used in packages for different purposes:

- **Declaring a variable:** When a variable is used in a package (or in certain elements of the topology that are used in the package), it is strongly recommended that you insert a Declare Variable step in the package. This step explicitly declares the variable in the package. How to create a Declare Variable step is covered in ["Adding a Variable step"](#) on page 7-5. Other variables that you explicitly use in your packages for setting, refreshing or evaluating their values do not need to be declared.
- **Refreshing a variable from its SQL SELECT statement:** A Refresh Variable step allows you to re-execute the command or query that computes the variable value. How to create a Refresh Variable step is covered in ["Adding a Variable step"](#) on page 7-5.
- **Assigning the value of a variable:** A Set Variable step of type Assign sets the current value of a variable.

In Oracle Data Integrator you can assign a value to a variable in the following ways:

- **Retrieving the variable value from a SQL SELECT statement:** When creating your variable, define a SQL statement to retrieve its value. For example, you can create a variable `NB_OF_OPEN_ORDERS` and set its SQL statement to:

```
select COUNT(*) from <%=odiRef.getObjectName("L", "ORDERS", "D")%>
where STATUS = 'OPEN'.
```

Then in your package, you will simply drag and drop your variable and select the "Refresh Variable" option in the Properties panel. At runtime, the ODI

agent will execute the SQL statement and assign the first returned value of the result set to the variable.

- **Explicitly setting the value in a package:** You can also manually assign a value to your variable for the scope of your package. Simply drag and drop your variable into your package and select the "Set Variable" and "Assign" options in the Properties panel as well as the value you want to set.
- **Incrementing the value:** Incrementing only applies to variables defined with a numeric data type. Drag and drop your numeric variable into the package and select the "Set Variable" and "Increment" options in the Properties panel as well as the desired increment. Note that the increment value can be positive or negative.
- **Assigning the value at runtime:** When you start a scenario generated from a package containing variables, you can set the values of its variables. You can do that in the StartScenario command by specifying the VARIABLE=VALUE list. Refer to the API command "OdiStartLoadPlan" in *Oracle Data Integrator Tool Reference*, and "Executing a Scenario from a Command Line" in *Administering Oracle Data Integrator*.

How to create a Assign Variable step is covered in ["Adding a Variable step"](#) on page 7-5.

- **Incrementing a numeric value:** A Set Variable step of type Increment increases or decreases a numeric value by the specified amount. How to create a Set Variable step is covered in ["Adding a Variable step"](#) on page 7-5.
- **Evaluating the value for conditional branching:** An Evaluate Variable step acts like an IF-ELSE step. It tests the current value of a variable and branches in a package depending on the result of the comparison. For example, you can choose to execute mappings A and B of your package only if variable EXEC_A_AND_B is set to "YES", otherwise you would execute mappings B and C. To do this, you would simply drag and drop the variable in your package diagram, and select the "Evaluate Variable" type in the properties panel. Evaluating variables in a package allows great flexibility in designing reusable, complex workflows. How to create an Evaluate Variable step is covered in ["Adding a Variable step"](#) on page 7-5.

11.2.3.2 Using Variables in Mappings

Variables can be used in mappings in two different ways:

1. As a value for a textual option of a Knowledge Module.
2. In all Oracle Data Integrator expressions, such as filter conditions and join conditions.

To substitute the value of the variable into the text of an expression, precede its name by the '#' character. The agent or the graphical interface will substitute the value of the variable in the command before executing it.

The following example shows the use of a global variable named 'YEAR':

```
Update CLIENT set LASTDATE = sysdate where DATE_YEAR = '#GLOBAL.YEAR' /* DATE_YEAR
is CHAR type */
Update CLIENT set LASTDATE = sysdate where DATE_YEAR = #GLOBAL.YEAR /* DATE_YEAR
is NUMERIC type */
```

The "bind variable" mechanism of the SQL language can also be used, however, this is less efficient, because the relational database engine does not know the value of the variable when it constructs the execution plan for the query. To use this mechanism,

precede the variable by the ':' character, and make sure that the datatype being searched is compatible with that of the variable. For example:

```
update CLIENT set LASTDATE = sysdate where DATE_YEAR =:GLOBAL.YEAR
```

The "bind variable" mechanism must be used for Date type variables that are used in a filter or join expression. The following example shows a filter:

```
SRC.END_DATE > :SYSDATE_VAR
```

where the variable SYSDATE_VAR is a "Date" type variable with the refresh query `select sysdate from dual`

If the substitution method is used for a date variable, you need to convert the string into a date format using the RDBMS specific conversion function.

You can drag-and-drop a variable into most expressions with the Expression Editor.

Table 11-3 Examples of how to use Variables in Mappings

Type	Expression	
Attribute Expression	'#PRODUCT_PREFIX' PR.PRODUCT_CODE	Concatenates the current project's product prefix variable with the product code. As the value of the variable is substituted, you need to enclose the variable with single quotes because it returns a string.
Join Condition	CUS.CUST_ID = #DEMO.UID * 1000 + FF.CUST_NO	Multiply the value of the UID variable of the DEMO project by 1000 and add the CUST_NO column before joining it with the CUST_ID column.
Filter Condition	ORDERS.QTY between #MIN_QTY and #MAX_QTY	Filter orders according to the MIN_QTY and MAX_QTY thresholds.
Option Value	TEMP_FILE_NAME: #DEMO.FILE_NAME	Use the FILE_NAME variable as the value for the TEMP_FILE_NAME option.

11.2.3.3 Using Variables in Object Properties

It is also possible to use variables as substitution variables in graphical module fields such as resource names or schema names in the topology. You must use the fully qualified name of the variable (Example: #GLOBAL.MYTABLENAME) directly in the Oracle Data Integrator graphical module's field.

Using this method, you can parameterize elements for execution, such as:

- The physical names of files and tables (Resource field in the datastore) or their location (Physical schema's schema (data) in the topology)
- Physical Schema
- Data Server URL

11.2.3.4 Using Variables in Procedures

You can use variables anywhere within your procedures' code as illustrated in the [Table 11-4](#).

Table 11–4 Example of how to use Variables in a Procedure

Step ID:	Step Type	Step Code	Description
1	SQL	Insert into #DWH.LOG_TABLE_NAME Values (1, 'Loading Step Started', current_date)	Add a row to a log table that has a name only known at runtime
2	Jython	f = open('#DWH.LOG_FILE_NAME', 'w') f.write('Inserted a row in table %s' % ('#DWH.LOG_TABLE_NAME')) f.close()	Open file defined by LOG_ FILE_NAME variable and write the name of the log table into which we have inserted a row.

You should consider using options rather than variables whenever possible in procedures. Options act like input parameters. Therefore, when executing your procedure in a package you would set your option values to the appropriate values.

In the example of [Table 11–4](#), you would write Step 1's code as follows:

```
Insert into <%=snpRef.getOption("LogTableName")%>
Values (1, 'Loading Step Started', current_date)
```

Then, when using your procedure as a package step, you would set the value of option LogTableName to #DWH.LOG_TABLE_NAME.

Note that when using Groovy scripting, you need to enclose the variable name in double quotes ("), for example "#varname" and "#GLOBAL.varname", otherwise the variables are not substituted with the ODI variable value.

11.2.3.5 Using Variables within Variables

It is sometimes useful to have variables depend on other variable values as illustrated in [Table 11–5](#).

Table 11–5 Example of how to use a variable within another variable

Variable Name	Variable Details	Description
STORE_ID	Alphanumeric variable. Passed as a parameter to the scenario	Gives the ID of a store
STORE_NAME	Alphanumeric variable. SELECT statement: Select name From <%=odiRef.getObjectName("L", "STORES", "D")%> Where id='#DWH.STORE_ID' '#DWH.STORE_CODE'	The name of the current store is derived from the Stores table filtered by the value returned by the concatenation of the STORE_ID and STORE_CODE variables.

In [Table 11–5](#), you would build your package as follows:

1. Drag and drop the STORE_ID variable to declare it. This would allow you to pass it to your scenario at runtime.
2. Drag and drop the STORE_NAME variable to refresh its value. When executing this step, the agent will run the select query with the appropriate STORE_ID value. It will therefore retrieve the corresponding STORE_NAME value.
3. Drag and drop the other mappings or procedures that use any of these variables.

Note that the "bind variable" mechanism must be used to define the refresh query for a "date" type variable that references another "date" type variable. For example:

```
VAR1 "Date" type variable has the refresh query select sysdate from dual
```

```
VAR_VAR1 "Date" type variable must have the refresh query select :VAR1 from dual
```

11.2.3.6 Using Variables in the Resource Name of a Datastore

You may face some situations where the names of your source or target datastores are dynamic. A typical example of this is when you need to load flat files into your Data Warehouse with a file name composed of a prefix and a dynamic suffix such as the current date. For example the order file for March 26 would be named `ORD2009.03.26.dat`.

Note that you can only use variables in the resource name of a datastore in a scenario when the variable has been previously declared.

To develop your loading mappings, you would follow these steps:

1. Create the `FILE_SUFFIX` variable in your DWH project and set its `SQL SELECT` statement to select `current_date` (or any appropriate date transformation to match the actual file suffix format)
2. Define your `ORDERS` file datastore in your model and set its resource name to: `ORD#DWH.FILE_SUFFIX.dat`.
3. Use your file datastore normally in your mappings.
4. Design a package as follows:
 1. Drag and drop the `FILE_SUFFIX` variable to refresh it.
 2. Drag and drop all mappings that use the `ORDERS` datastore.

At runtime, the source file name will be substituted to the appropriate value.

Note: The variable in the datastore resource name must be fully qualified with its project code.

When using this mechanism, it is not possible to view the data of your datastore from within Designer.

11.2.3.7 Using Variables in a Server URL

There are some cases where using contexts for different locations is less appropriate than using variables in the URL definition of your data servers. For example, when the number of sources is high (> 100), or when the topology is defined externally in a separate table. In these cases, you can refer to a variable in the URL of a server's definition.

Suppose you want to load your warehouse from 250 source applications - hosted in Oracle databases - used within your stores. Of course, one way to do it would be to define one context for every store. However, doing so would lead to a complex topology that would be difficult to maintain. Alternatively, you could define a table that references all the physical information to connect to your stores and use a variable in the URL of your data server's definition. [Example 11-3](#) illustrates how you would implement this in Oracle Data Integrator:

Example 11-3 Referring to a Variable in the URL of a Server's Definition

1. Create a `StoresLocation` table as follows:

StoreID	Store Name	Store URL	IsActive
1	Denver	10.21.32.198:1521:OR A1	YES
2	San Francisco	10.21.34.119:1525:SA NF	NO
3	New York	10.21.34.11:1521:NY	YES
...

2. Create three variables in your EDW project:

- STORE_ID: takes the current store ID as an input parameter
- STORE_URL: refreshes the current URL for the current store ID with SELECT statement: `select StoreUrl from StoresLocation where StoreId = #EDW.STORE_ID`
- STORE_ACTIVE: refreshes the current activity indicator for the current store ID with SELECT statement: `select IsActive from StoresLocation where StoreId = #EDW.STORE_ID`

3. Define one physical data server for all your stores and set its JDBC URL to:

```
jdbc:oracle:thin:@#EDW.STORE_URL
```

4. Define your package for loading data from your store.

The input variable STORE_ID will be used to refresh the values for STORE_URL and STORE_ACTIVE variables from the StoresLocation table. If STORE_ACTIVE is set to "YES", then the next 3 steps will be triggered. The mappings refer to source datastores that the agent will locate according to the value of the STORE_URL variable.

To start such a scenario on Unix for the New York store, you would issue the following operating system command:

```
startscen.sh LOAD_STORE 1 PRODUCTION "EDW.STORE_ID=3"
```

If you want to trigger your LOAD_STORE scenario for all your stores in parallel, you would simply need to create a procedure with a single SELECT/action command as follows:

Source Technology	Oracle (technology of the data server containing the StoresLocation table).
Source Logical Schema	Logical schema containing the StoresLocation table.
Source Command	<code>Select StoreId From StoresLocation</code>
Target Technology	ODITools
Target Logical Schema	None

The LOAD_STORE scenario will then be executed for every store with the appropriate STORE_ID value. The corresponding URL will be set accordingly.

Refer to "[Binding Source and Target Data](#)" on page 11-8 and "Managing Agents" in *Administering Oracle Data Integrator* for further details.

11.2.3.8 Using Variables in On Connect/Disconnect Commands

Variables can be used in the On connect/Disconnect SQL commands. See "Creating a Data Server (Advanced Settings)" in *Administering Oracle Data Integrator* for more information.

11.2.3.9 Passing a Variable to a Scenario

It is also possible to pass a variable to a scenario in order to customize its behavior. To do this, pass the name of the variable and its value on the OS command line which executes the scenario. For more information, see "Executing a Scenario from a Command Line" in *Administering Oracle Data Integrator*.

11.2.3.10 Generating a Scenario for a Variable

It is possible to generate a single step scenario for refreshing a variable.

How to generate a scenario for a variable is covered in "Generating a Scenario" on page 12-2.

11.2.3.11 Tracking Variables and Sequences

Tracking variables and sequences allows to determine the actual values of Oracle Data Integrator user variables that were used during an executed session. With the variable tracking feature you can also determine whether the variable was used in a source/target operation or an internal operation such as an Evaluate step.

Variable tracking takes place and is configured at several levels:

- When defining a variable, you can select **Secure Value** if you do not want the variable to be recorded. This is useful when the variable contains passwords or other sensitive data. If **Secure Value** is selected, the variable will never be tracked: It will be displayed unresolved in the source or target code, it will not be tracked in the repository, and it will not be historized. See "Creating Variables" on page 11-12 for more information.
- When executing or restarting a session, select **Log Level 6** in the Execution or Restart Session dialog to enable variable tracking. Log level 6 has the same behavior as log level 5, but with the addition of variable tracking.
- When reviewing the execution results in Operator Navigator, you can:
 - View tracked variables and sequences in the **Variables and Sequence Values** section of the Session Step or Session Task Editor.
 - Review the source/target operations of an execution in the Session Task Editor. In the Code tab of the Session Task Editor, click **Show/Hide Values** to display the code with resolved variable and sequence values. Note that only variables in substitution mode (#VARIABLE) can be displayed with resolved variable values and that if the variable values are shown, the code becomes read-only.

Tracking variables and sequences is useful for debugging purposes. See "Handling Failed Sessions" in *Administering Oracle Data Integrator* for more information on how to analyze errors in Operator Navigator and activate variable tracking.

Variable tracking is available in ODI Studio and ODI Console sessions.

Note the following when tracking variables in Oracle Data Integrator:

- Each value taken by a variable in a session can be tracked.

- The values of all tracked variables can be displayed at step and task level. This includes when a variable is modified by a step or a task, the Step or Task Editor displays the name and the new value of the variable.
- The source and target code for a step or task can be viewed either with resolved variable and sequence values or with hidden variable values that display the variable and sequence names. Note that if the variable values are shown, the code becomes read-only.
- Variables that are defined as **Secure Value**, such passwords, are never displayed in the resolved code or variable list. A secure variable does not persist any value in the repository, even if it is refreshed. Note also that the refresh of a secure variable does not work across two sessions.
- When a session is purged, all variable values tracked for that session are purged along with the session.
- Bind variables (:VARIABLE_NAME) and native sequences (:<SEQUENCE_NAME>_NEXTVAL) will not have their values resolved in the source and target code contents; only substituted variables and sequences (#VARIABLE_NAME and #<SEQUENCE_NAME>_NEXTVAL) will be resolved.
- Tracked values are exported and imported as part of a session when the session is exported or imported.

11.3 Working with Sequences

This section provides an introduction to sequences and describes how to create and use sequences in Oracle Data Integrator. This section includes the following topics:

- [Introduction to Sequences](#)
- [Creating Sequences](#)
- [Using Sequences and Identity Columns](#)
- [Sequence Enhancements](#)

11.3.1 Introduction to Sequences

A **Sequence** is a variable that increments itself automatically each time it is used. Between two uses, the value can be stored in the repository or managed within an external RDBMS table. Sequences can be strings, lists, tuples or dictionaries.

Oracle Data Integrator sequences are intended to map native sequences from RDBMS engines, or to simulate sequences when they do not exist in the RDBMS engine. Non-native sequences' values can be stored in the Repository or managed within a cell of an external RDBMS table.

A sequence can be created as a global sequence or in a project. Global sequences are common to all projects, whereas project sequences are only available in the project where they are defined.

Oracle Data Integrator supports three types of sequences:

- **Standard sequences**, whose current values are stored in the Repository.
- **Specific sequences**, whose current values are stored in an RDBMS table cell. Oracle Data Integrator reads the value, locks the row (for concurrent updates) and updates the row after the last increment.
- **Native sequence**, that maps a RDBMS-managed sequence.

Note the following on standard and specific sequences:

- Oracle Data Integrator locks the sequence when it is being used for multi-user management, but does not handle the sequence restart points. In other words, the SQL statement ROLLBACK does not return the sequence to its value at the beginning of the transaction.
- Oracle Data Integrator standard and specific sequences were developed to compensate for their absence on some RDBMS. If native sequences exist, they should be used. This may prove to be faster because it reduces the dialog between the agent and the database.
- The value of standard and specific sequences (#<SEQUENCE_NAME>_NEXTVAL) can be tracked. A side effect that only happens to tracking native sequence is that the native sequence value is incremented once more when it is accessed for tracking purpose. See "[Tracking Variables and Sequences](#)" on page 11-21 for more information.

The following sections describe how to create and use sequences.

11.3.2 Creating Sequences

The procedure for creating sequences vary depending on the sequence type. Refer to the corresponding section:

- [Creating Standard Sequences](#)
- [Creating Specific Sequences](#)
- [Creating Native Sequences](#)

11.3.2.1 Creating Standard Sequences

To create a standard sequence:

1. In Designer Navigator select the **Sequences** node in a project or the **Global Sequences** node in the **Global Objects** view.
2. Right-click and select **New Sequence**. The Sequence Editor opens.
3. Enter the sequence **Name**, then select **Standard Sequence**.
4. Enter the **Increment**.
5. From the **File** menu, click **Save**.

The sequence appears in the **Projects** or **Global Objects** section in Designer Navigator.

11.3.2.2 Creating Specific Sequences

Select this option for storing the sequence value in a table in a given data schema.

To create a specific sequence:

1. In Designer Navigator select the **Sequences** node in a project or the **Global Sequences** node in the **Global Objects** view.
2. Right-click and select **New Sequence**. The Sequence Editor opens.
3. Enter the sequence **Name**, then select **Specific Sequence**.
4. Enter the **Increment** value.
5. Specify the following sequence parameters:

Schema	Logical schema containing the sequences table
Table	Table containing the sequence value
Column	Name of the column containing the sequence value.
Filter to retrieve a single row	Type in a Filter which will allow Oracle Data Integrator to locate a specific row in the table when the sequence table contains more than one row. This filter picks up the SQL syntax of the data server. For example: <code>CODE_TAB = '3'</code> You can use the Expression Editor to edit the filter. Click Testing query on the DBMS to check the syntax of your expression.

6. From the **File** menu, click **Save**.

The sequence appears in the **Projects** or **Global Objects** section in Designer Navigator.

Note: When Oracle Data Integrator wants to access the specific sequence value, the query executed on the schema will be `SELECT column FROM table WHERE filter`.

11.3.2.3 Creating Native Sequences

Select this option if your sequence is implemented in the database engine. Position and increment are fully handled by the database engine.

To create a native sequence:

1. In Designer Navigator select the **Sequences** node in a project or the **Global Sequences** node in the **Global Objects** view.
2. Right-click and select **New Sequence**. The Sequence Editor opens.
3. Enter the sequence **Name**, then select **Native Sequence**.
4. Select the logical **Schema** containing your native sequence.
5. Type in the **Native Sequence Name** or click the browse button to select a sequence from the list pulled from the data server.
6. If you clicked the Browse button, in the **Native Sequence Choice** dialog, select a **Context** to display the list of sequences in this context for your logical schema.
7. Select one of these sequences and click **OK**.
8. From the **File** menu, click **Save**.

The sequence appears in the **Projects** or **Global Objects** tree in Designer Navigator.

11.3.3 Using Sequences and Identity Columns

In order to increment sequences, the data needs to be processed row-by-row by the agent. Therefore, using sequences is not recommended when dealing with large numbers of records. In this case, you would use database-specific sequences such as identity columns in Teradata, IBM DB2, Microsoft SQL Server or sequences in Oracle.

The sequences can be used in all Oracle Data Integrator expressions. For example:

- The Expression property of a component attribute

- The Filter Condition property of a Filter component
- The Join Condition property of a Join component

Sequences can be used either as:

- A substituted value, using the #<SEQUENCE_NAME>_NEXTVAL syntax
- A bind variable in SQL statements, using the :<SEQUENCE_NAME>_NEXTVAL syntax

Using a sequence as a substituted value

A sequence can be used in all statements with the following syntax: #<SEQUENCE_NAME>_NEXTVAL

With this syntax, the sequence value is incremented only once before the command is run and then substituted by its value into the text of the command. The sequence value is the same for all records.

Using a sequence as a bind variable

Only for SQL statements on a target command of a KM or procedure, sequences can be used with the following syntax: :<SEQUENCE_NAME>_NEXTVAL

With this syntax, the sequence value is incremented, then passed as a bind variable of the target SQL command. The sequence value is incremented in each record processed by the command. The behavior differs depending on the sequence type:

- **Native sequences** are always incremented for each processed record.
- **Standard** and **specific sequences** are resolved by the run-time agent and are incremented only when records pass through the agent. The command in a KM or procedure that uses such a sequence must use a SELECT statement on the source command and an INSERT or UPDATE statement on the target command rather than a single INSERT/UPDATE... SELECT in the target command.

Note: A sequence can only increment for each row if:

- The staging area is not on the target
 - You use an LKM SQL Multi-Connect on Access Points
 - You use a multi-technology IKM on the target datastore, such as IKM SQL to SQL Control Append
-
-

For example:

- In the SQL statement `insert into fac select :NO_FAC_NEXTVAL, date_fac, mnt_fac` the value of a **standard** or **specific sequence** will be incremented only once, even if the SQL statement processes 10,000 rows, because the agent does not process each record, but just sends the command to the database engine. A **native sequence** will be incremented for each row.
- To increment the value of a **standard** or **specific sequence** for each row, the data must pass through the agent. To do this, use a KM or procedure that performs a SELECT on the source command and an INSERT on the target command:

```
SELECT date_fac, mnt_fac /* on the source connection */
```

```
INSERT into FAC (ORDER_NO, ORDER_DAT, ORDER_AMNT) values (:NO_FAC_NEXTVAL,
:date_fac, :mnt_fac) /* on the target connection */
```

Sequence Scope

Unlike for variables, you do not need to state the scope of sequences explicitly in code.

11.3.3.1 Tips for Using Standard and Specific Sequences

To make sure that a sequence is updated for each row inserted into a table, each row must be processed by the Agent. To make this happen, follow the steps below:

1. Make the mapping containing the sequence be executed on the target.
2. Set the mapping to be active for inserts only. Updates are not supported for sequences.
3. If you are using an "incremental update" IKM, you should make sure that the update key in use does not contain a column populated with the sequence. For example, if the sequence is used to load the primary key for a datastore, you should use an alternate key as the update key for the mapping.
4. If using Oracle Data Integrator sequences with bind syntax (:<SEQUENCE_NAME>_NEXTVAL), you must configure the data flow such that the IKM transfers all the data through the agent. You can verify this by checking the generated integration step in Operator. It should have separate INSERT and SELECT commands executed on different connections, rather than a single SELECT...INSERT statement.

Limitations of Sequences

Sequences have the following limitations:

- A column mapped with a sequence should not be checked for not null.
- Similarly, static control and flow control cannot be performed on a primary or alternate key that references the sequence.

11.3.3.2 Identity Columns

Certain databases also natively provide identity columns, which are automatically populated with unique, self-incrementing values.

When populating an identity column, you should follow these steps:

1. The mapping loading the identity column should be blank and inactive. It should not be activated for inserts or updates.
2. If you are using "incremental update" IKMs, make sure that the update key in use does not contain the identity column. If the identity column is part of the primary key, you should define an alternate key as the update key for the mapping.

Limitations of Identity Columns

Identity columns have the following limitations:

- Not null cannot be checked for an identity column.
- Static and flow control cannot be performed on a primary or alternate key containing the identity column.

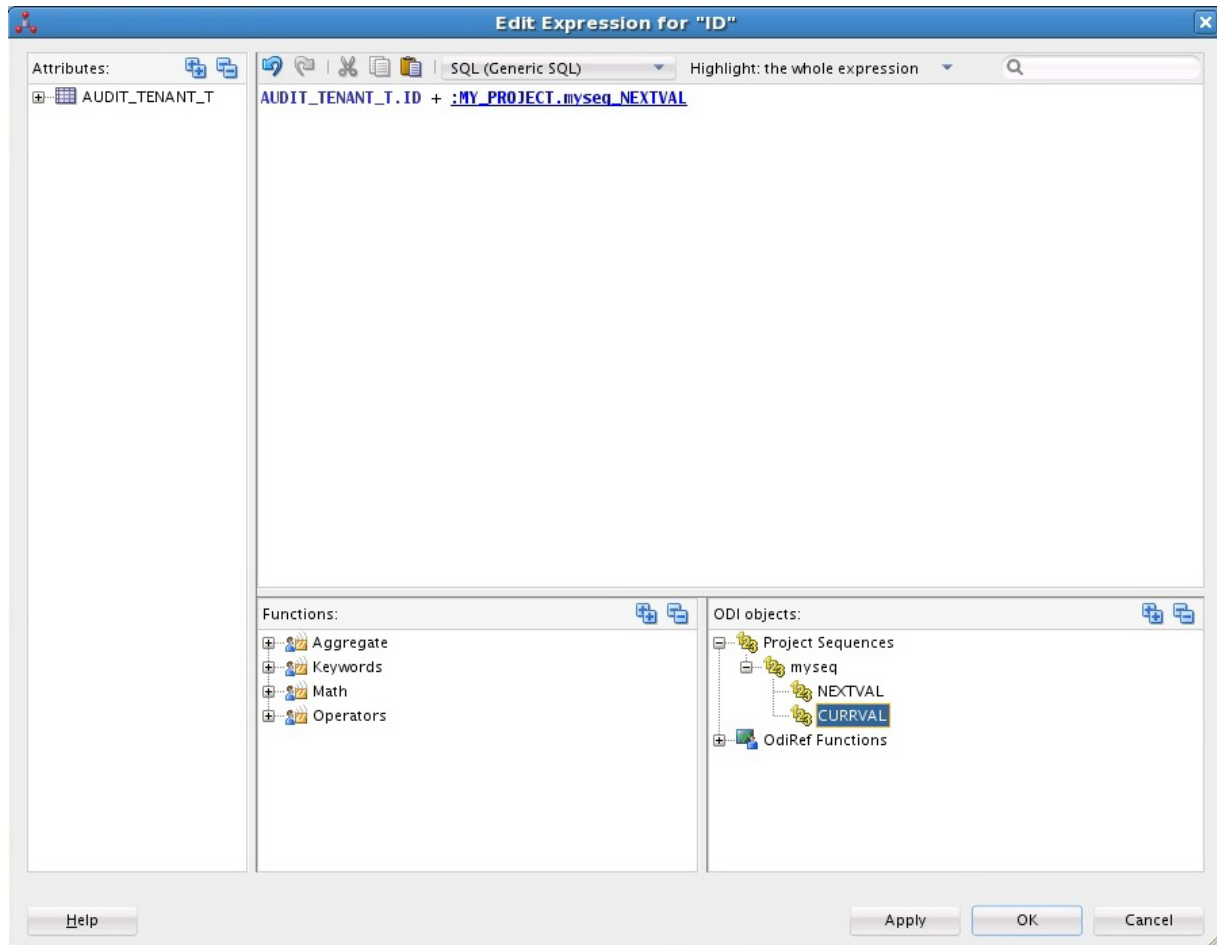
11.3.4 Sequence Enhancements

Sequence in Oracle Data Integrator is enhanced to support the CURRVAL operator. The expression editor now displays the NEXTVAL and CURRVAL operators for each sequence that is listed in the **ODI objects** panel as shown in [Figure 11-1](#).

Before using current value for native sequences:

- Confirm the technology supports native sequence. For example, MySQL does not support native sequence.
- Configure the local sequence current value mask
- Configure the remote sequence current value mask
- Configure the sequence current value in non-bound mode

Figure 11–1 Expression editor enhancement: SEQUENCE



11.4 Working with User Functions

This section provides an introduction to user functions and describes how to create and use user functions in Oracle Data Integrator. This section contains the following topics:

- [Introduction to User Functions](#)
- [Creating User Functions](#)
- [Using User Functions](#)

11.4.1 Introduction to User Functions

User functions are used for defining customized functions that can be used in mappings or procedures. It is recommended to use them in your projects when the

same complex transformation pattern needs to be assigned to different datastores within different mappings. User functions improve code sharing and reusability and facilitate the maintenance and the portability of your developments across different target platforms.

User functions are implemented in one or more technologies and can be used anywhere in mappings, joins, filters and conditions. Refer to "[Using User Functions](#)" on page 11-29.

A function can be created as a **global** function or in a **project**. In the first case, it is common to all projects, and in the second, it is attached to the project in which it is defined.

User functions can call other user functions. A user function cannot call itself recursively.

Note: Aggregate functions are not supported User Functions. The aggregate function code will be created, but the GROUP BY expression will not be generated.

The following sections describe how to create and use user functions.

11.4.2 Creating User Functions

To create a user function:

1. In Designer Navigator select the **User Functions** node in a project or the **Global User Functions** node in the **Global Objects** view.
2. Right-click and select **New User Function**. The User Function Editor opens.
3. Fill in the following fields:
 - **Name:** Name of the user function, for example `NullValue`
 - **Group:** Group of the user function. If you type a group name that does not exist, a new group will be created with this group name when the function is saved.
 - **Syntax:** Syntax of the user function that will appear in the Expression Editor; The arguments of the function must be specified in this syntax, for example `NullValue($(variable), $(default))`
4. From the **File** menu, click **Save**.

The function appears in the **Projects** or **Global Objects** tree in Designer Navigator. Since it has no implementation, it is unusable.

To create an implementation:

1. In Designer Navigator double-click the User Function for which you want to create the implementation. The User Function Editor opens.
2. In the **Implementations** tab of the User Function Editor, click **Add Implementation**. The Implementation dialog opens.
3. In the **Implementation syntax** field, type the code of the implementation, for example `nvl($(variable), $(default))`
4. Check the boxes for the implementation's Linked technologies

5. Check **Automatically include new technologies** if you want the new technologies to use this syntax.
6. Click **OK**.
7. From the **File** menu, click **Save**.

To change an implementation:

1. In the **Implementations** tab of the User Function Editor, select an implementation, then click **Edit**.
2. In the **Implementations** tab of the user function, select an implementation, then click **Edit Implementation**. The Implementation dialog opens.
3. Change the Implementation syntax and the Linked technologies of this implementation
4. Check **Automatically include new technologies** if you want the new technologies to use this syntax.
5. Click **OK**.
6. From the **File** menu, click **Save**.

To remove an implementation:

In the implementations tab of the user function, select an implementation, then click **Delete Implementation**.

To make a user function available for a specific technology:

1. Open the Technology editor of the specific technology.
2. In the **Language** column, select the language of the technology.
3. Select **Default**.
4. Make sure that you have selected the corresponding technology from the Technology type list on the Definition tab. The Oracle Data Integrator API does not work with user functions.

11.4.3 Using User Functions

The user functions can be used in all Oracle Data Integrator expressions. For example:

- The Expression property of a component attribute
- The Filter Condition property of a Filter component
- The Join Condition property of a Join component

A user function can be used directly by specifying its syntax, for example:

```
NullValue(CITY_NAME, 'No City')
```

User functions are implemented in one or more technologies. For example, the Oracle `nvl(VARIABLE, DEFAULT_VALUE)`, function - which returns the value of `VARIABLE`, or `DEFAULT_VALUE` if `VARIABLE` is null - has no equivalent in all technologies and must be replaced by the formula:

```
case when VARIABLE is null
then DEFAULT_VALUE
else VARIABLE
end
```

With user functions, it is possible to declare a function called `NullValue(VARIABLE,DEFAULT_VALUE)` and to define two implementations for the

syntax above. When executing, depending on the technology on which the function will be executed, the NullValue function will be replaced by one syntax or the other.

The next example illustrates how to implement a user function that would be translated into code for different technologies:

Suppose you want to define a function that, given a date, gives you the name of the month. You want this function to be available for your mappings when executed on Oracle, Teradata or Microsoft SQL Server. [Table 11–6](#) shows how to implement this as a user function.

Table 11–6 User Function Translated into Code for Different Technologies (Example 1)

Function Name	GET_MONTH_NAME
Function Syntax	GET_MONTH_NAME(\$(date_input))
Description	Retrieves the month name from a date provided as date_input
Implementation for Oracle	Initcap(to_char(\$(date_input), 'MONTH'))
Implementation for Teradata	<pre> case when extract(month from \$(date_input)) = 1 then 'January' when extract(month from \$(date_input)) = 2 then 'February' when extract(month from \$(date_input)) = 3 then 'March' when extract(month from \$(date_input)) = 4 then 'April' when extract(month from \$(date_input)) = 5 then 'May' when extract(month from \$(date_input)) = 6 then 'June' when extract(month from \$(date_input)) = 7 then 'July' when extract(month from \$(date_input)) = 8 then 'August' when extract(month from \$(date_input)) = 9 then 'September' when extract(month from \$(date_input)) = 10 then 'October' when extract(month from \$(date_input)) = 11 then 'November' when extract(month from \$(date_input)) = 12 then 'December' end </pre>
Implementation for Microsoft SQL	datename(month, \$(date_input))

You can now use this function safely in your mappings for building attribute expressions, filter conditions, and join conditions. Oracle Data Integrator will generate the appropriate code depending on the execution location of your expression.

Another example of a user function translated into code for different technologies is defining the following mapping:

`substring(GET_MONTH_NAME(CUSTOMER.LAST_ORDER_DATE), 1, 3)`, Oracle Data Integrator will generate code similar to the following, depending on your execution technology:

Table 11–7 User Function Translated into Code for Different Technologies (Example 2)

Implementation for Oracle	<code>substring(Initcap(to_char(CUSTOMER.LAST_ORDER_DATE 'MONTH')) , 1, 3)</code>
Implementation for Teradata	<code>substring(case when extract(month from CUSTOMER.LAST_ORDER_DATE) = 1 then 'January'when extract(month from CUSTOMER.LAST_ORDER_DATE) = 2 then 'February'...end, 1, 3)</code>
Implementation for Microsoft SQL	<code>substring(datename(month, CUSTOMER.LAST_ORDER_DATE) , 1, 3)</code>

A function can be created as a **global** function or in a **project**. In the first case, it is common to all projects, and in the second, it is attached to the project in which it is defined.

User functions can call other user functions.

Using Scenarios

This chapter describes how to work with scenarios. A **scenario** is designed to put a source component (mapping, package, procedure, variable) into production. A scenario results from the generation of code (SQL, shell, etc.) for this component.

This chapter includes the following sections:

- [Introduction to Scenarios](#)
- [Generating a Scenario](#)
- [Regenerating a Scenario](#)
- [Generating a Group of Scenarios](#)
- [Controlling Concurrent Execution of Scenarios and Load Plans](#)
- [Exporting Scenarios](#)
- [Importing Scenarios in Production](#)
- [Encrypting and Decrypting a Scenario](#)

12.1 Introduction to Scenarios

When a component is finished and tested, you can generate the **scenario** corresponding to its actual state. This operation takes place in the Designer Navigator.

The scenario code (the language generated) is frozen, and all subsequent modifications of the components which contributed to creating it will not change it in any way.

It is possible to generate scenarios for packages, procedures, mappings, or variables. Scenarios generated for procedures, mappings, or variables are single step scenarios that execute the procedure, mapping, or refresh the variable.

Scenario variables are variables used in the scenario that should be set when starting the scenario to parameterize its behavior.

Once generated, the scenario is stored inside the work repository. The scenario can be exported, and then imported to another repository (remote or not) and used in different contexts. A scenario can only be created from a development work repository, but can be imported into both development and execution work repositories.

Scenarios appear in both the Operator and Designer Navigators, in the Load Plans and Scenarios section. Scenarios can also appear within a project in the Projects section of the Designer navigator.

Scenarios can also be versioned. See [Chapter 19, "Using Version Control \(Legacy Mode\)"](#) for more information.

Scenarios can be launched from a command line, from the Oracle Data Integrator Studio and can be scheduled using the built-in scheduler of the run-time agent or an external scheduler. Scenario execution and scheduling scenarios is covered in "Running Integration Processes" in *Administering Oracle Data Integrator*.

12.2 Generating a Scenario

Generating a scenario for an object compiles the code for this object for deployment and execution in a production environment.

To generate a scenario:

1. In Designer Navigator double-click the Package, Mapping, Procedure or Variable under the project for which you want to generate the scenario. The corresponding Object Editor opens. Then, on the **ODI** menu, select **Generate** and then **Scenario**. The New Scenario dialog appears.

Alternatively, from the Designer Navigator, right-click a Package, Mapping, Procedure or Variable, and select **Generate Scenario...** The New Scenario dialog appears.

2. Enter the **Name** and the **Version** of the scenario. As this name can be used in an operating system command, the name is automatically uppercased and special characters are replaced by underscores.

Note that the **Name** and **Version** fields of the Scenario are preset with the following values:

- **Name:** The same name as the latest scenario generated for the component
- **Version:** The version number is automatically incremented, or set to 001 if no prior numeric version exists

If no scenario has been created yet for the component, a first version of the scenario is automatically created.

Note: New scenarios are named after the component according to the **Scenario Naming Convention** user parameter. You can set this parameter by clicking **Preferences** from the **Tools** option on the menu bar; expand the **ODI** node, and then the **System** node, and select the **Scenarios** node.

3. Click **OK**.
4. If you use variables in the scenario, you can define in the Scenario Variables dialog the variables that will be considered as parameters for the scenario.
 - Select **Use All** if you want all variables to be parameters
 - Select **Use Selected** to use the selected variables to be parameters
 - Select **None** to deselect all variables
5. Click **OK**.

The scenario appears on the Scenarios tab and under the Scenarios node of the source object under the project.

12.3 Regenerating a Scenario

An existing scenario can be regenerated with the same name and version number. This lets you replace the existing scenario by a scenario generated from the source object contents. Schedules attached to this scenario are preserved.

To regenerate a scenario:

1. Select a scenario in the **Projects** or **Load Plans and Scenarios** section of the Designer Navigator.
2. Right-click and select **Regenerate...**
3. Click **OK**.

Caution: Regenerating a scenario cannot be undone. For important scenarios, it is better to generate a scenario with a new version number.

12.4 Generating a Group of Scenarios

When a set of packages, mappings, procedures, and variables grouped under a project or folder is finished and tested, you can generate the scenarios. This operation takes place in Designer Navigator.

To generate a group of scenarios:

1. Select the Project or Folder containing the group of objects.
2. Right-click and select **Generate All Scenarios...**
3. In the **Scenario Source Objects** section, select the types of objects for which you want to generate scenarios.
4. In the **Marker Filter** section, you can filter the components to generate according to a marker from a marker group.
5. Select the scenario **Generation Mode**:
 - **Replace:** Overwrites for each object the last scenario version with a new one with the same internal ID, name and version. Sessions, scenario reports and schedules are deleted. If no scenario exists for an object, a scenario with version number 001 is created.
 - **Re-generate:** Overwrites for each object the last scenario version with a new one with the same internal ID, name and version. It preserves the schedule, sessions, scenario reports, variable selections, and concurrent execution control settings. If no scenario exists for an object, no scenario is created using this mode.
 - **Creation:** Creates for each object a new scenario with the same name as the last scenario version and with an automatically incremented version number. If no scenario exists for an object, a scenario named after the object with version number 001 is created.

Note: If no scenario has been created yet for the component, a first version of the scenario is automatically created.

New scenarios are named after the component according to the **Scenario Naming Convention** user parameter. You can set this parameter by clicking **Preferences** from the **Tools** option on the menu bar; expand the **ODI** node, and then the **System** node, and select the **Scenarios** node.

When selecting the **Creation** generation mode, the version number is automatically incremented, or set to 001 if no prior numeric version exists.

- Select **Generate scenario as if all underlying objects are materialized** to generate the scenario as if the shortcuts were real objects.
6. Click **OK**.
 7. If you use variables in the scenario, you can define in the Scenario Variables dialog the variables that will be considered as parameters for the scenario. Select **Use All** if you want all variables to be parameters, or **Use Selected** and check the parameter variables.

12.5 Controlling Concurrent Execution of Scenarios and Load Plans

By default, nothing prevents two instances of the same scenario or load plan from running simultaneously.

This situation could occur in several ways. For example:

- A load plan containing a Run Scenario Step is running in two or more instances, so the Run Scenario Step may be executed at the same time in more than one load plan instance.
- A scenario is run from the command line, from ODI Studio, or as scheduled on an agent, while another instance of the same scenario is already running (on the same or a different agent or ODI Studio session).

Concurrent executions of the same scenario or load plan apply across all remote and internal agents.

Concurrent execution of multiple instances of a scenario or load plan may be undesirable, particularly if the job involves writing data. You can control concurrent execution using the Concurrent Execution Control options.

ODI identifies a specific scenario or load plan by its internal ID, and not by the name and version. Thus, a regenerated or modified scenario or load plan having the same internal ID is still treated as the same scenario or load plan. Conversely, deleting a scenario and generating a new one with the same name and version number would be creating a different scenario (because it will have a different internal ID).

While Concurrent Execution Control can be enabled or disabled for a scenario or load plan at any time, there are implications to existing running sessions and newly invoked sessions:

- When switching Concurrent Execution Control from disabled to enabled, existing running and queued jobs are counted as executing jobs and new job submissions are processed with the Concurrent Execution Control settings at time of job submission.

- When switching Concurrent Execution Control from enabled to disabled for a scenario or load plan, jobs that are already submitted and in waiting state (or those that are restarted later) will carry the original Concurrent Execution Control setting values to consider and wait for running and queued jobs as executing jobs.

However, if new jobs are submitted at that point with Concurrent Execution Control disabled, they could be run ahead of already waiting jobs. As a result, a waiting job may be delayed if, at the time of polling, the system finds executing jobs that were started without Concurrent Execution Control enabled. And, after a waiting job eventually starts executing, it may still be affected by uncontrolled jobs submitted later and executing concurrently.

To limit concurrent execution of a scenario or load plan, perform the following steps:

1. Open the scenario or load plan by right-clicking it in the Designer or Operator Navigators and selecting **Open**.
2. Select the Definition tab and modify the Concurrent Execution Controller options:
 - Enable the **Limit Concurrent Executions** check box if you do not want to allow multiple instances of this scenario or load plan to be run at the same time. If **Limit Concurrent Executions** is disabled (unchecked), no restriction is imposed and more than one instance of this scenario or load plan can be run simultaneously.
 - If **Limit Concurrent Executions** is enabled, set your desired **Violation Behavior**:
 - **Raise Execution Error**: if an instance of the scenario or load plan is already running, attempting to run another instance will result in a session being created but immediately ending with an execution error message identifying the session that is currently running which caused the Concurrent Execution Control error.
 - **Wait to Execute**: if an instance of the scenario or load plan is already running, additional executions will be placed in a wait status and the system will poll for its turn to run. The session's status is updated periodically to show the currently running session, as well as all concurrent sessions (if any) that are waiting in line to run after the running instance is complete.

If you select this option, the **Wait Polling Interval** sets how often the system will check to see if the running instance has completed. You can only enter a **Wait Polling Interval** if **Wait to Execute** is selected.

If you do not specify a wait polling interval, the default for the executing agent will be used: in ODI 12.1.3, the default agent value is 30 seconds.
3. Click **Save** to save your changes.

12.6 Exporting Scenarios

The export (and import) procedure allows you to transfer Oracle Data Integrator objects from one repository to another.

It is possible to export a single scenario or groups of scenarios.

Exporting one single scenario is covered in "[Exporting one ODI Object](#)" on page 23-10.

To export a group of scenarios:

1. Select the Project or Folder containing the group of scenarios.

2. Right-click and select **Export All Scenarios...** The Export all scenarios dialog opens.
3. In the Export all scenarios dialog, specify the export parameters as follows:

Parameter	Description
Export Directory	Directory in which the export file will be created. Note that if the Export Directory is not specified, the export file is created in the Default Export Directory.
Child components export	If this option is checked, the objects linked to the object to be exported will be also exported. These objects are those visible under the exported object in the tree. It is recommended to leave this option checked. See " Exporting an Object with its Child Components " on page 23-9 for more details.
Replace existing files without warning	If this option is checked, the existing file will be replaced by the ones of the export.

4. Select the type of objects whose scenarios you want to export.
5. Set the encryption options. Set an Export Key if you want the exported scenario to preserve and encrypt sensitive data such as passwords. You will need to supply this Export Key when you later import this scenario if you want to import and decrypt the sensitive data.
6. Set the advanced options. This set of options allow to parameterize the XML output file format. It is recommended that you leave the default values.

Parameter	Description
XML Version	XML Version specified in the export file. Parameter xml version in the XML file header. <code><?xml version="1.0" encoding="ISO-8859-1"?></code>
Character Set	Encoding specified in the export file. Parameter encoding in the XML file header. <code><?xml version="1.0" encoding="ISO-8859-1"?></code>
Java Character Set	Java character set used to generate the file.

7. Click **OK**.

The XML-formatted export files are created at the specified location.

12.7 Importing Scenarios in Production

A scenario generated from Designer can be exported and then imported into a development or execution repository. This operation is used to deploy scenarios in a different repository, possibly in a different environment or site.

Importing a scenario in a development repository is performed with the Designer or Operator Navigator. With an execution repository, only the Operator Navigator is available for this purpose.

There are two ways to import a scenario:

- **Import** uses the standard object import method. During this import process, it is possible to choose to import the schedules attached to the exported scenario.

- **Import Replace** replaces an existing scenario with the content of an export file, preserving references from other objects to this scenario. Sessions, scenario reports and schedules from the original scenario are deleted and replaced with the schedules from the export file.

Scenarios can also be deployed and promoted to production using versions and solutions. See [Chapter 19, "Using Version Control \(Legacy Mode\),"](#) for more information.

12.7.1 Import Scenarios

To import one or more scenarios into Oracle Data Integrator:

1. In Operator Navigator, select the **Scenarios** panel.
2. Right-click and select **Import > Import Scenario**.
3. Select the **Import Type**. Refer to [Chapter 23, "Exporting and Importing,"](#) for more information on the import types.
4. Specify the **File Import Directory**.
5. Check the **Import schedules** option, if you want to import the schedules exported with the scenarios as well.
6. Select one or more scenarios to import from the **Select the file(s) to import** list.
7. Click **OK**.

The scenarios are imported into the work repository. They appear in the Scenarios tree of the Operator Navigator. If this work repository is a development repository, these scenario are also attached to their source Package, Mapping, Procedure, or Variable.

12.7.2 Replace a Scenario

Use the import replace mode if you want to replace a scenario with an exported one.

To import a scenario in replace mode:

1. In Designer or Operator Navigator, select the scenario you wish to replace.
2. Right-click the scenario, and select **Import Replace...**
3. In the Replace Object dialog, specify the scenario export file.
4. Click **OK**.

12.7.3 Working with a Scenario from a Different Repository

A scenario may have to be operated from a different work repository than the one where it was generated.

Examples

Here are two examples of organizations that give rise to this type of process:

- A company has a large number of agencies equipped with the same software applications. In its IT headquarters, it develops packages and scenarios to centralize data to a central data center. These scenarios are designed to be executed identically in each agency.
- A company has three distinct IT environments for developing, qualifying, and operating its software applications. The company's processes demand total separation of the environments, which cannot share the Repository.

Prerequisites

The prerequisite for this organization is to have a work repository installed on each environment (site, agency, or environment). The topology of the master repository attached to this work repository must be compatible in terms of its logical architecture (the same logical schema names). The connection characteristics described in the physical architecture can differ.

Note that in cases where some procedures or mappings explicitly specify a context code, the target topology must have the same context codes. The topology, that is, the physical and logical architectures, can also be exported from a development master repository, then imported into the target repositories. Use the Topology module to carry out this operation. In this case, the physical topology (the servers' addresses) should be personalized before operating the scenarios. Note also that a topology import simply references the new data servers without modifying those already present in the target repository.

To operate a scenario from a different work repository:

1. Export the scenario from its original repository (right-click, export)
2. Forward the scenario export file to the target environment
3. Open Designer Navigator in the target environment (connection to the target repository)
4. Import the scenario from the export file

12.8 Encrypting and Decrypting a Scenario

Encrypting a scenario allows you to protect valuable code. An encrypted scenario can be executed but cannot be read or modified if it is not decrypted. The commands generated in the log by an encrypted scenario are also unreadable.

Oracle Data Integrator uses a DES Encryption algorithm based on a personal encryption key. This key can be saved in a file and can be reused to perform encryption or decryption operations.

WARNING: There is no way to decrypt an encrypted scenario or procedure without the encryption key. It is therefore strongly advised to keep this key in a safe location.

To encrypt a scenario:

1. In Designer or Operator Navigator, select the scenario you want to encrypt.
2. Right-click and select **Encrypt**.
3. In the Encryption Options dialog, you can either:
 - **Encrypt with a personal key** that already exists by giving the location of the personal key file or by typing in the value of the personal key.
 - **Get a new encryption key** to have a new key generated.
4. Click **OK** to encrypt the scenario. If you have chosen to generate a new key, a dialog will appear with the new key. Click **Save** to save the key in a file.

Note: If you type in a personal key with too few characters, an invalid key size error appears.

To decrypt a scenario:

1. Right-click the scenario you want to decrypt.
2. Select **Decrypt**.
3. In the **Scenario Decryption** dialog, either
 - Select an existing encryption key file
 - or type in (or paste) the string corresponding to your personal key.

A message appears when decryption is finished.

Using Load Plans

This chapter gives an introduction to Load Plans. It describes how to create a Load Plan and provides information about how to work with Load Plans.

This chapter includes the following sections:

- [Introduction to Load Plans](#)
- [Creating a Load Plan](#)
- [Running Load Plans](#)
- [Using Load Plans in Production](#)

13.1 Introduction to Load Plans

Oracle Data Integrator is often used for populating very large data warehouses. In these use cases, it is common to have thousands of tables being populated using hundreds of scenarios. The execution of these scenarios has to be organized in such a way that the data throughput from the sources to the target is the most efficient within the batch window. Load Plans help the user organizing the execution of scenarios in a hierarchy of sequential and parallel steps for these type of use cases.

A *Load Plan* is an executable object in Oracle Data Integrator that can contain a hierarchy of *steps* that can be executed conditionally, in parallel or in series. The leaf nodes of this hierarchy are *Scenarios*. Packages, mappings, variables, and procedures can be added to Load Plans for executions in the form of scenarios. For more information, see "[Creating a Load Plan](#)" on page 13-6.

Load Plans allow setting and using variables at multiple levels. See "[Working with Variables in Load Plans](#)" on page 13-15 for more information. Load Plans also support exception handling strategies in the event of a scenario ending in error. See "[Handling Load Plan Exceptions and Restartability](#)" on page 13-16 for more information.

Load Plans can be started, stopped, and restarted from a command line, from Oracle Data Integrator Studio, Oracle Data Integrator Console or a Web Service interface. They can also be scheduled using the run-time agent's built-in scheduler or an external scheduler. When a Load Plan is executed, a *Load Plan Instance* is created. Each attempt to run this Load Plan Instance is a separate *Load Plan Run*. See "[Running Load Plans](#)" on page 13-19 for more information.

A Load Plan can be modified in production environments and steps can be enabled or disabled according to the production needs. Load Plan objects can be designed and viewed in the Designer and Operator Navigators. Various design operations (such as create, edit, delete, and so forth) can be performed on a Load Plan object if a user connects to a development work repository, but some design operations will not be available in an execution work repository. See "[Editing Load Plan Steps](#)" on page 13-12

for more information.

Once created, a Load Plan is stored in the work repository. The Load Plan can be exported then imported to another repository and executed in different contexts. Load Plans can also be versioned. See ["Exporting, Importing and Versioning Load Plans"](#) on page 13-20 for more information.

Load Plans appear in Designer Navigator and in Operator Navigator in the Load Plans and Scenarios accordion. The Load Plan Runs are displayed in the Load Plan Executions accordion in Operator Navigator.

13.1.1 Load Plan Execution Lifecycle

When running or scheduling a Load Plan you provide the variable values, the contexts and logical agents used for this Load Plan execution.

Executing a Load Plan creates a *Load Plan instance* and a first *Load Plan run*. This Load Plan instance is separated from the original Load Plan, and the Load Plan Run corresponds to the first attempt to execute this instance. If a run is restarted a new *Load Plan run* is created under this Load Plan instance. As a consequence, each execution attempt of the Load Plan Instance is preserved as a different Load Plan run in the Log.

See ["Running Load Plans"](#) on page 13-19 for more information.

For a load plan instance, only one run can be running, and it must be the last load plan instance run. However, as with Scenarios, it is possible to run multiple instances of the same load plan (determined by the load plan's internal ID) concurrently, depending on the Concurrent Execution Control settings for the load plan.

For more information about how ODI handles concurrent execution, and about using the Concurrent Execution Control, see ["Controlling Concurrent Execution of Scenarios and Load Plans"](#) on page 12-4,

13.1.2 Differences between Packages, Scenarios, and Load Plans

A *Load Plan* is the largest executable object in Oracle Data Integrator. It uses *Scenarios* in its steps. When an executable object is used in a Load Plan, it is automatically converted into a scenario. For example, a package is used in the form of a scenario in Load Plans. Note that Load Plans cannot be added to a Load Plan. However, it is possible to add a scenario in form of a Run Scenario step that starts another Load Plan using the `OdiStartLoadPlan` tool.

Load plans are not substitutes for packages or scenarios, but are used to organize at a higher level the execution of packages and scenarios.

Unlike packages, Load Plans provide native support for parallelism, restartability and exception handling. Load plans are moved to production as is, whereas packages are moved in the form of scenarios. Load Plans can be created in Production environments.

The *Load Plan instances* and *Load Plan runs* are similar to Sessions. The difference is that when a session is restarted, the existing session is overwritten by the new execution. The new Load Plan Run does not overwrite the existing Load Plan Run, it is added after the previous Load Plan Runs for this Load Plan Instance. Note that the Load Plan Instance cannot be modified at run-time.

13.1.3 Load Plan Structure

A Load Plan is made up of a sequence of several types of steps. Each step can contain several child steps. Depending on the step type, the steps can be executed

conditionally, in parallel or sequentially. By default, a Load Plan contains an empty root serial step. This root step is mandatory and the step type cannot be changed.

Table 13–1 lists the different types of Load Plan steps and the possible child steps.

Table 13–1 Load Plan Steps

Type	Description	Possible Child Steps
Serial Step	Defines a serial execution of its child steps. Child steps are ordered and a child step is executed only when the previous one is terminated. The root step is a Serial step.	<ul style="list-style-type: none"> ■ Serial step ■ Parallel step ■ Run Scenario step ■ Case step
Parallel Step	Defines a parallel execution of its child steps. Child steps are started immediately in their order of Priority.	<ul style="list-style-type: none"> ■ Serial step ■ Parallel step ■ Run Scenario step ■ Case step
Run Scenario Step	Launches the execution of a scenario.	This type of step cannot have a child steps.
Case Step When Step Else Steps	The combination of these steps allows conditional branching based on the value of a variable. Note: If you have several When steps under a Case step, only the first enabled When step that satisfies the condition is executed. If no When step satisfies the condition or the Case step does not contain any When steps, the Else step is executed.	<p>Of a Case Step:</p> <ul style="list-style-type: none"> ■ When step ■ Else step <p>Of a When step:</p> <ul style="list-style-type: none"> ■ Serial step ■ Parallel step ■ Run Scenario step ■ Case step <p>Of an Else step:</p> <ul style="list-style-type: none"> ■ Serial step ■ Parallel step ■ Run Scenario step ■ Case step
Exception Step	Defines a group of steps that is executed when an exception is encountered in the associated step from the Step Hierarchy. The same exception step can be attached to several steps in the Steps Hierarchy.	<ul style="list-style-type: none"> ■ Serial step ■ Parallel step ■ Run Scenario step ■ Case step

Figure 13–1 shows a sample Load Plan created in Oracle Data Integrator. This sample Load Plan loads a data warehouse:

- Dimensions are loaded in parallel. This includes the LOAD_TIME_DIM, LOAD_PRODUCT_DIM, LOAD_CUSTOMER_DIM scenarios, the geographical dimension and depending on the value of the ODI_VAR_SESS1 variable, the CUST_NORTH or CUST_SOUTH scenario.
- The geographical dimension consists of a sequence of three scenarios (LOAD_GEO_ZONE_DIM, LOAD_COUNTRIES_DIM, LOAD_CITIES_DIM).
- After the dimensions are loaded, the two fact tables are loaded in parallel (LOAD_SALES_FACT and LOAD_MARKETING_FACT scenarios).

Figure 13–1 Sample Load Plan

The screenshot displays the MyLoadPlan application interface. The main window shows a hierarchy of steps in a table format. The selected step is 'LOAD_TIME_DIM'.

#	Steps Hierarchy	Enabled	Scenario/Variable	Restart	Context	Logical Agent
0	root_step	<input checked="" type="checkbox"/>		Restart from failure		
1	Parallel	<input checked="" type="checkbox"/>		Restart all children		
2	LOAD_TIME_DIM	<input checked="" type="checkbox"/>	LOAD_TIME_DIM Version 001	Restart from new session	Global	AGENT_4
3	LOAD_PRODUCT_DIM	<input checked="" type="checkbox"/>	LOAD_PRODUCT_DIM Versio...	Restart from new session		
4	LOAD_CUSTOMER_DIM	<input checked="" type="checkbox"/>	LOAD_CUSTOMER_DIM Versi...	Restart from new session		
5	Case where: ODI_VAR_SESS1	<input checked="" type="checkbox"/>	ODI_VAR_SESS1.VAR			
6	Value = 9	<input checked="" type="checkbox"/>				
7	CUST_NORTH	<input checked="" type="checkbox"/>	CUST_NORTH Version 001	Restart from new session		
8	Else	<input checked="" type="checkbox"/>				
9	CUST_SOUTH	<input checked="" type="checkbox"/>	CUST_SOUTH Version 001	Restart from new session		
10	Serial	<input checked="" type="checkbox"/>		Restart from failure		
11	LOAD_GEO_ZONES_DIM	<input checked="" type="checkbox"/>	LOAD_GEO_ZONES_DIM Ver...	Restart from new session	CTX_DTCONV_GLOB	AGENT_3
12	LOAD_COUNTRIES_DIM	<input checked="" type="checkbox"/>	LOAD_COUNTRIES_DIM Vers...	Restart from new session		
13	LOAD_CITIES_DIM	<input checked="" type="checkbox"/>	LOAD_CITIES_DIM Version 002	Restart from new session		
14	Parallel	<input checked="" type="checkbox"/>		Restart all children		
15	LOAD_SALES_FACT	<input checked="" type="checkbox"/>	LOAD_SALES_FACT Version ...	Restart from new session		
16	LOAD_MARKETING_FACT	<input checked="" type="checkbox"/>	LOAD_MARKETING_FACT Ve...	Restart from new session		

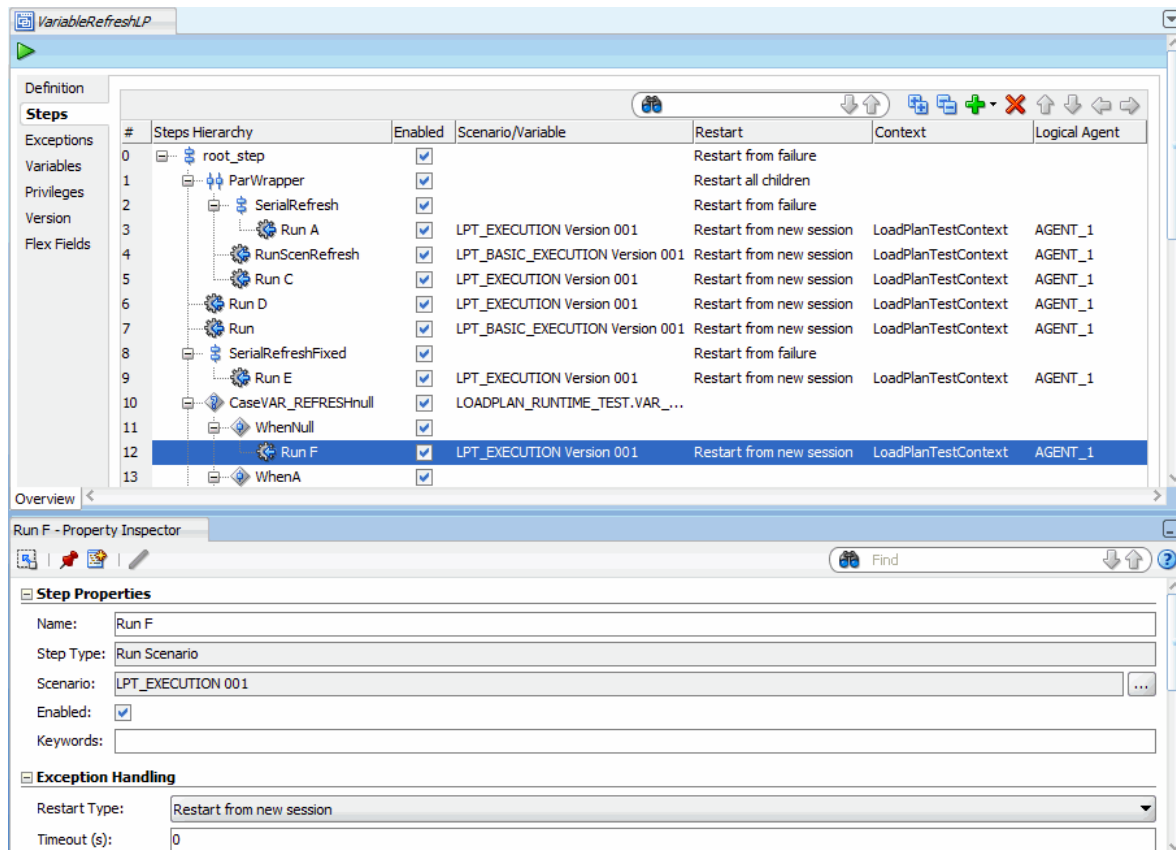
The 'LOAD_TIME_DIM - Property Inspector' window is open, showing the following properties:

- Name: LOAD_TIME_DIM
- Step Type: Run Scenario
- Scenario: LOAD_TIME_DIM 001
- Enabled:
- Keywords:

13.1.4 Introduction to the Load Plan Editor

The Load Plan Editor provides a single environment for designing Load Plans. [Figure 13–2](#) gives an overview of the Load Plan Editor.

Figure 13–2 Steps Tab of the Load Plan Editor



The Load Plan steps are added, edited and organized in the Steps tab of the Load Plan Editor. The *Steps Hierarchy* table defines the organization of the steps in the Load Plan. Each row in this table represents a step and displays its main properties.

You can drag components such as packages, integration mappings, variables, procedures, or scenarios from the Designer Navigator into the Steps Hierarchy table for creating Run Scenario steps for these components.

You can also use the Add Step Wizard or the Quick Step tool to add Run Scenario steps and other types of steps into this Load Plan. See "[Adding Load Plan Steps](#)" on page 13-8 for more information.

The *Load Plan Editor toolbar*, located on top of the Steps Hierarchy table, provides tools for creating, organizing, and sequencing the steps in the Load Plan. [Table 13–2](#) details the different toolbar components.

Table 13–2 Load Plan Editor Toolbar







Icon	Name	Description
	Search	Searches for a step in the Steps Hierarchy table.
	Expand All	Expands all tree nodes in the Steps Hierarchy table.

Table 13–2 (Cont.) Load Plan Editor Toolbar

Icon	Name	Description
	Collapse All	Collapses all tree nodes in the Steps Hierarchy table.
	Add Step	Opens a Add Step menu. You can either select the Add Step Wizard or a Quick Step tool to add a step. See "Adding Load Plan Steps" on page 13-8 for more information.
	Remove Step	Removes the selected step and all its child steps.
	Reorder arrows: Move Up, Move Down, Move Out, Move In	Use the reorder arrows to move the selected step to the required position.

The *Properties Panel*, located under the Steps Hierarchy table, displays the properties for the object that is selected in the Steps Hierarchy table.

13.2 Creating a Load Plan

This section describes how to create a new Load Plan in ODI Studio.

1. Define a new Load Plan. See ["Creating a New Load Plan"](#) on page 13-6 for more information.
2. Add Steps into the Load Plan and define the Load Plan Sequence. See ["Defining the Load Plan Step Sequence"](#) on page 13-8 for more information.
3. Define how the exceptions should be handled. See ["Handling Load Plan Exceptions and Restartability"](#) on page 13-16 for more information.

13.2.1 Creating a New Load Plan

Load Plans can be created from the Designer or Operator Navigator.

To create a new Load Plan:

1. In Designer Navigator or Operator Navigator, click **New Load Plan** in the toolbar of the Load Plans and Scenarios accordion. The Load Plan Editor is displayed.
2. In the Load Plan Editor, type in the **Name**, **Folder Name**, and a **Description** for this Load Plan.
3. Optionally, set the following parameters:
 - **Log Sessions:** Select how the session logs should be preserved for the sessions started by the Load Plan. Possible values are:
 - **Always:** Always keep session logs (Default)
 - **Never:** Never keep session logs. Note that for Run Scenario steps that are configured as *Restart from Failed Step* or *Restart from Failed Task*, the agent will behave as if the parameter is set to **Error** as the whole session needs to be preserved for restartability.

- **Error:** Only keep the session log if the session completed in an error state.
 - **Log Session Step:** Select how the logs should be maintained for the session steps of each of the session started by the Load Plan. Note that this applies only when the session log is preserved. Possible values are:
 - **By Scenario Settings:** Session step logs are preserved depending on the scenario settings. Note that for scenarios created from packages, you can specify whether to preserve or not the steps in the advanced step property called *Log Steps in the Journal*. Other scenarios preserve all the steps (Default).
 - **Never:** Never keep session step logs. Note that for Run Scenario steps that are configured as *Restart from Failed Step* or *Restart from Failed Task*, the agent will behave as if the parameter is set to **Error** as the whole session needs to be preserved for restartability.
 - **Errors:** Only keep session step log if the step is in an error state.
 - **Session Tasks Log Level:** Select the log level for sessions. This value corresponds to the *Log Level* value when starting unitary scenarios. Default is 5. Note that when Run Scenario steps are configured as *Restart from Failed Step* or *Restart From Failed Task*, this parameter is ignored as the whole session needs to be preserved for restartability.
 - **Keywords:** Enter a comma separated list of keywords that will be set on the sessions started from this load plan. These keywords improve the organization of ODI logs by session folders and automatic classification. Note that you can overwrite these keywords at the level of the child steps. See "Managing the Log" in *Administering Oracle Data Integrator* for more information.
4. Optionally, modify the **Concurrent Execution Controller** options:
- Enable the **Limit Concurrent Executions** check box if you do not want to allow multiple instances of this load plan to be run at the same time. If **Limit Concurrent Executions** is disabled (unchecked), no restriction is imposed and more than one instance of this load plan can be running simultaneously.
 - If **Limit Concurrent Executions** is enabled, set your desired **Violation Behavior**:
 - **Raise Execution Error:** if an instance of the load plan is already running, attempting to run another instance will result in a session being created but immediately ending with an execution error message identifying the session that is currently running which caused the Concurrent Execution Control error.
 - **Wait to Execute:** if an instance of the load plan is already running, additional executions will be placed in a wait status and the system will poll for its turn to run. The session's status is updated periodically to show the currently running session, as well as all concurrent sessions (if any) that are waiting in line to run after the running instance is complete.

If you select this option, the **Wait Polling Interval** sets how often the system will check to see if the running instance has completed. You can only enter a **Wait Polling Interval** if **Wait to Execute** is selected.

If you do not specify a wait polling interval, the default for the executing agent will be used: in ODI 12.1.3, the default agent value is 30 seconds.
5. Select the **Steps** tab and add steps as described in "[Defining the Load Plan Step Sequence](#)" on page 13-8.

6. If your Load Plan requires conditional branching, or if your scenarios use variables, select the **Variables** tab and declare variables as described in "[Declaring Load Plan Variables](#)" on page 13-15.
7. To add exception steps that are used in the event of a load plan step failing, select the **Exceptions** tab and define exception steps as described in "[Defining Exceptions Flows](#)" on page 13-17.
8. From the **File** menu, click **Save**.

The Load Plan appears in the Load Plans and Scenarios accordion. You can organize your Load Plans by grouping related Load Plans and Scenarios into a Load Plan and Scenarios folder.

13.2.2 Defining the Load Plan Step Sequence

Load Plans are an organized hierarchy of child steps. This hierarchy allows conditional processing of steps in parallel or in series.

The execution flow can be configured at two stages:

- At Design-time, when defining the Steps Hierarchy:
 - When you add a step to a Load Plan, you select the step type. The step type defines the possible child steps and how these child steps are executed: in parallel, in series, or conditionally based on the value of a variable (Case step). See [Table 13–1, "Load Plan Steps"](#) for more information on step types.
 - When you add a step to a Load Plan, you also decide where to insert the step. You can add a child step, a sibling step after the selected step, or a sibling step before the selected step. See "[Adding Load Plan Steps](#)" on page 13-8 for more information.
 - You can also reorganize the order of the Load Plan steps by dragging the step to the wanted position or by using the arrows in the Step table toolbar. See [Table 13–2, "Load Plan Editor Toolbar"](#) for more information.
- At design-time and run-time by enabling or disabling a step. In the Steps hierarchy table, you can enable or disable a step. Note that disabling a step also disables all its child steps. Disabled steps and all their child steps are not executed when you run the load plan.

This section contains the following topics:

- [Adding Load Plan Steps](#)
- [Editing Load Plan Steps](#)
- [Deleting a Step](#)
- [Duplicating a Step](#)

13.2.2.1 Adding Load Plan Steps

A Load Plan step can be added by using the Add Step Wizard or by selecting the Quick Step tool for a specific step type. A load plan step can be also created

by dragging an object (such as a scenario, package, etc.) and dropping it onto a container step. The step will be created as a child of the selected step. See [Table 13–1, "Load Plan Steps"](#) for more information on the different types of Load Plan steps. To create Run Scenario steps, you can also drag components such as packages, mappings, variables, procedures, or scenarios from the Designer Navigator into the Steps

Hierarchy table. Oracle Data Integrator automatically creates a Run Scenario step for the inserted component.

When a Load Plan step is added, it is inserted into the Steps Hierarchy with the minimum required settings. See ["Editing Load Plan Steps"](#) on page 13-12 for more information on how to configure Load Plan steps.

Adding a Load Plan Step with the Add Step Wizard

To insert Load Plan step with the Add Step Wizard:

1. Open the Load Plan Editor and go to the **Steps** tab.
2. Select a step in the Steps Hierarchy table.
3. In the Load Plan Editor toolbar, select **Add Step > Add Step Wizard**.
4. In the Add Step Wizard, select:
 - **Step Type.** Possible step types are: Serial, Parallel, Run Scenario, Case, When, and Else. See [Table 13-1, "Load Plan Steps"](#) for more information on the different step types.
 - **Step Location.** This parameter defines where the step is added.
 - **Add a child step to selection:** The step is added under the selected step.
 - **Add a sibling step after selection:** The step is added on the same level after the selected step.
 - **Add a sibling step before selection:** The step is added on the same level before the selected step.

Note: Only values that are valid for the current selection are displayed for the **Step Type** and **Step Location**.

5. Click **Next**.
6. Follow the instructions in [Table 13-3](#) for the step type you are adding.

Table 13-3 Add Step Wizard Actions

Step Type	Description and Action Required
Serial or Parallel step	Enter a Step Name for the new Load Plan step.

Table 13–3 (Cont.) Add Step Wizard Actions

Step Type	Description and Action Required
Run Scenario step	<ol style="list-style-type: none"> <li data-bbox="683 260 1094 287">1. Click the Lookup Scenario button. <li data-bbox="683 302 1341 352">2. In the Lookup Scenario dialog, you can select the scenario you want to add to your Load Plan and click OK. Alternately, to create a scenario for an executable object and use this scenario, select this object type in the Executable Object Type selection box, then select the executable object that you want to run with this Run Scenario step and click OK. Enter the new scenario name and version and click OK. A new scenario is created for this object and used in this Run Scenario Step. Tip: At design time, you may want to create a Run Scenario step using a scenario that does not exist yet. In this case, instead of selecting an existing scenario, enter directly a Scenario Name and a Version number and click Finish. Later on, you can select the scenario using the Modify Run Scenario Step wizard. See "Change the Scenario of a Run Scenario Step" on page 13-12 for more information. Note that when you use the version number -1, the latest version of the scenario will be used, based on the string's lexical sorting order. <li data-bbox="683 856 1349 934">3. The Step Name is automatically populated with the name of the scenario and the Version field with the version number of the scenario. Optionally, change the Step Name. <li data-bbox="683 949 846 976">4. Click Next. <li data-bbox="683 991 1365 1094">5. In the Add to Load Plan column, select the scenario variables that you want to add to the Load Plan variables. If the scenario uses certain variables as its startup parameters, they are automatically added to the Load Plan variables. See "Working with Variables in Load Plans" on page 13-15 for more information.
Case	<ol style="list-style-type: none"> <li data-bbox="683 1178 1357 1354">1. Select the variable you want to use for the conditional branching. Note that you can either select one of the load plan variables from the list or click Lookup Variable to add a new variable to the load plan and use it for this case step. See "Working with Variables in Load Plans" on page 13-15 for more information. <li data-bbox="683 1369 1349 1509">2. The Step Name is automatically populated with the step type and name of the variable. Optionally, change the Step Name. See "Editing Load Plan Steps" on page 13-12 for more information.

Table 13–3 (Cont.) Add Step Wizard Actions

Step Type	Description and Action Required
When	<ol style="list-style-type: none"> Select the Operator to use in the WHEN clause evaluation. Possible values are: <ul style="list-style-type: none"> Less Than (<) Less Than or Equal (<=) Different (<>) Equals (=) Greater Than (>) Greater Than or Equal (>=) Is not Null Is Null Enter the Value to use in the WHEN clause evaluation. The Step Name is automatically populated with the operator that is used. Optionally, change the Step Name. See "Editing Load Plan Steps" on page 13-12 for more information.
Else	<p>The Step Name is automatically populated with the step type. Optionally, change the Step Name.</p> <p>See "Editing Load Plan Steps" on page 13-12 for more information.</p>

- Click **Finish**.
- The step is added in the steps hierarchy.

Note: You can reorganize the order of the Load Plan steps by dragging the step to the desired position or by using the reorder arrows in the Step table toolbar to move a step in the Steps Hierarchy.

Adding a Load Plan Step with the Quick Step Tool

To insert Load Plan step with the Quick Step Tool:

- Open the Load Plan editor and go to the **Steps** tab.
- In the Steps Hierarchy, select the Load Plan step under which you want to create a child step.
- In the Steps toolbar, select **Add Step** and the Quick Step option corresponding to the Step type you want to add. [Table 13–4](#) lists the options of the Quick Step tool.

Table 13–4 Quick Step Tool







Quick Step tool option	Description and Action Required
	Adds a serial step as a child of the selected step. Default values are used. You can modify these values in the Steps Hierarchy table or in the Property Inspector. See "Editing Load Plan Steps" on page 13-12 for more information.

Table 13–4 (Cont.) Quick Step Tool

Quick Step tool option	Description and Action Required
	Adds a parallel step as a child of the selected step. Default values are used. You can modify these values in the Steps Hierarchy table or in the Property Inspector. See "Editing Load Plan Steps" on page 13-12 for more information.
	Adds a run scenario step as a child of the selected step. Follow the instructions for Run Scenario steps in Table 13–3 .
	Adds a Case step as a child of the selected step. Follow the instructions for Case steps in Table 13–3 .
	Adds a When step as a child of the selected step. Follow the instructions for When steps in Table 13–3 .
	Adds an Else step as a child of the selected step. Follow the instructions for Else steps in Table 13–3 .

Note: Only step types that are valid for the current selection are enabled in the Quick Step tool.

13.2.2.2 Editing Load Plan Steps

To edit a Load Plan step:

1. Open the Load Plan editor and go to the **Steps** tab.
2. In the Steps Hierarchy table, select the Load Plan step you want modify. The Property Inspector displays the step properties.
3. Edit the Load Plan step properties according to your needs.

The following operations are common tasks when editing steps:

- [Change the Scenario of a Run Scenario Step](#)
- [Set Advanced Options for Run Scenario Steps](#)
- [Open the Linked Object of Run Scenario Steps](#)
- [Change the Test Variable in Case Steps](#)
- [Define the Exception and Restart Behavior](#)
- [Regenerate Scenarios](#)
- [Refresh Scenarios to Latest Version](#)

Change the Scenario of a Run Scenario Step

To change the scenario:

1. In the Steps Hierarchy table of the Steps or Exceptions tab, select the Run Scenario step.
2. In the Step Properties section of the Properties Inspector, click **Lookup Scenario**. This opens the Modify Run Scenario Step wizard.
3. In the Modify Run Scenario Step wizard, click **Lookup Scenario** and follow the instructions in [Table 13–3](#) corresponding to the Run Scenario step.

Set Advanced Options for Run Scenario Steps

You can set the following properties for Run Scenario steps in the Property Inspector:

- **Priority:** Priority for this step when the scenario needs to start in parallel. The integer value range is from 0 to 100 (100 being the highest priority). Default is 0. The priority of a Run Scenario step is evaluated among all runnable scenarios within a running Load Plan. The Run Scenario step with the highest priority is executed first.
- **Context:** Context that is used for the step execution. Default context is the Load Plan context that is defined in the Start Load Plan Dialog when executing a Load Plan. Note that if you only specify the Context and no Logical Agent value, the step is started on the same physical agent that started the Load Plan, but in this specified context.
- **Logical Agent:** Logical agent that is used for the step execution. By default, the logical agent, which is defined in the Start Load Plan Dialog when executing a Load Plan, is used. Note that if you set only the Logical Agent and no context, the step is started with the physical agent corresponding to the specified Logical Agent resolved in the context specified when starting the Load Plan. If no Logical Agent value is specified, the step is started on the same physical agent that started the Load Plan (whether a context is specified for the step or not).

Open the Linked Object of Run Scenario Steps

Run Scenario steps can be created for packages, mappings, variables, procedures, or scenarios. Once this Run Scenario step is created, you can open the Object Editor of the original object to view and edit it.

To view and edit the linked object of Run Scenario steps:

1. In the Steps Hierarchy table of the Steps or Exceptions tab, select the Run Scenario step.
2. Right-click and select **Open the Linked Object**.

The Object Editor of the linked object is displayed.

Change the Test Variable in Case Steps

To change the variable that is used for evaluating the tests defined in the WHEN statements:

1. In the Steps Hierarchy table of the Steps tab or Exceptions tab, select the Case step.
2. In the Step Properties section of the Properties Inspector, click **Lookup Variable**. This opens the Modify Case Step Dialog.
3. In the Modify Case Step Dialog, click **Lookup Variable** and follow the instructions in [Table 13-3, "Add Step Wizard Actions"](#) corresponding to the Case step.

Define the Exception and Restart Behavior

Exception and Restart behavior can be set on the steps in the Steps Hierarchy table. See ["Handling Load Plan Exceptions and Restartability"](#) on page 13-16 for more information.

Regenerate Scenarios

To regenerate all the scenarios of a given Load Plan step, including the scenarios of its child steps:

1. From the Steps Hierarchy table of the Steps tab or Exceptions tab, select the Load Plan step.
2. Right-click and select **Regenerate**. Note that this option is not available for scenarios with the version number -1.
3. Click **OK**.

Caution: Regenerating a scenario cannot be undone. For important scenarios, it is better to generate a scenario with a new version number.

Refresh Scenarios to Latest Version

To modify all the scenario steps of a given Load Plan step, including the scenarios of its child steps, and set the scenario version to the latest version available for each scenario:

1. From the Steps Hierarchy table of the Steps tab or Exceptions tab, select the Load Plan step.
2. Right-click and select **Refresh Scenarios to Latest Version**. Note that the latest scenario version is determined by the Scenario Creation timestamp. While during the ODI agent execution, the latest scenario is determined by alphabetical ascending sorting of the Scenario Version string value and picking up the last from the list.

Note: This option is not available for scenarios with the version number -1.

3. Click **OK**.

13.2.2.3 Deleting a Step

To delete a step:

1. Open the Load Plan Editor and go to the **Steps** tab.
2. In the Steps Hierarchy table, select the step to delete.
3. In the Load Plan Editor toolbar, select **Remove Step**.

The step and its child steps are removed from the Steps Hierarchy table.

Note: It is not possible to undo a delete operation in the Steps Hierarchy table.

13.2.2.4 Duplicating a Step

To duplicate a step:

1. Open the Load Plan Editor and go to the **Steps** tab.
2. In the Steps Hierarchy table, right-click the step to duplicate and select **Duplicate Selection**.
3. A copy of this step, including its child steps, is created and added as a sibling step after the original step to the Step Hierarchy table.

You can now move and edit this step.

13.2.3 Working with Variables in Load Plans

Project and Global Variables used in a Load Plan are declared as Load Plan Variables in the Load Plan editor. These variables are automatically available in all steps and their value passed to the Load Plan steps.

The variables values are passed to the Load Plan on startup as startup parameters. At a step level, you can overwrite the variable value (by setting it or forcing a refresh) for this step and its child steps.

Note: Load plan variables are copies of Project and Global variables. Thus, changes to the definition of the original project and global variables are not automatically propagated to corresponding variables that are already created in a load plan. You can use the **Refresh Variable Definition** option on the right-click context menu to update the definition of a load plan variable with the current value from the corresponding Project or Global variable.

Because a load plan variable is a copy of the original project or global variable, at startup, Load Plans do not take into account the default value of the original project or global variable, or the historized/latest value of the variable in the execution context. The value of the variable is either the one specified when starting the Load Plan, or the value set/refreshed within the Load Plan.

You can use variables in Run Scenario steps - the variable values are passed as startup parameters to the scenario - or in Case/When/Else steps for conditional branching.

This section contains the following topics:

- [Declaring Load Plan Variables](#)
- [Setting Variable Values in a Step](#)

13.2.3.1 Declaring Load Plan Variables

To declare a Load Plan variable:

1. Open the Load Plan editor and go to the **Variables** tab.
2. From the Load Plan Editor toolbar, select **Add Variable**. The Lookup Variable dialog is displayed.
3. In the Lookup Variable dialog, select the variable to add your Load Plan.
4. The variable appears in the Variables tab of the Load Plan Editor and in the Property Inspector of each step.

13.2.3.2 Setting Variable Values in a Step

Variables in a step inherit their value from the value from the parent step and ultimately from the value specified for the variables when starting the Load Plan.

For each step, except for Else and When steps, you can also overwrite the variable value, and change the value used for this step and its child steps.

Variable values overwritten or refreshed at a given step are available to all the step's descendants, until the value is overwritten or refreshed again for a descendant branch.

Similarly, a variable value overwritten or refreshed at a given step does not affect the value for sibling or parent steps.

To override variable values at step level:

1. Open the Load Plan editor and go to the **Steps** tab.
2. In the Steps Hierarchy table, select the step for which you want to overwrite the variable value.
3. In the Property Inspector, go to the Variables section. The variables that are defined for this Load Plan are listed in this Variables table. You can modify the following variable parameters:

Select **Overwrite**, if you want to specify a variable value for this step and all its children. Once you have chosen to overwrite the variable value, you can either:

- Set a new variable value in the **Value** field.
- Select **Refresh** to refresh this variable prior to executing the step. The Refresh option can be selected only for variables with a Select Query defined for refreshing the variable value.

Note: If the refresh SQL of a Global or Project variable has changed, the variable refresh SQL of the corresponding load plan variable is not updated automatically. You can update the load plan variable refresh SQL by selecting the **Refresh Variable Definition** option from the right-click context menu for a load plan variable on the Variables tab of the load plan editor.

13.2.4 Handling Load Plan Exceptions and Restartability

Load Plans provide two features for handling error cases in the execution flows: *Exceptions* and *Restartability*.

Exceptions

An *Exception Step* contains a hierarchy of steps that is defined on the Exceptions tab of the Load Plan editor.

You can associate a given exception step to one or more steps in the Load Plan. When a step in the Load Plan errors out, the associated exception step is executed automatically.

Exceptions can be optionally raised to the parent step of the failing step. Raising an exception fails the parent step, which can consequently execute its exception step.

Restartability

When a Load Plan Run is restarted after a failure, the failed Load Plan steps are restarted depending on the Restart Type parameter. For example, you can define whether a parallel step should restart all its child steps or only those that have failed.

This section contains the following topics:

- [Defining Exceptions Flows](#)
- [Using Exception Handling](#)
- [Defining the Restart Behavior](#)

13.2.4.1 Defining Exceptions Flows

Exception steps are created and defined on the Exceptions tab of the Load Plan Editor.

This tab contains a list of *Exception Steps*. Each Exception Step consists in a hierarchy of Load Plan steps.

The Exceptions tab is similar to the Steps tab in the Load Plan editor. The main differences are:

- There is no root step for the Exception Step hierarchy. Each exception step is a separate root step.
- The Serial, Parallel, Run Scenario, and Case steps have the same properties as on the Steps tab but do not have an Exception Handling properties group. An exception step that errors out cannot raise another exception step.

An Exception step can be created either by using the Add Step Wizard or with the Quick Step tool by selecting the **Add Step > Exception Step** in the Load Plan Editor toolbar. By default, the Exception step is created with the Step name: *Exception*. You can modify this name in the Steps Hierarchy table or in the Property Inspector.

To create an Exception step with the Add Step Wizard:

1. Open the Load Plan Editor and go to the **Exceptions** tab.
2. In the Load Plan Editor toolbar, select **Add Step > Add Step Wizard**.
3. In the Add Step Wizard, select **Exception** from the **Step Type** list.

Note: Only values that are valid for the current selection are displayed for the **Step Type**.

4. Click **Next**.
5. In the **Step Name** field, enter a name for the Exception step.
6. Click **Finish**.
7. The Exception step is added in the steps hierarchy.

You can now define the exception flow by adding new steps and organizing the hierarchy under this exception step.

13.2.4.2 Using Exception Handling

Defining exception handling for a Load Plan step consists of associating an Exception Step to this Load Plan step and defining the exception behavior. Exceptions steps can be set for each step except for When and Else steps.

To define exception handling for a Load Plan step:

1. Open the Load Plan Editor and go to the **Steps** tab.
2. In the Steps Hierarchy table, select the step for which you want to define an exception behavior. The Property Inspector displays the Step properties.
3. In the **Exception Handling** section of the Property Inspector, set the parameters as follows:
 - **Timeout (s):** Enter the maximum time (in seconds) that this step takes before it is aborted by the Load Plan. When a time-out is reached, the step is marked in error and the Exception step (if defined) is executed. In this case, the exception

step never times out. If needed, a timeout can be set on a parent step to safe guard such a potential long running situation.

If the step fails before the timeout and an exception step is executed, then the execution time of the step plus the execution time of the exception step should not exceed the timeout, otherwise the exception step will fail when the timeout is reached.

Note that the default value of zero (0) indicates an infinite timeout.

- **Exception Step:** From the list, select the Exception step to execute if this step fails. Note that only Exception steps that have been created and defined on the Exceptions tab of the Load Plan Editor appear in this list. See "[Defining Exceptions Flows](#)" on page 13-17 for more information on how to create an Exception step.
- **Exception Behavior:** Defines how this step behaves in case an exception is encountered. Select one of the following:
 - **Run Exception and Raise:** Runs the Exception Step (if any) and raises the exception to the parent step.
 - **Run Exception and Ignore:** Runs the Exception Step (if any) and ignores the exception. The parent step is notified of a successful run. Note that if an exception is caused by the exception step itself, the parent step is notified of the failure.

For Parallel steps only, the following parameters may be set:

Max Error Child Count: Displays the maximum number of child steps in error that is accepted before this step is to be considered in error. When the number of failed child steps exceeds this value, the parallel step is considered failed. The currently running child steps are continued or stopped depending on the Restart Type parameter for this parallel step:

- If the Restart type is **Restart from failed children**, the Load Plan waits for all child sessions (these are the currently running sessions and the ones waiting to be executed) to run and complete before it raises the error to the parent step.
- If the Restart Type is **Restart all children**, the Load Plan kills all running child sessions and does not start any new ones before it raises the error to the parent.

13.2.4.3 Defining the Restart Behavior

The Restart Type option defines how a step in error restarts when the Load Plan is restarted. You can define the **Restart Type** parameter in the Exception Handling section of the Properties Inspector.

Depending on the step type, the **Restart Type** parameter can take the values listed in [Table 13-5](#).

Table 13-5 Restart Type Values

Step Type	Values and Description
Serial	<ul style="list-style-type: none"> ■ Restart all children: When the Load Plan is restarted and if this step is in error, the sequence of steps restarts from the first one. ■ Restart from failure: When the Load Plan is restarted and if this step is in error, the sequence of child steps starts from the one that has failed.

Table 13–5 (Cont.) Restart Type Values

Step Type	Values and Description
Parallel	<ul style="list-style-type: none"> ■ Restart all children: When the Load Plan is restarted and if this step is in error, all the child steps are restarted regardless of their status. This is the default value. ■ Restart from failed children: When the Load Plan is restarted and if this step is in error, only the failed child steps are restarted in parallel.
Run Scenario	<ul style="list-style-type: none"> ■ Restart from new session: When restarting the Load Plan and this Run Scenario step is in error, start the scenario and create a new session. This is the default value. ■ Restart from failed step: When restarting the Load Plan and this Run Scenario step is in error, restart the session from the step in error. All the tasks under this step are restarted. ■ Restart from failed task: When restarting the Load Plan and this Run Scenario step is in error, restart the session from the task in error. <p>The same limitation as those described in "Restarting a Session" in <i>Administering Oracle Data Integrator</i> apply to the sessions restarted from a failed step or failed task.</p>

13.3 Running Load Plans

You can run a Load Plan from Designer Navigator or Operator Navigator in ODI Studio.

Caution: Unless concurrent execution has been limited by using the **Concurrent Execution Controller** options on the **Definition** tab of a load plan, no restriction is imposed to prevent multiple instances of a load plan from running simultaneously. It is possible for two or more instances of a load plan to perform data read/write operations on the same data sources and targets simultaneously. Use the **Limit Concurrent Executions** option to disallow this behavior programmatically if concurrent execution would be undesirable.

See "[Creating a New Load Plan](#)" on page 13-6 for details.

To run a Load Plan in Designer Navigator or Operator Navigator:

1. In the Load Plans and Scenarios accordion, select the Load Plan you want to execute.
2. Right-click and select **Execute**.
3. In the Start Load Plan dialog, select the execution parameters:
 - Select the **Context** into which the Load Plan will be executed.
 - Select the **Logical Agent** that will run the Load Plan.
 - In the Variables table, enter the **Startup values** for the variables used in this Load Plan.
4. Click **OK**.
5. The **Load Plan Started** dialog is displayed.
6. Click **OK**.

The Load Plan execution starts: a Load Plan instance is created along with the first Load Plan run. You can review the Load Plan execution in the Operator Navigator.

For more information, see "Monitoring Integration Processes" in *Administering Oracle Data Integrator*, and see also "Running Integration Processes" in *Administering Oracle Data Integrator* for more information on the other run-time operations on Load Plans.

13.4 Using Load Plans in Production

Using Load Plans in production involves the following tasks:

- Scheduling, starting, monitoring, stopping and restarting Load Plans. See ["Scheduling and Running Load Plans in Production"](#) on page 13-20 for information.
- Moving Load Plans across environments. See ["Exporting, Importing and Versioning Load Plans"](#) on page 13-20

13.4.1 Scheduling and Running Load Plans in Production

"Running Integration Processes" in *Administering Oracle Data Integrator* describes how to schedule and run load plans, including executing, restarting, and stopping load plan runs.

13.4.2 Exporting, Importing and Versioning Load Plans

A Load Plan can be exported and then imported into a development or execution repository. This operation is used to deploy Load Plans in a different repository, possibly in a different environment or site.

The export (and import) procedure allows you to transfer Oracle Data Integrator objects from one repository to another.

13.4.2.1 Exporting Load Plans

It is possible to export a single Load Plan or several Load Plans at once.

Exporting one single Load Plan follows the standard procedure described in ["Exporting one ODI Object"](#) on page 23-10.

For more information on exporting several Load Plans at once, see ["Export Multiple ODI Objects"](#) on page 23-11.

Note that when you export a Load Plan and you select **Export child objects**, all its child steps, schedules, and variables are also exported.

Note: The export of a Load Plan does not include the scenarios referenced by the Load Plan. Scenarios used in a Load Plan need to be exported separately. How to export scenarios is described in ["Exporting Scenarios"](#) on page 12-5.

13.4.2.2 Importing Load Plans

Importing a Load Plan in a development repository is performed via Designer or Operator Navigator. With an execution repository, only Operator Navigator is available for this purpose.

The Load Plan import uses the standard object import method. See ["Importing Objects"](#) on page 23-11 for more information.

Note: The export of a Load Plan does not include the scenarios referenced by the Load Plan. Scenarios used in a Load Plan need to be imported separately.

13.4.2.3 Versioning Load Plans

Load Plans can also be deployed and promoted to production using versions and solutions. See [Chapter 19, "Using Version Control \(Legacy Mode\),"](#) for more information.

Using Web Services

This chapter describes how to work with Web services in Oracle Data Integrator.

This chapter includes the following sections:

- [Introduction to Web Services in Oracle Data Integrator](#)
- [Oracle Data Integrator Run-Time Services and Data Services](#)
- [Invoking Third-Party Web Services](#)

14.1 Introduction to Web Services in Oracle Data Integrator

Oracle Data Integrator provides the following entry points into a service-oriented architecture (SOA):

- Data services
- Oracle Data Integrator run-time services
- Invoking third-party Web services

Figure 14–1 shows an overview of how the different types of Web services can interact.

Figure 14–1 Web Services in Action

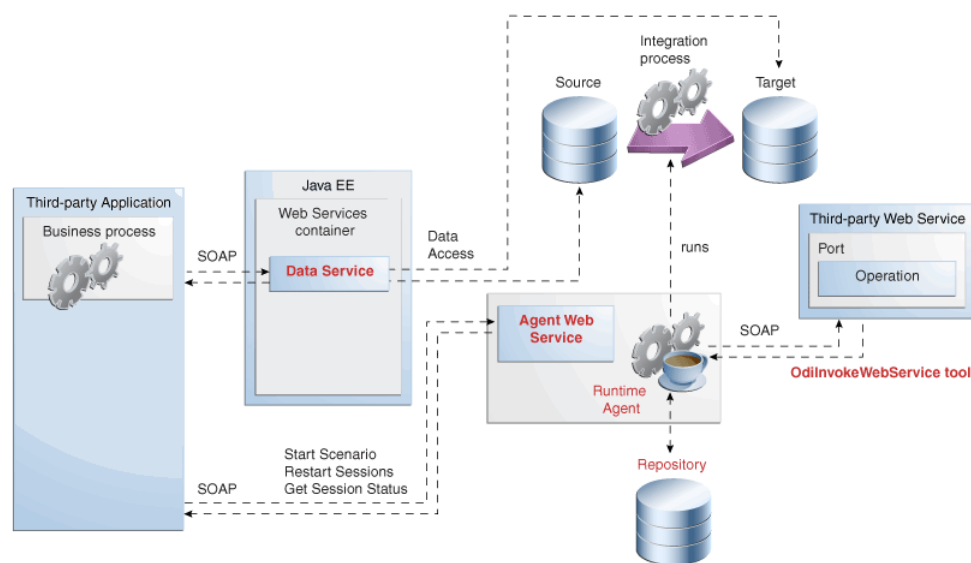


Figure 14–1 shows a simple example with the Data Services, Run-Time Web services (Public Web service and Agent Web service) and the OdiInvokeWebService tool.

The Data Services and Run-Time Web services components are invoked by a third-party application, whereas the OdiInvokeWebService tool invokes a third-party Web service:

- The *Data Services* provides access to data in data stores (both source and target data stores), as well as changes trapped by the Changed Data Capture framework. This Web service is generated by Oracle Data Integrator and deployed in a Java EE application server.
- The *Agent Web service* commands the Oracle Data Integrator Agent to: start and monitor a scenario; restart a session; get the ODI version; start, stop or restart a load plan; or refresh agent configuration. Note that this Web service is built in the Java EE or Standalone Agent.
- The *OdiInvokeWebService* tool is used in a package and invokes a specific operation on a port of the third-party Web service, for example to trigger a BPEL process.

14.2 Oracle Data Integrator Run-Time Services and Data Services

Oracle Data Integrator Run-Time Web services and *Data Services* are two different types of Web services:

- *Oracle Data Integrator Run-Time Services* (Agent Web service) are Web services that enable users to leverage Oracle Data Integrator features in a service-oriented architecture (SOA).

Oracle Data Integrator Run-Time Web services enable you to access the Oracle Data Integrator features through Web services. These Web services are invoked by a third-party application and manage execution of runtime artifacts developed with Oracle Data Integrator.

"Managing Executions Using Web Services" in *Administering Oracle Data Integrator* describes how to perform the different ODI execution tasks with ODI Run-Time Services, such as executing a scenario and restarting a session. That topic also provides examples of SOAP requests and responses.

- *Data Services* are specialized Web services that provide access to data in datastores, and to changes captured for these datastores using Changed Data Capture. Data Services are generated by Oracle Data Integrator to give you access to your data through Web services. These Web services are deployed to a Web services container in an application server.

For more information on how to set up, generate and deploy Data Services refer to "Generating and Deploying Data Services" in *Administering Oracle Data Integrator*.

14.3 Invoking Third-Party Web Services

This section describes how to invoke third-party Web services in Oracle Data Integrator.

This section includes the following topics:

- [Introduction to Web Service Invocation](#)
- [Using HTTP Analyzer](#)
- [Using the OdiInvokeWebService Tool](#)

14.3.1 Introduction to Web Service Invocation

You can invoke Web services:

- In Oracle Data Integrator packages or procedures using the HTTP Analyzer tool. This tool allows you to invoke any third party Web service, and save the response in a SOAP file that can be processed with Oracle Data Integrator.
You can use the results to debug a locally or remotely deployed Web service.
- For testing Data Services. The easiest way to test whether your generated data services are running correctly is to use the HTTP Analyzer tool.
- In Oracle Data Integrator packages or procedures using the OdiInvokeWebService tool. This tool allows you to invoke any third party Web service, and save the response in an XML file that can be processed with Oracle Data Integrator.

The three approaches are described in the sections that follow.

14.3.2 Using HTTP Analyzer

The HTTP Analyzer allows you to monitor request/response traffic between a Web service client and the service. The HTTP Analyzer helps you to debug your Web service in terms of the HTTP traffic sent and received.

When you run the HTTP Analyzer, there are a number of windows that provide information for you.

The HTTP Analyzer enables you to:

- Observe the exact content of the request and response TCP packets of your Web service.
- Edit a request packet, re-send the packet, and see the contents of the response packet.
- Test Web services that are secured using policies; encrypted messages will be decrypted.

This section describes the following topics:

- [Using HTTP Analyzer: Main Steps](#)
- [What Happens When You Run the HTTP Analyzer](#)
- [How to Specify HTTP Analyzer Settings](#)
- [How to Use the Log Window](#)
- [How to Use the Test Window](#)
- [How to Use the Instances Window](#)
- [How to Use Multiple Instances](#)
- [Using Credentials With HTTP Analyzer](#)
- [Using SSL With HTTP Analyzer](#)
- [How to Debug Web Pages Using the HTTP Analyzer](#)
- [How to Use Rules to Determine Behavior](#)
- [How to Set Rules](#)
- [Reference: Troubleshooting the HTTP Analyzer](#)

14.3.2.1 Using HTTP Analyzer: Main Steps

To examine the packets sent and received by the client to a Web service:

Note: In order to use the HTTP Analyzer, you may need to update the proxy settings.

1. Create and run the Web service.
2. Start the HTTP Analyzer by selecting **Tools > HTTP Analyzer**.
You can also start it from the **HTTP Analyzer** button on the **General** tab of the **OdiInvokeWebService** tool step.
It opens in its own window.
3. Click the **Create New Soap Request** button in the HTTP Analyzer Log window.
4. Enter the URL of a Web service, or open the WSDL for a Web service, to get started.
5. Run the client proxy to the Web service. The request/response packet pairs are listed in the HTTP Analyzer Test window.

The test window allows you examine the headers and parameters of a message. You can test the service by entering a parameter that is appropriate and clicking **Send Request**.

6. You can examine the contents of the HTTP headers of the request and response packets to see the SOAP structure (for JAX-WS Web services), the HTTP content, the Hex content or the raw message contents by choosing the appropriate tab at the bottom of the HTTP Analyzer Test window.

Note: The WADL structure (for RESTful services) is not supported by Oracle Data Integrator.

7. You can test Web services that are secured using policies by performing one of the following tasks:
 - Select an existing credential from the **Credentials** list.
Oracle Data Integrator delivers with a set of preconfigured credentials, **HTTPS Credential**.
 - Click **New** to create a new credential. In the Credential dialog, define the credentials to use in the HTTP Analyzer Test window.

14.3.2.2 What Happens When You Run the HTTP Analyzer

When you start the HTTP Analyzer and test a Web service, the Web service sends its traffic via the HTTP Analyzer, using the proxy settings in the HTTP Analyzer Preferences dialog.

By default, the HTTP Analyzer uses a single proxy on an analyzer instance (the default is 8099), but you can add additional proxies of your own if you need to.

Each analyzer instance can have a set of rules to determine behavior, for example, to redirect requests to a different host/URL, or to emulate a Web service.

14.3.2.3 How to Specify HTTP Analyzer Settings

By default, the HTTP Analyzer uses a single proxy on an analyzer instance (the default is 8099), but you can add additional proxies of your own if you need to.

To set HTTP Analyzer preferences:

1. Open the HTTP Analyzer preferences dialog by doing one of the following:
 - Click the Start HTTP Analyzer button in the HTTP Analyzer Instances window or Log window.
 - Choose **Tools > Preferences** to open the Preferences dialog, and navigating to the HTTP Analyzer page.

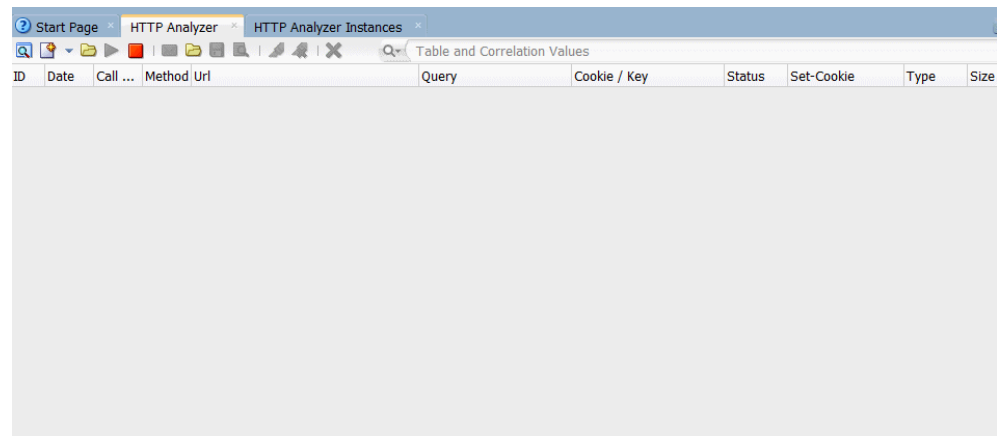
For more information at any time, press F1 or click **Help** from the HTTP Analyzer preferences dialog.

2. Make the changes you want to the HTTP Analyzer instance. For example, to use a different host and port number, open the Proxy Settings dialog by clicking **Configure Proxy**.

14.3.2.4 How to Use the Log Window

When you open the HTTP Analyzer from the Tools menu, the HTTP Analyzer Log window opens, illustrated in [Figure 14-2](#).












Figure 14-2 HTTP Analyzer Log Screen



When HTTP Analyzer runs, it outputs request/response messages to the HTTP Analyzer log window. You can group and reorder the messages:

- To reorder the messages, select the Sequence tab, then sort using the column headers (click on the header to sort, double-click to secondary sort).
- To group messages, click the Correlation tab.
- To change the order of columns, grab the column header and drag it to its new position.

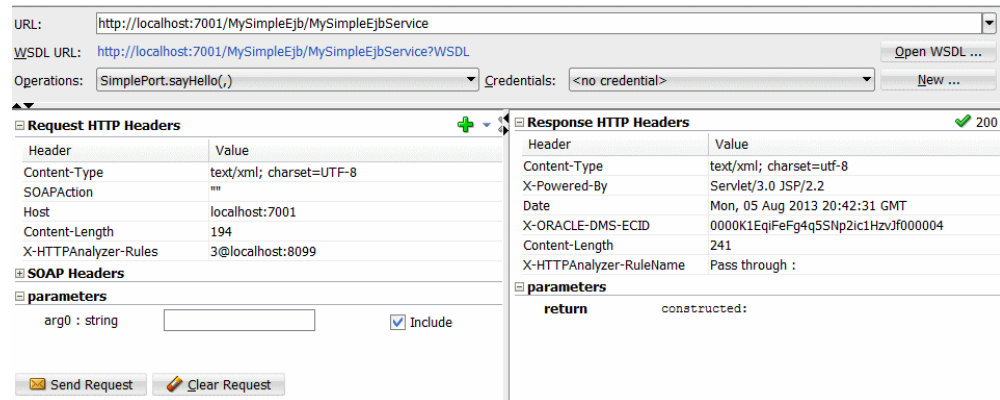
Table 14–1 HTTP Analyzer Log Window Toolbar Icons

Icon	Name	Function
	Analyzer Preferences	Click to open the HTTP Analyzer Preferences dialog where you can specify a new listener port, or change the default proxy. An alternative way to open this dialog is to choose Tools > Preferences , and then navigate to the HTTP Analyzer page. For more information, see
	Create New Request	Click to open the HTTP Analyzer Test window, where you enter payload details, and edit and resend messages.
	Start HTTP Analyzer	Click to start the HTTP Analyzer running. The monitor runs in the background, and only stops when you click Stop or exit JDeveloper. If you have more than one listener defined clicking this button starts them all. To start just one listener, click the down arrow and select the listener to start.
	Stop HTTP Analyzer	Click to stop the HTTP Analyzer running. If you have more than one listener running, clicking this button stops them all. To stop just one listener click the down arrow and select the listener to stop.
	Send Request	Click to resend a request when you have changed the content of a request. The changed request is sent and you can see any changes in the response that is returned.
	Open WS-I log file	Click to open the Select WS-I Log File to Upload dialog, where you can navigate to an existing WS-I log file.
	Save Packet Data	Click to save the contents of the HTTP Analyzer Log Window to a file.
	WS-I Analyze	This tool does not apply to Oracle Data Integrator.
	Select All	Click to select all the entries in the HTTP Analyzer Log Window.
	Deselect All	Click to deselect all the entries in the HTTP Analyzer.
	Clear Selected History (Delete)	Click to clear the entries in the HTTP Analyzer.

14.3.2.5 How to Use the Test Window

An empty HTTP Analyzer test window appears when you click the **Create New Soap Request** button in the HTTP Analyzer Log window.

Enter the URL of a Web service, or open the WSDL for a Web service, and then click **Send Request**. The results of the request are displayed in the test window, as shown in [Figure 14–3](#).

Figure 14–3 HTTP Analyzer Test Window

You can examine the contents of the HTTP headers of the request and response packets to see the SOAP structure, the HTTP content, the Hex content or the raw message contents by choosing the appropriate tab at the bottom of the HTTP Analyzer Test window.

The test window allows you examine the headers and parameters of a message. You can test the service by entering a parameter that is appropriate and clicking **Send Request**.

The tabs along the bottom of the test window allow you choose how you see the content of the message. You can choose to see the message as:

- The SOAP structure, illustrated in [Figure 14–3](#).
- The HTTP code, for example:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://www.example.com/wls">
  <env:Header/>
  <env:Body>
    <ns1:sayHello>
      <arg0/>
    </ns1:sayHello>
  </env:Body>
```

- The hex content of the message, for example:

```
[000..015] 3C 65 65 20 78 6D ... <env:Envelope xm
[016..031] 6C 6E 70 3A 2F 2F ... lns:env="http://
[032..047] 73 63 6F 61 70 2E ... schemas.xmlsoap.
[048..063] 6F 72 65 6C 6F 70 ... org/soap/envelop
[064..079] 65 2F 22 20 78 6D ... e/" xmlns:ns1="h
[080..095] 74 74 70 3A 2F 2F ... ttp://www.bea.co
[096..111] 6D 2F 77 6C 73 22 ... m/wls"><env:Head
[112..127] 65 72 2F 3E 3C 65 ... er/><env:Body><n
[128..143] 73 31 3A 73 61 79 ... s1:sayHello><arg
[144..159] 30 3E 3C 2F 61 72 ... 0></arg0></ns1:s
[160..175] 61 79 48 65 6C 6C ... ayHello></env:Bo
[176..191] 64 79 3E 3C 2F 65 ... dy></env:Envelop
[192..193] 65 3E ... e>
```

- The raw message, for example:

```
POST http://localhost:7001/MySimpleEjb/MySimpleEjbService HTTP/1.1
Content-Type: text/xml; charset=UTF-8
```

```

SOAPAction: ""
Host: localhost:7001
Content-Length: 194
X-HTTPAnalyzer-Rules: 3@localhost:8099

<?xml version = '1.0' encoding = 'UTF-8'?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://www.example.com/wls">
  <env:Header/>
  <env:Body>
    <ns1:sayHello>
      <arg0/>
    </ns1:sayHello>
  </env:Body>
</env:Envelope>

```

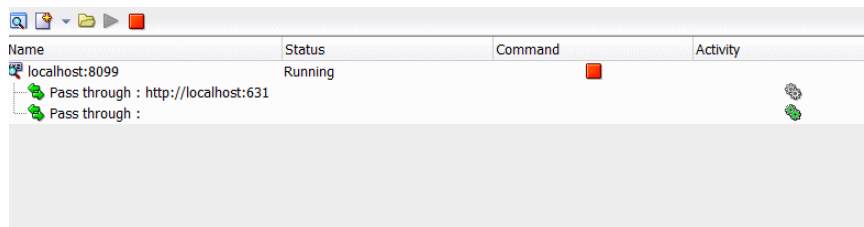
14.3.2.6 How to Use the Instances Window

When you open the HTTP Analyzer from the **Tools** menu, the HTTP Analyzer tab appears by default.

Click the HTTP Analyzer Instances tab. The HTTP Analyzer Instances window appears, as shown in [Figure 14-4](#).

This window provides information about the instances of the HTTP Analyzer that are currently running, or that were running and have been stopped. The instance is identified by the host and port, and any rules are identified. You can start and stop the instance from this window.

Figure 14-4 HTTP Analyzer Instances Window




You create a new instance in the HTTP Analyzer dialog, which opens when you click the **Create New Soap Request** button.

Table 14-2 HTTP Analyzer Instances Window Toolbar Icons

Icon	Name	Function
	Analyzer Preferences	Click to open the HTTP Analyzer dialog where you can specify a new listener port, or change the default proxy.
	Create New Request	Click to open a new instance of the HTTP Analyzer Test window, where you enter payload details, and edit and resend messages.
	Start HTTP Analyzer	Click to start the HTTP Analyzer running. The monitor runs in the background, and only stops when you click Stop or exit ODI. If you have more than one listener defined clicking this button starts them all. To start just one listener, click the down arrow and select the listener to start.

Table 14–2 (Cont.) HTTP Analyzer Instances Window Toolbar Icons

Icon	Name	Function
	Stop HTTP Analyzer	Click to stop the HTTP Analyzer running. If you have more than one listener running, clicking this button stops them all. To stop just one listener click the down arrow and select the listener to stop.

14.3.2.7 How to Use Multiple Instances

You can have more than one instance of HTTP Analyzer running. Each will use a different host and port combination, and you can see a summary of them in the HTTP Analyzer Instances window.

To add an additional HTTP Analyzer Instance:

1. Open the HTTP Analyzer preferences dialog by doing one of the following:
 - Click the Analyzer Preferences button in the HTTP Analyzer Instances window or Log window.
 - Choose **Tools > Preferences** to open the Preferences dialog, and navigating to the HTTP Analyzer page.

For more information at any time, press F1 or click Help from the HTTP Analyzer preferences dialog.

2. To create a new HTTP Analyzer instance, that is a new listener, click **Add**. The new listener is listed and selected by default for you to change any of the values.

14.3.2.8 Using Credentials With HTTP Analyzer

You can use the HTTP Analyzer to test Web services that are secured using policies. You choose the credentials to use in the HTTP Analyzer Test window.

HTTP Analyzer supports the following credentials for this purpose:

- HTTPS. The message is encrypted prior to transmission using a public key certificate that is signed by a trusted certificate authority. The message is decrypted on arrival.
- Username token. This token does not apply to Oracle Data Integrator.
This is a way of carrying basic authentication information using a token based on username/password.
- X509. This token does not apply to Oracle Data Integrator.
This is a PKI standard for single sign-on authentication, where certificates are used to provide identity, and to sign and encrypt messages.
- STS. This token does not apply to Oracle Data Integrator.
Security Token Service (STS) is a Web service which issues and manages security tokens.

14.3.2.9 Using SSL With HTTP Analyzer

You can use the HTTP Analyzer with secured services or applications, for example, Web services secured by policies. Oracle Data Integrator includes a credential, `HTTPS Credential`, for this purpose.

Once you have configured the credentials, you can choose which to use in the HTTP Analyzer Test window.

HTTPS encrypts an HTTP message prior to transmission and decrypts it upon arrival. It uses a public key certificate signed by a trusted certificate authority. When the integrated application server is first started, it generates a `DemoIdentity` that is unique, and the key in it is used to set up the HTTPS channel.

For more information about keystores and keystore providers, see *Understanding Security for Oracle WebLogic Server*.

When the default credential `HTTPS Credential` is selected, you need to specify the keystores that the HTTP Analyzer should use when handling HTTPS traffic.

Two keystores are required to run the HTTP Analyzer:

- The "Client Trusted Certificate Keystore," containing the certificates of all the hosts to be trusted by the Analyzer (client trust) when it makes onward connections. The server's certificate must be in this keystore.

The "Client Keystore" is required only when mutual authentication is required.

- The "Server Keystore," containing a key that the Analyzer can use to authenticate itself to calling clients.

To configure the HTTP Analyzer to use different HTTPS values:

1. From the main menu, choose **Tools > Preferences**.
2. In the Preferences dialog, select the Credentials node. For more information, press F1 or click **Help** from within the dialog page.
3. Enter the new keystore and certificate details you want to use.

14.3.2.10 How to Debug Web Pages Using the HTTP Analyzer

You can use the HTTP Analyzer when you are debugging Web pages, such as HTML, JSP, or JSF pages. This allows you to directly examine the traffic that is sent back and forth to the browser.

To debug Web pages using the HTTP Analyzer:

1. Configure a browser to route messages through the HTTP Analyzer so that you can see the traffic between the web browser and client.
2. Start the HTTP Analyzer running.
3. Run the class, application, or Web page that you want to analyze in the usual way.

Each request and response packet is listed in the HTTP Analyzer Log window, and detailed in the HTTP Analyzer Test Window.

14.3.2.11 How to Use Rules to Determine Behavior

You can set rules so that the HTTP Analyzer runs using behavior determined by those rules. You can set more than one rule in an HTTP Analyzer instance. If a service's URL matches a rule, the rule is applied. If not, the next rule in the list is checked. If the service does not match any of the rules the client returns an error. For this reason, you should always use a Pass Through rule with a blank filter (which just passes the request through) as the last rule in a list to catch any messages not caught by the preceding rules.

The types of rule available are:

- Pass Through Rule
- Forward Rule

- URL Substitution Rule
- Tape Rule

14.3.2.11.1 Using the Pass Through Rule The Pass Through simply passes a request on to the service if the URL filter matches. When you first open the Rule Settings dialog, two Pass Through Rules are defined:

- The first has a URL filter of `http://localhost:631` to ignore print service requests.
- The second has a blank URL filter, and it just passes the request to the original service. This rule should normally be moved to end of the list if new rules are added.

14.3.2.11.2 Using the Forward Rule The Forward rule is used to intercept all URLs matched by the filter and it forwards the request on to a single URL.

14.3.2.11.3 Using the URL Substitution Rule The URL Substitution rule allows you to re-host services by replacing parts of URL ranges. For example, you can replace the machine name when moving between the integrated application server and Oracle WebLogic Server.

14.3.2.11.4 Using the Tape Rule The tape rule allows you to run the HTTP Analyzer in simulator mode, where a standard WS-I log file is the input to the rule. When you set up a tape rule, there are powerful options that you can use:

- Loop Tape, which allows you to run the tape again and again.
- Skip to matching URL and method, which only returns if it finds a matching URL and HTTP request method. This means that you can have a WSDL and an endpoint request in the same tape rule.
- Correct header date and Correct Content Size, which allow you change the header date and content size of the message to current values so that the request does not fail.

An example of using a tape rule would be to test a Web service client developed to run against an external Web service.

To test a Web service client developed to run against an external Web service:

1. Create the client to the external Web service.
2. Run the client against the Web service with the HTTP Analyzer running, and save the results as a WS-I log file.

You can edit the WS-I file to change the values returned to the client.

3. In the HTTP Analyzer page of the Preferences dialog, create a tape rule.

Ensure that it is above the blank Pass Through rule in the list of rules.

4. In the Rule Settings dialog, use the path of the WS-I file as the Tape path in the Rule Settings dialog.

When you rerun the client, it runs against the entries in the WS-I file instead of against the external Web service.

There are other options that allow you to:

- Correct the time and size of the entries in the WS-I log file so the message returned to the client is correct.

- Loop the tape so that it runs more than once.
- Skip to a matching URL and HTTP request method, so that you can have a WSDL and an endpoint request in the same tape rule.

Note: Tape Rules will not work with SOAP messages that use credentials or headers with expiry dates in them.

14.3.2.12 How to Set Rules

You can set rules so that the HTTP Analyzer runs using behavior determined by those rules. Each analyzer instance can have a set of rules to determine behavior, for example, to redirect requests to a different host/URL, or to emulate a Web service.

To set rules for an HTTP Analyzer instance:

1. Open the HTTP Analyzer by choosing **Tools > HTTP Analyzer**. The HTTP Analyzer docked window opens.

Alternatively, the HTTP Analyzer automatically opens when you choose **Test Web Service** from the context menu of a Web service container in the Applications window.
2. Click the Analyzer Preferences button to open the HTTP Analyzer preferences dialog, in which you can specify a new listener port, or change the default proxy.

Alternatively, choose **Tools > Preferences**, and then navigate to the HTTP Analyzer page.
3. Click **Configure Rules** to open the Rule Settings dialog in which you define rules to determine the actions the HTTP Analyzer should take. For more help at any time, press F1 or click **Help** in the Rule Settings dialog.
4. In the Rule Settings dialog, enter the URL of the reference service you want to test against as the Reference URL. This will help you when you start creating rules, as you will be able to see if and how the rule will be applied.
5. Define one or more rules for the service to run the client against. To add a new rule, click the down arrow next to **Add**, and choose the type of rule from the list. The fields in the dialog depend on the type of rule that is currently selected.
6. The rules are applied in order from top to bottom. Reorder them using the up and down reorder buttons. It is important that the last rule is a blank Pass Through rule.

14.3.2.13 Reference: Troubleshooting the HTTP Analyzer

This section contains information to help resolve problems that you may have when running the HTTP Analyzer.

14.3.2.13.1 Running the HTTP Analyzer While Another Application is Running If you have an application waiting for a response, do not start or stop the HTTP Analyzer. Terminate the application before starting or stopping the HTTP Analyzer.

The HTTP Analyzer can use one or more different sets of proxy settings. These settings are specific to the IDE only. If enabled, Oracle Data Integrator uses these settings to access the Internet through your organization proxy server. If you do not enable the proxy server setting, then your Web application may not be able to access the Internet. Proxy server settings are visible in the preferences settings for your machine's default browser.

When you run the HTTP Analyzer, it can use one or more different sets of proxy settings. These proxy settings override the HTTP Proxy Server settings when the HTTP Analyzer is running.

14.3.2.13.2 Changing Proxy Settings When you use the HTTP Analyzer, you may need to change the proxy settings in Oracle Data Integrator. For example:

- If you are testing an external service and your machine is behind a firewall, ensure that Oracle Data Integrator is using the HTTP proxy server.
- If you are testing a service in the integrated application server, for example when you choose **Test Web Service** from the context menu of a Web service in the Applications window, ensure that Oracle Data Integrator is not using the HTTP proxy server.

If you run the HTTP Analyzer, and see the message

```
500 Server Error
The following error occurred: [code=CANT_CONNECT_LOOPBACK] Cannot connect due to
potential loopback problems
```

you probably need to add localhost|127.0.0.1 to the proxy exclusion list.

To set the HTTP proxy server and edit the exception list:

1. Choose **Tools > Preferences**, and select **Web Browser/Proxy**.
2. Ensure that **Use HTTP Proxy Server** is selected or deselected as appropriate.
3. Add any appropriate values to the Exceptions list, using | as the separator.

In order for Java to use localhost as the proxy ~localhost must be in the Exceptions list, even if it is the only entry.

14.3.3 Using the OdiInvokeWebService Tool

The OdiInvokeWebService tool invokes a Web service using the HTTP or HTTPS protocol and is able to write the returned response to an XML file, which can be an XML payload or a full-formed SOAP message including a SOAP header and body.

You can configure OdiInvokeWebService tool parameters using Http Analyzer. To do this, click the **Http Analyzer** button on the **General** tab of the OdiInvokeWebService step in the package editor. This opens the OdiInvokeWebServiceAdvance editor, which you can use to configure command parameters.

See "OdiInvokeWebService" in *Oracle Data Integrator Tool Reference* for details on the OdiInvokeWebService tool parameters.

The OdiInvokeWebService tool invokes a specific operation on a port of a Web service whose description file (WSDL) URL is provided. If this operation requires a SOAP request, it is provided either in a request file or in the tool command. The response of the Web service request is written to an XML file that can be used in Oracle Data Integrator.

Note: When using the XML payload format, the OdiInvokeWebService tool does not support the SOAP headers of the request. In order to work with SOAP headers, for example for secured Web service invocation, use a full SOAP message and manually modify the SOAP headers.

This tool can be used as a regular Oracle Data Integrator tool in a tool step of a package and also in procedures and knowledge modules. See "[Adding Oracle Data Integrator Tool Steps](#)" on page 7-6 for information on how to create a tool step in a package.

You can process the information from your responses using regular Oracle Data Integrator interfaces sourcing for the XML technology. Refer to the *Connectivity and Modules Guide for Oracle Data Integrator* for more information on XML file processing.

Note: Each XML file is defined as a model in Oracle Data Integrator. When using XML file processing for the request or response file, a model will be created for each request or response file. It is recommended to use model folders to arrange them. See "[Organizing Models with Folders](#)" on page 18-2 for more information.

Oracle Data Integrator provides the OdiXMLConcat and OdiXMLSplit tools for processing the Web service response. Refer to "XML" in *Oracle Data Integrator Tool Reference* for details on how to use these tools.

Using the Binding Mechanism for Requests

It is possible to use the Binding mechanism when using a Web service call in a Procedure. With this method, it is possible to call a Web service for each row returned by a query, parameterizing the request based on the row's values. Refer to "[Binding Source and Target Data](#)" on page 11-8 for more information.

This chapter gives an introduction to shortcuts and describes how to work with shortcuts in Oracle Data Integrator.

This chapter includes the following sections:

- [Introduction to Shortcuts](#)
- [Introduction to the Shortcut Editor](#)
- [Creating a Shortcut](#)
- [Working with Shortcuts in your Projects](#)

15.1 Introduction to Shortcuts

Oracle Data Integrator is often used for populating very large data warehouses sourcing from various versions of source applications. To express the large commonality that often exists between two different versions of the same source application, such as same tables and columns, same constraints, and same transformations, *shortcuts* have been introduced into Oracle Data Integrator. Shortcuts are created for common objects in separate locations. At deployment time, for example during an export from the design repository to the runtime repository, these shortcuts are *materialized* as final objects for the given version of the source application.

15.1.1 Shortcutting Concepts

A *shortcut* is a link to an Oracle Data Integrator object. You can create a shortcut for datastores, mappings, reusable mappings, packages, and procedures.

A *referenced object* is the object directly referenced by the shortcut. The referenced object of a shortcut may be a shortcut itself.

The *base object* is the original base object. It is the real object associated with the shortcut. Changes made to the base object are reflected in all shortcuts created for this object.


When a shortcut is *materialized*, it is converted in the design repository to a real object with the same properties as the ultimate *base object*. The materialized shortcut retains its name, relationships, and object ID.

Release tags have been introduced to manage the materialization of shortcuts based on specific tags. Release tags can be added to folders and models.

See "[Working with Shortcuts in your Projects](#)" on page 15-4 for more information.

15.1.2 Shortcut Objects

You can create a shortcut for the following ODI objects: datastores, mappings, packages, and procedures.

Shortcuts can be distinguished from the original object by the arrow that appears on the icon. For example, the  icon is a shortcut for a procedure.

Shortcut reference objects display the same nodes as the base objects in Designer Navigator.

Guidelines for Creating Shortcuts

Shortcuts are generally used like the objects they are referring to. However, the following rules apply when creating shortcuts for:

- *Datastores*: It is possible to create an object shortcut for datastores across different models/sub models but the source and destination models must be defined with the same technology. Also, a model cannot contain a datastore and a shortcut to another datastore with the same table name. Two shortcuts within a model cannot contain the same base object.

Datastore shortcuts can be used as sources or the target of a mapping and as datastores within a package. The mappings and packages containing datastore shortcuts refer to the datastore shortcut and the model in addition to the base datastore.

- *Packages, Mappings, Reusable Mappings, and Procedures*: It is possible to create an object shortcut for packages, mappings, and procedures belonging to a specific ODI folder.

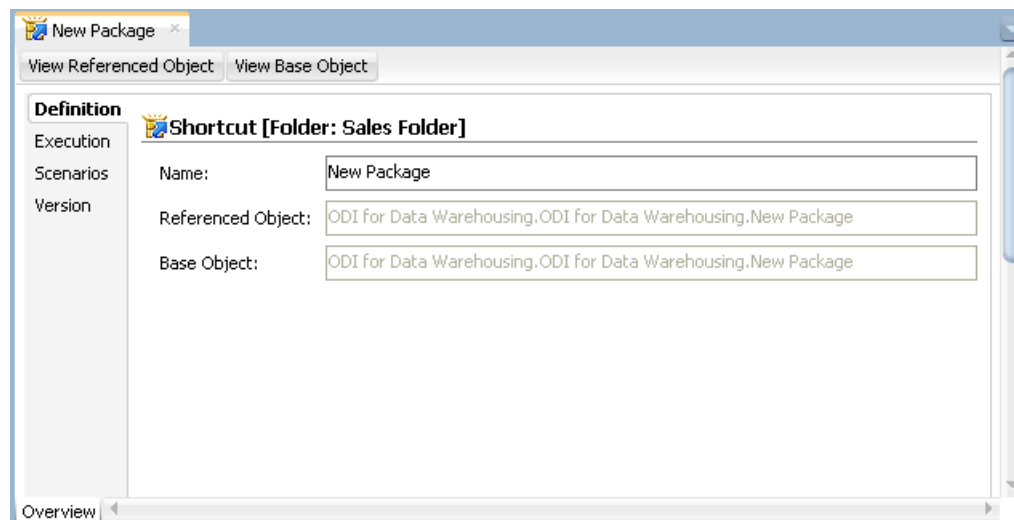
Mapping, procedure, and package shortcuts within a Project can only refer to objects (mappings, procedures, and packages) that belong to the same object.

- Package shortcuts can be used in Load Plan steps
- Mapping shortcuts can be used within a mapping, a reusable mapping, a package, or a Load Plan step
- Procedure shortcuts can be used in a package or a Load Plan step

When adding a shortcut to a Load Plan step, Oracle Data Integrator generates a scenario and adds it as a Run Scenario step.

15.2 Introduction to the Shortcut Editor

The Shortcut editor provides a single environment for editing and managing shortcuts in Oracle Data Integrator. [Figure 15–1](#) gives an overview of the Shortcut editor.

Figure 15–1 Shortcut Editor of a Package Shortcut

The Shortcut Editor has the following tabs:

- **Definition**
Includes the names of the shortcut, the referenced object, and the base object
- **Execution** (only for shortcuts of Packages, Mappings (other than reusable mappings), and Procedures)
Is organized into the **Direct Executions** and the **Scenario Execution** tabs and shows the results of previous executions
- **Scenarios** (only for shortcuts of Packages, Mappings (other than reusable mappings), and Procedures)
Displays in a table view the scenarios generated for this component
- **Version**
Includes the details necessary to manage versions of the shortcut

The Shortcut Editor provides two buttons to handle its references:

- **View Referenced Object:** Click to display the editor of the referenced object
- **View Base Object:** Click to display the editor of the base object

15.3 Creating a Shortcut

Shortcuts can have the same name as the base object. It is possible to rename a shortcut but note that the shortcut reference object name must be used instead of the base name for object usages and materialization purposes.

Shortcuts can be created for one or multiple objects at a time and also for the same object in more than one location.

Also note that two shortcut objects within a folder cannot refer to the same base object and follow the "[Guidelines for Creating Shortcuts](#)" on page 15-2.

To create a shortcut:

1. In Designer Navigator, select the object that you want to create a shortcut to.
Note that you can select multiple objects of the same type.

2. Right-click the object(s) and select **Copy**.
3. Go to the location where you want to insert the shortcut. This must be the parent of another folder or model.
4. Right-click the folder or model and select **Paste as Shortcut**.

Note that the menu item **Paste as Shortcut** is only enabled if:

- The previous operation was a **Copy** operation.
 - The copied object is an object for which shortcuts can be created. See "[Shortcut Objects](#)" on page 15-2 for more information.
 - The new location is legal for the copied objects. Legal locations are:
 - *For datastore shortcuts*: A model or sub-model node different of the source model
 - *For mapping, package, and procedure shortcuts*: A folder node in the same project as the source folder but not the source folder
5. The new shortcut appears in Designer Navigator.

Tip: It is possible to create several shortcuts at once. See "[Duplicating a Selection with Shortcuts](#)" on page 15-4 for more information.

15.4 Working with Shortcuts in your Projects

This section describes the actions that you can perform when you work with shorts in your Oracle Data Integrator projects. These actions include:

- [Duplicating a Selection with Shortcuts](#)
- [Jump to the Reference Shortcut](#)
- [Jump to the Base Object](#)
- [Executing Shortcuts](#)
- [Materializing Shortcuts](#)
- [Exporting and Importing Shortcuts](#)
- [Using Release Tags](#)
- [Advanced Actions](#)

15.4.1 Duplicating a Selection with Shortcuts

It is possible to create several shortcuts at once for the objects within a given model or folder.

- If you perform a quick massive shortcut creation on a *model node*, the new model will be a copy of the source model with all datastores created as shortcuts.
- If you perform a quick massive shortcut creation on a *folder node*, the new folder will be a copy of the source folder with all mappings, packages, and procedures created as shortcuts.

To perform a quick massive shortcut creation:

1. In Designer Navigator, select a folder or model node.
2. Right-click and select **Duplicate Selection with Shortcuts**.

15.4.2 Jump to the Reference Shortcut

Use this action if you want to move the current selection in Designer Navigator to the referenced object.

To jump to the referenced object:

1. In Designer Navigator, select the shortcut whose referenced object you want to find.
2. Right-click and select **Shortcut > Follow Shortcut**.

The referenced object is selected in Designer Navigator.

15.4.3 Jump to the Base Object

Use this action if you want to move the current selection in Designer Navigator to the base object.

To jump to the base object:

1. In Designer Navigator, select the shortcut whose base object you want to find.
2. Right-click and select **Shortcut > Jump to Base**.

The base object is selected in Designer Navigator.

15.4.4 Executing Shortcuts

Executing a shortcut executes the underlying procedure the shortcut is referring to. Shortcuts are executed like any other object in Oracle Data Integrator. See "Running Integration Processes" in *Administering Oracle Data Integrator*.

15.4.5 Materializing Shortcuts

When a shortcut is *materialized*, it is converted in the design repository to a real object with the same properties as the ultimate *base object*. The materialized shortcut retains its name, relationships, and object ID. All direct references to the shortcut are automatically updated to point to the new object. This applies also to release tags. If the materialized shortcut contained release tags, all references to the base object within the *release tag folder* or *model* would be changed to the new object.

Note: When materializing a mapping shortcut and the mapping has a multilogical schema environment, for example when the mapping has a staging area on the same logical schema as the source datastore(s), the materialized mapping might contain errors such as changes in the flow. Review the materialized mapping to check for errors.

15.4.6 Exporting and Importing Shortcuts

Shortcuts can be exported and imported either as materialized objects or as shortcuts.

Standard and multiple export do not support materialization. When using standard or multiple export, a shortcut is exported as a shortcut object. Any import will import the shortcut as a shortcut.

When you perform a Smart export and your export contains shortcuts, you can choose to materialize the shortcuts:

- If you select not to materialize the shortcuts, both the shortcuts and the base objects will be exported.
- If you select to materialize the shortcuts, the export file will contain the converted shortcuts as real objects with the same object ID as the shortcut. You can import this export file into a different repository or back to the original repository.
 - When this export file is imported into a different repository, the former shortcut objects will now be real objects.
 - When this export file is imported back into the original repository, the materialized object ID will be matched with the shortcut ID. Use the Smart import feature to manage this matching process. The Smart import feature is able to replace the shortcut by the materialized object.

See "[Smart Export and Import](#)" on page 23-14 for more information.

15.4.7 Using Release Tags

Release tags allow you to manage the materialization of shortcuts based on specific tags. You can also use release tags to organize your own metadata. Release tags can be added in form of a text string to folders and model folders.

Note the following concerning release tags:

- No two models may have the same release tag and logical schema. The release tag is set in the model and in the folder.
- The release tag is used only during materialization and export.
- The release tag on a folder applies only to the package, mapping, and procedure contents of the folder. The release tag is not inherited by any subfolder.

To add a new release tag or assign an existing release tag:

1. From the Designer Navigator toolbar menu, select **Edit Release Tag...**

This opens the Release Tag wizard.

2. In the Release Tag Name field, do one of the following:

- Enter a new release tag name.
- Select a release tag name from the list.

This release tag name will be added to a given folder.

3. The available folders are displayed on the left, in the **Available** list. From the **Available** list, select the folder(s) to which you wish to add the release tag and use the arrows to move the folder to the **Selected** list.

4. Click **Next** to add the release tag to a model.

You can click **Finish** if you do not want to add the release tag to a model.

5. The available models and model folders are displayed on the left, in the **Available** list. From the **Available** list, select the model(s) and/or model folder(s) to which you wish to add the release tag and use the arrows to move the model(s) and/or model folder(s) to the **Selected** list.

6. Click **Finish**.

The release tag is added to the selected project folders and models.

Tip: You can use release tags when performing a Smart Export by choosing to add all objects of a given release to the Smart Export. See ["Performing a Smart Export"](#) on page 23-14 for more information.

15.4.8 Advanced Actions

This section describes the advanced actions you can perform with shortcuts. Advanced actions include:

- [Data/View Data Action on a Datastore Shortcut](#)
- [Perform a Static Check on a Model, Submodel or Datastore Shortcut](#)
- [Review Erroneous Records of a Datastore Shortcut](#)
- [Generate Scenarios of a Shortcut](#)
- [Reverse-Engineer a Model that Contains Shortcuts](#)

Data/View Data Action on a Datastore Shortcut

You can perform a Data or View Data action on a datastore shortcut to view or edit the data of the underlying datastore the shortcut is referring to.

To view or edit the datastore's data the shortcut is referring to, follow the standard procedure described in ["Editing and Viewing a Datastore's Data"](#) on page 3-11.

Perform a Static Check on a Model, Submodel or Datastore Shortcut

You can perform a static check on a model, submodel or datastore shortcut. This performs a static check on the underlying object this shortcut is referring to.

To perform a static check on a model, submodel or datastore shortcut, follow the standard procedure described in ["Perform a Static Check on a Model, Sub-Model or Datastore"](#) on page 3-13.

Review Erroneous Records of a Datastore Shortcut

You can review erroneous records of the datastore a datastore shortcut is referring to.

To review erroneous records of the datastore shortcut, follow the standard procedure described in ["Reviewing Erroneous Records"](#) on page 3-14.

Generate Scenarios of a Shortcut

You can generate a scenario from mapping (but not reusable mapping), package, and procedure shortcuts. This generates a scenario of the underlying object this shortcut is referring to. Note that the generated scenario will appear under the shortcut and not the referenced object in Designer Navigator.

To generate a scenario of a shortcut, follow the standard procedure described in ["Generating a Scenario"](#) on page 12-2.

Reverse-Engineer a Model that Contains Shortcuts

You can reverse-engineer a model that contains shortcuts using the RKM Oracle. This Knowledge Module provides the option `SHORTCUT_HANDLING_MODE` to manage shortcuts that have the same table name as actual tables being retrieved from the database. This option can take three values:

- `ALWAYS_MATERIALIZE`: Conflicted shortcuts are always materialized and datastores are reversed (default).
- `ALWAYS_SKIP`: Conflicted shortcuts are always skipped and not reversed.

- PROMPT: The Shortcut Conflict Detected dialog is displayed. You can define how to handle conflicted shortcuts:
 - Select **Materialize**, to materialize and reverse-engineer the conflicted datastore shortcut.
 - Leave **Materialize** unselected, to skip the conflicted shortcuts. Unselected datastores are not reversed and the shortcut remains.

Note: When you reverse-engineer a model that contains datastore shortcuts and you choose to materialize the shortcuts, the reverse-engineering process will be incremental for database objects that have different attributes than the datastores shortcuts. For example, if the datastore shortcut has an attribute that does not exist in the database object, the attribute will not be removed from the reversed and materialized datastore under the assumption that the attribute is used somewhere else.

If you use any other RKM or standard reverse-engineering to reverse-engineer a shortcut model, the conflicted shortcuts will be materialized and the datastores reversed.

For more information on reverse-engineering, see [Chapter 3, "Creating and Using Data Models and Datastores."](#)

Using Groovy Scripting

This chapter provides an introduction to the Groovy language and explains how to use Groovy scripting in Oracle Data Integrator.

This appendix includes the following sections:

- [Introduction to Groovy](#)
- [Introduction to the Groovy Editor](#)
- [Using the Groovy Editor](#)
- [Automating Development Tasks - Examples](#)

16.1 Introduction to Groovy

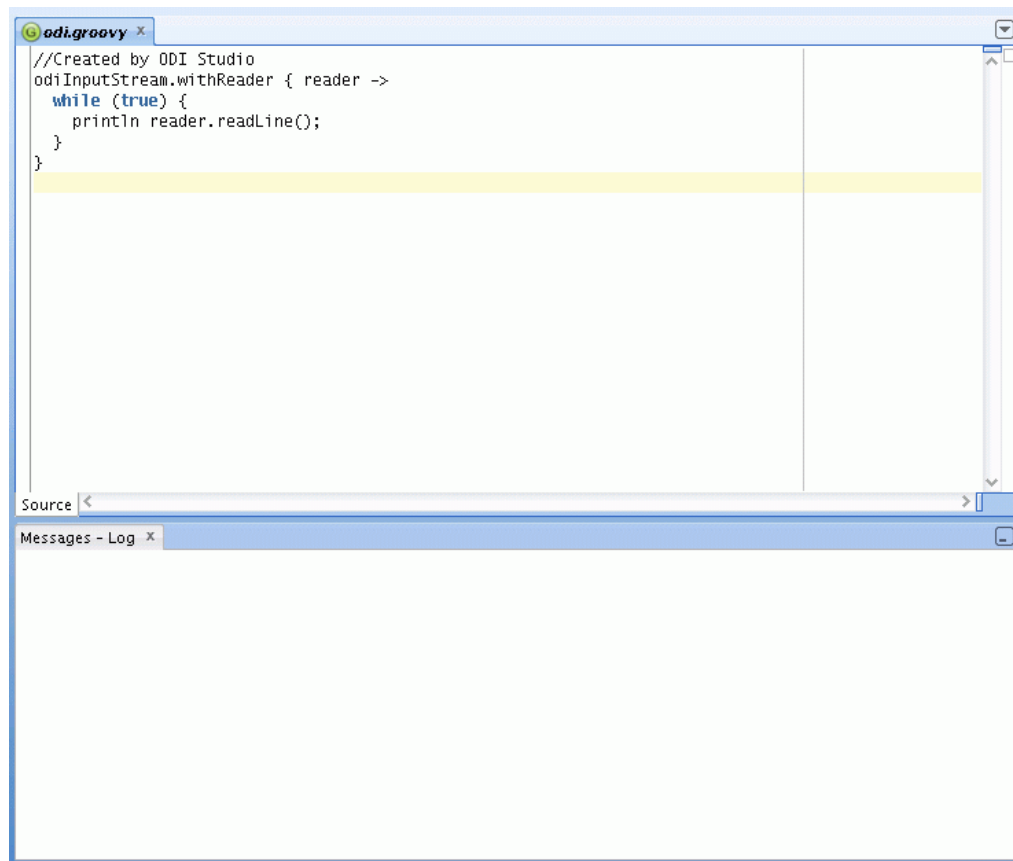
Groovy is a scripting language with Java-like syntax for the Java platform. The Groovy scripting language simplifies the authoring of code by employing dot-separated notation, yet still supporting syntax to manipulate collections, Strings, and JavaBeans.

For more information about the Groovy language, see the following web site:

<http://groovy.codehaus.org/>

16.2 Introduction to the Groovy Editor

The Groovy editor provides a single environment for creating, editing, and executing Groovy scripts within the ODI Studio context. [Figure 16–1](#) gives an overview of the Groovy editor.

Figure 16–1 Groovy Editor

The Groovy editor provides all standard features of a code editor such as syntax highlighting and common code editor commands except for debugging. The following commands are supported and accessed through the context menu or through the **Source** main menu:

- Show Whitespace
- Text Edits
 - Join Line
 - Delete Current Line
 - Trim Trailing Whitespace
 - Convert Leading Tabs to Spaces
 - Convert Leading Spaces to Tabs
- Indent Block
- Unindent Block
- Move Up
- Move Down

16.3 Using the Groovy Editor

You can perform the following actions with the Groovy editor:

- [Create a Groovy Script](#)
- [Open and Edit an Existing Groovy Script](#)
- [Save a Groovy Script](#)
- [Execute a Groovy Script](#)
- [Stop the Execution of a Groovy Script](#)
- [Perform Advanced Actions](#)

16.3.1 Create a Groovy Script

To create a Groovy script in ODI Studio:

1. From the **Tools** Main menu select **Groovy > New Script**.

This opens the Groovy editor.

2. Enter the Groovy code.

You can now save or execute the script.

16.3.2 Open and Edit an Existing Groovy Script

To edit a Groovy Script that has been previously created:

1. From the **Tools** Main menu select **Groovy > Open Script** or **Recent Scripts**.

2. Select the Groovy file and click **Open**.

This opens the selected file in the Groovy editor.

3. Edit the Groovy script.

You can now save or execute the script.

16.3.3 Save a Groovy Script

To save a Groovy script that is currently open in the Groovy editor:

From the **Tools** Main menu select **Groovy > Save Script** or **Save Script As**.

Note: The **Save** main toolbar option is not associated with the Groovy Editor.

16.3.4 Execute a Groovy Script

You can execute one or several Groovy scripts at once and also execute one script several times in parallel.

You can only execute a script that is opened in the Groovy editor. ODI Studio does not execute a selection of the script, it executes the whole Groovy script.

To execute a Groovy script in ODI Studio:

1. Select the script that you want to execute in the Groovy editor.
2. Click **Execute** in the toolbar.
3. The script is executed.

You can now follow the execution in the Log window.

Notes:

- Each script execution launches its own Log window. The Log window is named according to the following format: *Running <script_name>*
- Groovy writes output to two different streams. If it is a class, it writes to System.out, which is a global output stream. If it is from a script (non-class), then it creates one stream for every execution. This can be captured by ODI. So, only output written to a script is shown in the Log window.

You can add `System.setOut(out)` in the beginning of a Groovy script so that the messages printed by an external class can be redirected to messages log.

16.3.5 Stop the Execution of a Groovy Script

You can only stop running scripts. If no script is running, the Stop buttons are deactivated.

The execution of Groovy scripts can be stopped using two methods:

- **Clicking Stop in the Log tab.** This stops the execution of the particular script.
- **Click Stop on the toolbar.** If several scripts are running, you can select the script execution to stop from the drop down list.

16.3.6 Perform Advanced Actions

This section describes some advanced actions that you can perform with the Groovy editor.

Use Custom Libraries

The Groovy editor is able to access external libraries for example if an external driver is needed.

To use external libraries, do one of the following:

- Copy the custom libraries to the *userlib* folder. This folder is located:
 - On Windows operating systems:
%APPDATA%/odi/oracledi/userlib
 - On UNIX operating systems:
~/ .odi/oracledi/userlib
- Add the custom libraries to the *additional_path.txt* file. This file is located in the *userlib* folder and has the following content:

```
; Additional paths file
; You can add here paths to additional libraries
; Examples:
; C:\ java\libs\myjar.jar
; C:\ java\libs\myzip.zip
; C:\java\libs\*.jar will add all jars contained in the C:\java\libs\
directory
; C:\java\libs\**\*.jar will add all jars contained in the C:\java\libs\
directory and subdirectories
```

Define Additional Groovy Execution Classpath

You can define a Groovy execution classpath in addition to all classpath entries available to ODI Studio.

To define an additional Groovy execution classpath:

1. Before executing the Groovy script, select from the **Tools Main menu Preferences...**
2. In the Preferences dialog, navigate to the **Groovy Preferences** page.
3. Enter the classpath and click **OK**.

Note: You do not need to restart ODI Studio after adding or changing the classpath.

Read Input with `odiInputStream` Variable

Oracle Data Integrator provides the `odiInputStream` variable to read input streams. This variable is used as follows:

```
odiInputStream.withReader { println (it.readLine()) }
```

When this feature is used an Input text field is displayed on the bottom of the Log tab. Enter a string text and press ENTER to pass this value to the script. The script is exited once the value is passed to the script.

[Example 16–1](#) shows another example of how to use an input stream. In this example you can provide input until you click **Stop <script_name>**.

Example 16–1 *InputStream*

```
odiInputStream.withReader { reader ->
    while (true) {
        println reader.readLine();
    }
}
```

Using Several Scripts

If you are using several scripts at once, note the following:

- A log tab is opened for each execution.
- If a script is referring to another script, the output of the second will not be redirected to the log tab. This is a known Groovy limitation with no workaround.

Using the ODI Instance

Oracle Data Integrator provides the variable `odiInstance`. This variable is available for any Groovy script running within ODI Studio. It represents the ODI instance, more precisely the connection to the repository, in which the script is executed. Note that this instance will be null if ODI Studio is not connected to a repository.

The `odiInstance` variable is initialized by the ODI Studio code before executing the code. You can use bind APIs of the Groovy SDK for this purpose. [Example 16–2](#), "Creating a Project" shows how you can use the `odiInstance` variable.

16.4 Automating Development Tasks - Examples

Oracle Data Integrator provides support for the use of Groovy to automate development tasks. These tasks include for example:

- [Example 16–2, "Creating a Project"](#)
- [Example 16–3, "External Groovy File"](#)
- [Example 16–4, "Class from External File"](#)
- [Example 16–5, "For Studio UI Automation"](#)

[Example 16–2](#) shows how to create an ODI Project with a Groovy script.

Example 16–2 Creating a Project

```
import oracle.odi.core.persistence.transaction.ITransactionDefinition;
import
oracle.odi.core.persistence.transaction.support.DefaultTransactionDefinition;
import oracle.odi.core.persistence.transaction.ITransactionManager;
import oracle.odi.core.persistence.transaction.ITransactionStatus;
import oracle.odi.domain.project.OdiProject;
import oracle.odi.domain.project.OdiFolder;

ITransactionDefinition txnDef = new DefaultTransactionDefinition();
ITransactionManager tm = odiInstance.getTransactionManager()
ITransactionStatus txnStatus = tm.getTransaction(txnDef)
OdiProject myProject = new OdiProject("New Project 1", "NEW_PROJECT_1")
OdiFolder myFolder = new OdiFolder(myProject, "Test Folder 001")
odiInstance.getTransactionEntityManager().persist(myProject)
tm.commit(txnStatus)
```

[Example 16–3](#) shows how to import an external Groovy script.

Example 16–3 External Groovy File

```
//Created by ODI Studio
import gd.Test1;
println "Hello World"
Test1 t1 = new Test1()
println t1.getName()
```

[Example 16–4](#) shows how to call a class from a different Groovy script.

Example 16–4 Class from External File

```
import gd.GroovyTestClass

GroovyTestClass tc = new GroovyTestClass()
println tc.getClassLoaderName()
```

[Example 16–5](#) shows how to implement Studio UI automation.

Example 16–5 For Studio UI Automation

```
import javax.swing.JMenuItem;
import javax.swing.JMenu;
import oracle.ide.Ide;

((JMenuItem) Ide.getMenuBar().getGUI(false).getComponent(4)).doClick();
```

```
JMenu mnu = ((JMenu)Ide.getMenubar().getGUI(false).getComponent(4));  
((JMenuItem)mnu.getMenuComponents()[0]).doClick()
```


Part IV

Managing Integration Projects

This part describes how to organize and maintain your Oracle Data Integrator projects.

This part contains the following chapters:

- [Chapter 17, "Exchanging Global ODI Objects"](#)
- [Chapter 18, "Organizing and Documenting Integration Projects"](#)
- [Chapter 19, "Using Version Control \(Legacy Mode\)"](#)
- [Chapter 20, "Integrating ODI with Version Control Systems"](#)
- [Chapter 21, "Release Management"](#)
- [Chapter 22, "Life Cycle Management Guidelines"](#)
- [Chapter 23, "Exporting and Importing"](#)

Exchanging Global ODI Objects

This chapter describes how to use the Check for Updates wizard to browse and download global ODI objects.

17.1 Using the Check for Updates Wizard

You can browse and download the global user-functions, global KMs, and custom mapping components using the Check for Updates wizard. The Check for Updates wizard allows you to browse, download, and update the ODI objects that are available for distribution by Oracle and by ODI users in separate Update Centers. Each ODI object is packaged into an update bundle and placed in an Update Center.

Note: The ODI Objects that are provided through the Customers and Partners Update Center are not supported by Oracle.

To check for updates using the Check for Updates wizard:

1. Click **Help> Check for Updates**. The Check for Updates wizard appears.
2. On the Source panel, select the update centers that you want to search from those listed under **Search Update Centers**.

If required, you can add new update centers, remove newly added update centers, or edit the update centers.

Note: An Internet connection is required to search web-based update centers. You can click **Proxy Settings** to define the proxy settings for your Internet connection.

If there are issues connecting to the Internet, the Unable to Connect dialog is displayed. You can verify the proxy settings in this dialog and retry.

3. Click **Next**.
4. On the Updates panel, select the updates that you want to install.
Select **Show Upgrades Only** to see updates only for those ODI objects that are already installed.
5. Click **Next**.

6. On the Download panel, the progress of the updates being downloaded is displayed. When the download completes, a summary of the downloaded updates is displayed.
7. On the Summary panel, review the downloaded updates.
8. Click **Finish** to install the downloaded updates.

Note: After importing a Global KM or User Function, you must use **Refresh** to see it in the Designer navigator. After importing a Custom Mapping Component, you must exit and restart ODI to see it in the mapping components palette, if a mapping has already been opened in the current ODI session.

Organizing and Documenting Integration Projects

This chapter describes how to organize and document your work in Oracle Data Integrator.

This chapter includes the following sections:

- [Organizing Projects with Folders](#)
- [Organizing Models with Folders](#)
- [Using Cross-References](#)
- [Using Markers and Memos](#)
- [Handling Concurrent Changes](#)
- [Creating PDF Reports](#)

18.1 Organizing Projects with Folders

Before you begin creating an integration project with Oracle Data Integrator, it is recommended to think about how the project will be organized.

Rearranging your project afterwards may be dangerous. You might have to redo all the links and cross-references manually to reflect new locations.

Within a project, mappings, procedures and packages are organized into folders and sub-folders. It is recommended to maintain your project well organized by grouping related project components into folders and sub-folders according to criteria specific to the project. Folders simplify finding objects developed in the project and facilitate the maintenance tasks. Sub-folders can be created to an unlimited number of levels.

Note that you can also use markers to organize your projects. Refer to "[Using Markers and Memos](#)" on page 18-6 for more information.

18.1.1 Creating a New Folder

To create a new folder:

1. In Designer Navigator expand the **Projects** accordion.
2. Select the project into which you want to add a folder.
3. Right-click and select **New Folder**.
4. In the **Name** field, enter a name for your folder.
5. Select **Save** from the File main menu.

The empty folder appears.

To create a sub-folder:

1. Create a new folder, as described in ["Creating a New Folder"](#) on page 18-1.
2. Drag and drop the new folder into the parent folder.

18.1.2 Arranging Project Folders

To arrange your project folders in the project hierarchy, drag and drop a folder into other folders or on the Project. Note that it is not possible to move a folder from one Project to another Project.

18.2 Organizing Models with Folders

A model folder groups related models according to criteria specific to the project. A model folder may also contain other model folders. Sub-folders can be created to an unlimited number of levels.

Note that you can also use markers to organize your models. Refer to ["Using Markers and Memos"](#) on page 18-6 for more information.

18.2.1 Creating a New Model Folder

To create a model folder:

1. In Designer Navigator expand the **Models** accordion.
2. Click **New Model Folder** in the toolbar of the **Models** accordion.
3. In the **Name** field, enter a name for your folder.
4. Select **Save** from the File main menu.

The empty model folder appears.

18.2.2 Arranging Model Folders

To move a model into a folder:

1. In Designer Navigator expand the **Models** accordion.
2. Select the model, then drag and drop it on the icon of the destination model folder.

The model moves from its current location to the selected model folder.

Note the following when arranging model folders:

- A model can only be in one folder at a time.
- Model folders can be also moved into other model folders.

18.2.3 Creating and Organizing Sub-Models

A sub-model is an object that allows you to organize and classify the datastores of a model in a hierarchical structure. The root of the structure is the model. A sub-model groups functionally homogeneous datastores within a model. The datastores of a model can be inserted into a sub-model using drag and drop, or by automatic distribution.

The classification is performed:

- During the reverse-engineering process, the RKM may create sub-models and automatically distribute datastores into these sub-models. For example RKM handling large data models from ERP systems use this method.
- Manually, by drag and dropping existing datastores into the sub-models.
- Automatically, using the distribution based on the datastore's name.

To create a sub-model:

1. In Designer Navigator expand the **Models** accordion.
2. In the **Models** accordion, select the model or the sub-model into which you want to add a sub-model.
3. Right-click and select **New Sub-Model**.
4. On the Definition tab, enter a name for your sub-model in the **Name** field.
5. Click **OK**.

The new sub-model is created with no datastore.

Arranging Sub-Models

To manually file a datastore into a sub-model:

1. In Designer Navigator expand the **Models** accordion.
2. In the **Models** accordion, select the datastore you want to move into the sub-folder.
3. Drag and drop it into the sub-model.

The datastore disappears from the model and appears in the sub-model.

Setting-up Automatic Distribution

Distribution allows you to define an automatic distribution of the datastores in your sub-models.

Datastore names are compared to the automatic assignment mask. If they match this pattern, then they are moved into this sub-model. This operation can be performed manually or automatically depending on the Datastore Distribution Rule.

There are two methods to classify:

- By clicking **Distribution** in the Distribution tab of a sub-model, the current rule is applied to the datastores.
- At the end of a reverse-engineering process, all sub-model rules are applied, the order defined by the **Order of mask application after a Reverse Engineer** values for all sub-models.

To set up the automatic distribution of the datastores in a sub-model:

1. In the sub-model's Distribution tab, select the **Datastore distribution rule**:

The Datastore Distribution rule determines which datastores will be taken into account and compared to the automatic assignment mask:

- **No automatic distribution:** No datastore is taken in account. Distribution must be made manually.
- **Automatic Distribution of all Datastores not already in a Sub-Model:** Datastores located in the root model in the sub-model tree are taken in account.

- **Automatic Distribution of all Datastores:** All datastores in the model (and sub-models) are taken in account.
- 2. In the **Automatic Assignment Mask** field, enter the pattern the datastore names must match to be classified into this sub-model.
- 3. In the **Order of mask application after a Reverse Engineer** field, enter the order in which all rules should be applied at the end of a reverse. Consequently, a rule with a high order on all datastores will have precedence. A rule with a high order on non-classified datastores will apply only to datastores ignored by the other rules' patterns. At the end of the reverse, new datastores are considered non classified. Those already classified in a sub-model stay attached to their sub-model.
- 4. Click **Distribution**. The current rule is applied to the datastores.

18.3 Using Cross-References

Objects in Oracle Data Integrator (datastores, models, mappings, etc.) are interlinked by relationships varying from simple usage associations (a mapping uses Knowledge Modules) to complex ones such as code-interpretation relationships (a variable is used in the mappings or filters of a mapping). These relationships are implemented as cross-references. They are used to check/maintain consistency between related objects within a work repository. Cross-references, for example, prevent you from deleting an object if it is currently referenced elsewhere in the work repository.

Not all relationships appear as cross-references:

- Relationships with objects from the master repository (For example, a data model is related to a technology) are not implemented as cross-references, but as loose references based on object codes (context code, technology code, datatype code, etc). Modifications to these codes may cause inconsistency in the references.
- Strong relationships in the work repository (a folder belongs to a project) are enforced in the graphical user interface and within the repository (in the host database as foreign keys). These relationships may not normally be broken.

18.3.1 Browsing Cross-References

When modifying an object, it is necessary to analyze the impact of these changes on other developments. For example, if the length of a column is altered, the mappings using this column as a source or a target may require modification. Cross-references enable you to immediately identify the objects referenced or referencing a given object, and in this way provide effective impact analysis.

Cross-references may be browsed in Designer Navigator as described in [Table 18–1](#).

Table 18–1 *Cross-References in Designer Navigator*




Accordion	Icon	Description
Projects and Other accordion		The Uses and Used by nodes appear under an object node. The Uses node lists the objects from which the current object is referenced. In the case of a variable, for example, the packages containing steps referencing this variable and the mappings, filters, etc. will be displayed. The Used by node lists the objects that are using the current object.

Table 18–1 (Cont.) Cross-References in Designer Navigator

Accordion	Icon	Description
Models accordion		The Uses node appears under an object node and lists the objects referencing the current datastore, model or sub-model as a source or a target of a mapping, or in package steps.
Models accordion		The Used to Populate and Populated By nodes display the datastores used to populate, or populated by, the current datastore

These cross-referenced nodes can be expanded. The referencing or referenced objects can be displayed or edited from the cross-reference node.

18.3.2 Resolving Missing References

When performing version restoration operations, it may happen that an object in the work repository references nonexistent objects. For example, restoring an old version of a project without restoring all the associated objects used in its procedures or packages.

Note: The mapping framework has built into its design a mechanism for dealing with missing or invalid references. Because of this, ODI import in 12c will not create and report the import missing references for mappings. Instead, the mapping framework attempts to reconcile invalid or missing references when a mapping is opened.

Such a situation causes **Missing References** errors messages in Oracle Data Integrator when opening the objects (for example, a package) which references nonexistent objects. An object with missing cross-references is marked in the tree with the missing reference marker and its parent objects are flagged with a warning icon.

To display the details of the missing references for an object:

1. In Designer Navigator, double-click the object with the missing reference marker.
2. The object editor opens. In the object editor, select the Missing References tab.
3. The list of referenced objects missing for the cross-references is displayed in this tab.

To resolve missing references:

Missing cross-reference may be resolved in two ways:

- By importing/restoring the missing referenced object. See [Chapter 19, "Using Version Control \(Legacy Mode\)"](#), and [Chapter 23, "Exporting and Importing,"](#) for more information.
- By modifying the referencing object in order to remove the reference to the missing object (for example, remove the Refresh Variable step referencing the nonexistent variable from a package, and replace it with another variable).

Note: If a block of code (such a procedure command) contains one or more missing references, the first change applied to this code is considered without any further check. This is because all the missing references are removed when the code is changed and the cross-references computed, even if some parts of the code are still referring to an object that doesn't exist.

18.4 Using Markers and Memos

Almost all project and model elements may have descriptive markers and memos attached to them to reflect your project's methodology or help with development.

18.4.1 Markers

Flags are defined using markers. These markers are organized into groups, and can be applied to most objects in a project or a models.

Typical marker groups are:

- The development cycle (development, test, production)
- Priorities (low, medium, urgent, critical)
- Progress (10%, 20%, etc)

Global and Project Markers

Markers are defined in a project or in the Other view (Global Markers). The project markers can be used only on objects of the project, and global markers can be used in all models of the repository.

Flagging Objects

To flag an object with an icon marker:

1. In Designer Navigator, select an object in the Projects or Models accordion.
2. Right-click and select **Add Marker**, then select the marker group and the marker you want to set.

The marker icon appears in the tree. The marked object also appears under the marker's node. You can thus see all objects having a certain marker.

If you click in the tree an icon marker belonging to an auto-incremented marker group, you switch the marker to the next one in the marker group, and the icon changes accordingly.

Note: Markers will not appear if the option **Show Markers and Memo Flags** is not checked. See "[Hiding Markers and Memos](#)" on page 18-7 for more information.

To flag an object with string, numeric and date markers:

1. In Designer Navigator, double-click the object in the Projects or Models accordion.
2. In the object editor, select the Markers tab.
3. Click **Insert a Marker**.
4. In the new line, select the Group and Marker. You may also set the Value.

If the marker has an associated icon, it appears in the tree.

Filtering Using Markers

Markers can be used for informational purposes (for example, to have a global view of a project progress and resources). They can also be used when automating scenario generation by filter the packages. See "[Generating a Group of Scenarios](#)" on page 12-3 for more information.

The list of all objects using a certain marker is shown below the marker's node.

Customizing Markers

A new project is created with default markers. It is possible to customize the markers for a specific project as well as the global markers.

To define a marker group:

1. In Designer Navigator, click the **Markers** node in the Project accordion, or the **Global Markers** node in the Others accordion.
2. Right-click and select **New Marker Group**.
3. In the Group Name field, enter the name for the marker group, then define its Display Properties and Attributes.
4. Click **Insert a new Marker** to create a new marker in the group.
5. Select the marker **Icon**. If a marker stores date or a number, the icon should be set to **<none>**.
6. Select the marker **Name**, **Type** and other options.
7. Repeat operations 4 to 6 to add more markers to the group.
8. Select **Save** from the File main menu.

18.4.2 Memos

A memo is an unlimited amount of text attached to virtually any object, visible on its Memo tab. When an object has a memo attached, the memo icon appears next to it.

To edit an object's memo:

1. Right-click the object.
2. Select **Edit Memo**.
3. The Object editor opens, and the Memo tab is selected.

Hiding Markers and Memos

You can temporarily hide all markers and memo flags from the tree views, to improve readability.

To hide all markers and memo flags:

Deselect the **Display Markers and Memo Flags** option in the Designer Navigator toolbar menu. This preference is stored on a per-machine basis.

18.5 Handling Concurrent Changes

Several users can work simultaneously in the same Oracle Data Integrator project or model. As they may be all connected to the same repository, the changes they perform are considered as concurrent.

Oracle Data Integrator provides two methods for handling these concurrent changes: "[Concurrent Editing Check](#)" on page 18-8 and "[Object Locking](#)" on page 18-8. These two methods can be used simultaneously or separately.

18.5.1 Concurrent Editing Check

The user parameter, **Check for concurrent editing**, can be set to prevent you from erasing the work performed by another user on the object you try to save. You can set this parameter by clicking **Preferences** from the **Tools** option on the menu bar; expand the **ODI** node, and then the **System** node, and select the **Concurrent Development** node.

If this parameter is checked, when saving changes to any object, Oracle Data Integrator checks whether other changes have been made to the same object by another user since you opened it. If another user has made changes, the object cannot be saved, and you must cancel your changes.

18.5.2 Object Locking

The object locking mechanism can be activated in Oracle Data Integrator automatically, when closing the Oracle Data Integrator, or manually, by explicitly locking and unlocking objects.

As long as an object is locked, only the user owning the lock can perform modifying the object, such as editing or deleting. Other operations, such as executing, can be performed by other users, but with a warning.

Automatic Object Locking

Automatic object locking causes objects to be locked whenever you open them for editing. Optionally, you can configure the system to ask whether to lock an object when it is opened in a user interface, by generating a dialog.

An object locked by you appears with a yellow lock icon. An object locked by another user appears with a red lock icon.

When the edition window is closed, a popup window appears to ask if you want to unlock the object.

These windows are configured by the **Lock object when opening** and **Unlock object when closing** user parameters. You can set these parameters by clicking **Preferences** from the **Tools** option on the menu bar; expand the **ODI** node, and then the **System** node, and select the **Concurrent Development** node. You can set each option to **Yes**, **No**, or **Ask**. If you set **Lock object when opening** to **Ask**, a dialog is opened. If you set it to **Yes**, objects are automatically locked when opened.

Releasing locks when closing the user interface

When closing Oracle Data Integrator, by default a window appears asking to unlock or save objects that you have locked or kept opened. This behavior is controlled by the **Unlock object when closing** parameter.

You can keep objects locked even if you are not connected to Oracle Data Integrator. This allows you to prevent other users from editing them in the meanwhile.

Managing locks manually

You can also manually manage locks on objects.

To manually lock an object:

1. Select the object in the tree.
2. Right-click, then select **Locks > Lock**.

A lock icon appears after the object in the tree.

To manually unlock an object:

1. Select the object in the tree
2. Right-click, then select **Locks > Unlock**.

The lock icon disappears in the tree.

To manage all locks:

1. Select **Locked objects** from the ODI menu.
2. The Locked Objects editor appears displaying all locked objects that you can unlock.

Note: A user with the Supervisor privilege can remove locks for all other users.

18.6 Creating PDF Reports

In Oracle Data Integrator you have the possibility to print and share several types of reports with the PDF generation feature:

- Topology reports of the physical architecture, the logical architecture, or the contexts
- Reports of the version comparison results.
- Reports of an ODI object
- Diagram reports

Note: In order to view the generated reports, you must specify the location of Adobe Acrobat Reader in the user preferences before launching the PDF generation. To set this value, select the **Preferences...** option from the **Tools** menu. Expand the **ODI** node, and then the **System** node, and select **Reports**. Enter (or search for) the location of your preferred **PDF Viewer**.

18.6.1 Generating a Topology Report

Oracle Data Integrator provides the possibility to generate Topology reports in PDF format of the physical architecture, the logical architecture or the contexts.

To generate a topology report:

1. From the Topology Navigator toolbar menu select **Generate Report** and then the type of report you wish to generate:
 - Physical Architecture
 - Logical Architecture
 - Contexts
2. In the Report generation editor, enter the output **PDF file location** for your PDF report. Note that if no PDF file location is specified, the report in Adobe™ PDF

format is generated in your default directory for PDF generation specified in the user parameters. To set this value, select the **Preferences...** option from the **Tools** menu. Expand the **ODI** node, and then the **System** node, and select **Reports**. Enter (or search for) the location of your **Default PDF generation directory**.

3. If you want to view the PDF report after generation, select the **Open file after the generation?** option.
4. Click **Generate**.

18.6.2 Generating a Report for the Version Comparison Results

You can create and print a report of your comparison results via the Version Comparison Tool. Refer to ["Generating and Printing a Report of your Comparison Results"](#) on page 19-6 for more information.

18.6.3 Generating a Report for an Oracle Data Integrator Object

In Designer Navigator you can generate different types of reports depending on the type of object. [Table 18-2](#) lists the different report types for ODI objects.

Table 18-2 *Different report types for ODI objects*

Object	Reports
Project	Knowledge Modules
Project Folder	Folder, Packages, mappings, Procedures
Model Folder	Model Folder
Model	Model
Sub-model	Sub-model

To generate a report in Designer Navigator:

1. In Designer Navigator, select the object for which you wish to generate a report.
2. Right-click and select **Print >Print <object>**.
3. In the Report generation editor, enter the output **PDF file location** for your PDF report. Note that if no PDF file location is specified, the report in Adobe™ PDF format is generated in your default directory for PDF generation specified in the user parameters. To set this value, select the **Preferences...** option from the **Tools** menu. Expand the **ODI** node, and then the **System** node, and select **Reports**. Enter (or search for) the location of your **Default PDF generation directory**.
4. If you want to view the PDF report after generation, select the **Open file after the generation?** option.
5. Click **Generate**.

18.6.4 Generating a Diagram Report

You can generate a complete PDF report of your diagram. Refer to ["Printing a Diagram"](#) on page 5-4 for more information.

Using Version Control (Legacy Mode)

This chapter describes how to work with version management in Oracle Data Integrator.

Note: This chapter is applicable only for the legacy mode of versioning in ODI. If you want to enable VCS for Subversion, then none of the description from this chapter is applicable and you must refer to "[Integrating ODI with Version Control Systems](#)".

Oracle Data Integrator provides a comprehensive system for managing and safeguarding changes. The version management system allows **flags** on developed objects (such as projects, models, etc) to be automatically set, to indicate their status, such as new or modified. It also allows these objects to be backed up as stable checkpoints, and later restored from these checkpoints. These checkpoints are created for individual objects in the form of **versions**, and for consistent groups of objects in the form of **labels**.

Note: Version management is supported for master repositories installed on database engines such as Oracle, Hypersonic SQL, and Microsoft SQL Server. For a complete list of certified database engines supporting version management refer to the Platform Certifications document on OTN at:
<http://www.oracle.com/technology/products/oracle-data-integrator/index.html>.

This chapter includes the following sections:

- [Working with Object Flags](#)
- [Working with Versions](#)
- [Working with the Version Comparison Tool](#)
- [Working with Labels](#)

19.1 Working with Object Flags

When an object is created or modified in Designer Navigator, a flag is displayed in the tree on the object icon to indicate its status. [Table 19-1](#) lists these flags.

Table 19–1 Object Flags

Flag	Description
i	Object status is inserted.
u	Object status is updated.

When an object is inserted, updated or deleted, its parent objects are recursively flagged as updated. For example, when a step is inserted into a package, it is flagged as **inserted**, and the package, folder(s) and project containing this step are flagged as **updated**.

When an object version is checked in (refer to "[Working with Versions](#)" on page 19-2 for more information), the flags on this object are reset.

19.2 Working with Versions

A **version** is a backup copy of an object. It is checked in at a given time and may be restored later. Versions are saved in the master repository. They are displayed in the Version tab of the object window.

The following objects can be checked in as versions:

- Projects, Folders
- Packages, Scenarios
- Mappings (including Resuable Mappings), Procedures, Knowledge Modules
- Sequences, User Functions, Variables
- Models, Model Folders
- Labels
- Load Plans

Checking in a version

To check in a version:

1. Select the object for which you want to check in a version.
2. In the property inspector, select the **Version** tab. In the **Versions** table, click the **Create a new version** button (a green plus-sign icon). Alternatively, right-click the object and select **Version**, and then **Create Version**, from the context menu.
3. In the **Versioning** dialog, review **Previous Versions** to see the list of versions already checked in.
4. A version number is automatically generated in the **Version** field. Modify this version number if necessary.
5. Enter the details for this version in the **Description** field.
6. Click **OK**.

When a version is checked in, the flags for the object are reset.

Displaying previous versions of an object

To display previous versions of an object:

When editing the object, the Version tab provides creation and update information, the internal and global IDs for this object, and a list of versions checked in, with the check in date and the name of the user who performed the check in operation.

Restoring a version from the Version tab

Note: You can also restore a version from the Version Browser. See: ["Restoring a version with the Version Browser"](#) on page 19-4.

WARNING: Restoring a version cannot be undone. It permanently erases the current object and replaces it by the selected version. Consider creating a new version of your current object before restoring a version.

To restore a version from the **Version** tab:

1. Select the object for which you want to restore a version.
2. In the property inspector, select the **Version** tab. Alternatively, right-click the object and select **Version**, and then **Restore...**, from the context menu.

In the **Versions** table, select the row corresponding to the version you want to restore. Click the **Restore a version** button, or right-click the row and select **Restore** from the context menu.

3. Click **Yes** to confirm the restore operation.

Browsing versions

To browse versions:

Oracle Data Integrator contains a tool, the Version Browser, which is used to display the versions stored in the repository.

1. From the main menu, select **ODI > Version Browser...**
2. Use the **Object Type** and **Object Name** drop down lists to filter the objects for which you want to display the list of versions.

From the Version Browser, you can compare two versions, restore a version, export a version as an XML file or delete an existing version.

Note: The Version Browser displays the versions that existed when you opened it. Click **Refresh** to view all new versions created since then.

Comparing two versions with the Version Browser

To compare two versions with the Version Browser, see ["Working with the Version Comparison Tool"](#) on page 19-4.

Deleting a version with the Version Browser

To delete a version with the Version Browser:

1. Open the Version Browser.
2. Select the version you want to delete.

3. Click the **Delete** icon in the version table (a red X button), or right-click and select **Delete** from the context menu.

The version is deleted.

Restoring a version with the Version Browser

WARNING: Restoring a version cannot be undone. It permanently erases the current object and replaces it by the selected version. Consider creating a new version of your current object before restoring a version.

To restore a version with the Version Browser:

1. Open the Version Browser.
2. Select the version you want to restore.
3. Click the **Restore this version** button, or right-click and select **Restore** from the context menu.
4. Click **OK** to confirm the restore operation.

The version is restored in the repository.

Exporting a version with the Version Browser

To export a version with the Version Browser:

This operation exports the version to a file without restoring it. This exported version can be imported into another repository.

Note: Exporting a version exports the object contained in the version and not the version information. This allows you to export an old version without having to actually restore it in the repository.

1. Open the Version Browser.
2. Select the version you want to export.
3. Click the **Export this version as an XML file** button, or right-click and select **Export** from the context menu.
4. Select the **Export Directory** and specify the **Export Name**. Select **Replace existing files without warning** to overwrite files of the same name in the export directory without confirmation.
5. Click **OK**.

The version is exported to the given location.

19.3 Working with the Version Comparison Tool

Oracle Data Integrator provides a comprehensive version comparison tool. This graphical tool is to view and compare two different versions of an object.

The version comparison tool provides the following features:

- **Color-coded side-by-side display of comparison results:** The comparison results are displayed in two panes, side-by-side, and the differences between the two compared versions are color coded.
- **Comparison results organized in tree:** The tree of the comparison tool displays the comparison results in a hierarchical list of node objects in which expanding and collapsing the nodes is synchronized.
- **Report creation and printing in PDF format:** The version comparison tool is able to generate and print a PDF report listing the differences between two particular versions of an object.
- **Supported objects:** The version comparison tool supports the following objects: Project, Folder, Package, Scenario, Mapping, Procedure, Knowledge Module, Sequence, User Function, Variable, Model, Model folder, and Label.
- **Difference viewer functionality:** This version comparison tool is a difference viewer and is provided only for consultation purposes. Editing or merging object versions is not supported. If you want to edit the object or merge the changes between two versions, you have to make the changes manually directly in the concerned objects.

19.3.1 Viewing the Differences between two Versions

To view the differences between two particular versions of an object, open the Version Comparison tool.

There are three different way of opening the version comparison tool:

By selecting the object in the Projects tree

1. From the Projects tree in Designer Navigator, select the object whose versions you want to compare.
2. Right-click the object.
3. Select **Version > Compare with version...**
4. In the Compare with version editor, select the version with which you want to compare the current version of the object.
5. Click **OK**.
6. The Version Comparison tool opens.

Using the Versions tab of the object

1. In Designer Navigator, open the object editor of the object whose versions you want to compare.
2. Go to the Version tab.
The Version tab provides the list of all versions created for this object. This list also indicates the creation date, the name of the user who created the version, and a description (if specified).
3. Select the two versions you want to compare by keeping the <CTRL> key pressed.
4. Right-click and select **Compare...**
5. The Version Comparison tool opens.

Using the Version Browser

1. Open the Version Browser.

2. Select two versions that you want to compare, by using control-left click to multi-select two different rows. You must select rows that correspond to the exact same object.
3. Click the **Compare two versions for identical objects** icon in the version table, or right-click and select **Compare** from the context menu.
4. The Version Comparison tool opens.
5. To print a copy of the object comparison, click **Print**. See: "[Generating and Printing a Report of your Comparison Results](#)" on page 19-6.

Click **Close** when you are done reviewing the comparison.

The Version Comparison tool shows the differences between two versions: on the left pane the newer version and on the right pane the older version of your selected object.

The differences are color highlighted. The following color code is applied:

Color	Description
White (default)	unchanged
Red	deleted
Green	added/new
Yellow	object modified
Orange	field modified (the value inside of this fields has changed)

Note: If one object does not exist in one of the versions (for example, when it has been deleted), it is represented as an empty object (with empty values).

19.3.2 Using Comparison Filters

Once the version of an object is created, the Version Comparison tool can be used at different points in time.

Creating or checking in a version is covered in "[Working with Versions](#)" on page 19-2.

The Version Comparison tool provides two different types of filters for customizing the comparison results:

- **Object filters:** By selecting the corresponding check boxes (**New** and/or **Deleted** and/or **Modified** and/or **Unchanged**) you can decide whether you want only newly added and/or deleted and/or modified and/or unchanged objects to be displayed.
- **Field filters:** By selecting the corresponding check boxes (**New** and/or **Deleted** and/or **Modified** and/or **Unchanged**) you can decide whether you want newly added fields and/or deleted fields and/or modified fields and/or unchanged fields to be displayed.

19.3.3 Generating and Printing a Report of your Comparison Results

To generate a report of your comparison results in Designer Navigator:

1. In the Version Comparison tool, click **Print**.

2. In the Report Generation dialog, set the **Filters on objects** and **Filters on fields** according to your needs.
3. In the **PDF file location** field, specify a file name to write the report to. If no path is specified, the file will be written to the default directory for PDF files. This is a user preference.
4. Check the box next to **Open file after generation** if you want to view the generated report in a PDF viewer.

Note: In order to view the generated report, you must specify the location of Adobe Acrobat Reader in the user parameters. You can also set the default PDF generation directory. To set these values, select the **Preferences...** option from the **Tools** menu. Expand the **ODI** node, and then the **System** node, and select **Reports**. Enter (or search for) the location of your preferred **PDF Viewer**, and of your **Default PDF generation directory**.

5. Click **Generate**.

A report in Adobe PDF format is written to the file specified in step 3.

19.4 Working with Labels

A **label** is a comprehensive and consistent set of interdependent versions of objects. Like other objects, it can be checked in at a given time as a version, and may be restored at a later date. Labels are saved into the master repository. A label assembles a group of versions called the label's **elements**.

A label is automatically assembled using cross-references. By scanning cross-references, a label automatically includes all dependent objects required for a particular object. For example, when adding a project to a label, versions for all the models used in this project's interfaces are automatically checked in and added to the label. You can also manually add or remove elements into and from the label.

Labels are displayed in the Labels accordion in Designer Navigator and in Operator Navigator.

The following objects may be added into labels:

- Projects
- Models, Model Folders
- Scenarios
- Load Plans
- Global Variables, Knowledge Modules, User Functions and Sequences.

To create a label:

1. In Designer Navigator or Operator Navigator, from the Labels toolbar menu select **New Label**.
2. In the Labels editor, enter the **Name** of your label and a **Description**.
3. From the File menu select **Save**.

The resulting label is an empty shell into which elements may then be added.

19.4.1 Working with Elements in a Label

This section details the different actions that can be performed when working with elements of a label.

Adding Elements

To add an element, drag the object from the tree into the Elements list in the label editor. Oracle Data Integrator scans the cross-references and adds any Required Elements needed for this element to work correctly. If the objects being added have been inserted or updated since their last checked in version, you will be prompted to create new versions for these objects.

Removing Elements

To remove an element from a label, select the element you want to remove in the Elements list and then click the Delete button. This element disappears from the list. Existing checked in versions of the object are not affected.

Rolling Back Objects

To roll an object back to a version stored in the label, select the elements you want to restore and then click the Restore button. The elements selected are all restored from the label's versions.

19.4.2 Synchronizing Labels

Synchronizing a label automatically adds required elements that have not yet been included in the label, creates new versions of modified elements and automatically removes unnecessary elements. The synchronization process brings the content of the label up to date with the elements (projects, models, etc) stored in the repository.

To synchronize a label:

1. Open the label you want to synchronize.
2. Click **Synchronize** in the toolbar menu of the Elements section.
3. Oracle Data Integrator scans the cross-references. If the cross-reference indicates that the label is up to date, then a message appears. Otherwise, a list of elements to add or remove from the label is shown. These elements are grouped into Principal Elements (added manually), Required Elements (directly or indirectly referenced by the principal elements) and Unused Elements (no longer referenced by the principal elements).
4. Check the Accept boxes to version and include the required elements or delete the unused ones.
5. Click **OK** to synchronize the label. Version creation windows may appear for elements requiring a new version to be created.

You should synchronize your labels regularly to keep the label contents up-to-date. You should also do it before checking in a label version.

19.4.3 Restoring and Checking in a Label

The procedure for checking in and restoring a label version is similar to the method used for single elements. See "[Working with Versions](#)" on page 19-2 for more details.

You can also restore a label to import scenarios into production in Operator Navigator or Designer Navigator.

To restore a scenario from a label:

1. Double-click a label to open the Label editor.
2. Select a scenario from the **Principal** or **Required Elements** section. Note that other elements, such as projects and mappings, cannot be restored.
3. Click **Restore** in the toolbar menu of the Elements section.

The scenario is now accessible in the Scenarios tab.

Note that you can also use the Version Browser to restore scenarios. See "[Restoring a version with the Version Browser](#)" on page 19-4.

Note: When restoring a label, elements in the label are not automatically restored. They must be restored manually from the Label editor.

19.4.4 Importing and Exporting Labels

Labels can be exported and imported similarly to other objects in Oracle Data Integrator. Export/Import is used to transfer labels from one master repository to another. Refer to [Chapter 23, "Exporting and Importing,"](#) for more information.

Integrating ODI with Version Control Systems

This chapter describes how to integrate Oracle Data Integrator (ODI) with external Version Control Systems (VCS). It provides information on how to set up VCS with ODI and step-by-step instructions to use the VCS features in ODI.

This chapter includes the following sections:

- [Introduction to ODI-VCS integration](#)
- [Selecting the VCS to use with ODI](#)
- [Creating an SVN Connection](#)
- [Editing an SVN Connection](#)
- [Configuring Subversion Settings](#)
- [Configuring Subversion Repository with ODI](#)
- [Creating a Default Subversion Project Structure](#)
- [Populating a New ODI Repository from a Subversion Branch/Trunk](#)
- [Populating a Restored ODI Repository from a Subversion Branch/Trunk](#)
- [Understanding Generic Profiles in ODI](#)
- [Creating a Full Tag in the Subversion Repository](#)
- [Creating a Partial Tag in the Subversion Repository](#)
- [Creating a Branch from a Tag](#)
- [Unlocking the ODI Repository](#)
- [Adding Non-versioned ODI Objects to the Subversion Repository](#)
- [Adding a Single Non-versioned ODI Object to the Subversion Repository](#)
- [Creating versions of a version controlled ODI Object](#)
- [Restoring a Version Controlled ODI Object from its Previous Version](#)
- [Restoring a Version Controlled ODI Object Deleted in ODI Repository](#)
- [Viewing the Version History of a Version Controlled ODI Object](#)
- [Comparing Versions of an ODI Object from the Version History Dialog](#)
- [Viewing Version Tree of a Version Controlled ODI Object](#)
- [Comparing Versions of an ODI Object from the Version Tree Editor](#)
- [Performing a Merge](#)

20.1 Introduction to ODI-VCS integration

ODI integrates with external Version Control Systems (VCS) to enable version control of ODI objects. You can store versioned copies of the ODI objects into an external VCS repository. Since, VCS relies on file-based storage, ODI objects are stored as XML files in the VCS repository.

Note: Currently ODI supports only Apache™ Subversion®.

To version control ODI objects using VCS, you need to configure a connection between the ODI repository and the VCS repository. Once the connection is configured, you can add unversioned ODI objects to VCS, retrieve older versions of ODI objects from VCS, and view the differences between two versions of ODI objects from within ODI.

You can add first class objects, for example, packages, mappings, variables, security objects, and topology objects, and container objects, for example, projects, folders, and model folders to VCS. When you add container objects to VCS, all its descendents are re-versioned if they are newer than the version in the VCS.

Following are some of the important advanced ODI VCS integration features:

- Create full or partial tags for the consistent set of ODI objects in VCS, which can be used later for branching and deploying.
- Populate a new or a restored ODI repository from a branch/trunk in the VCS.
- Auto-version version controlled ODI objects as they are saved.
- Encrypt (or decrypt) confidential information when exporting or importing ODI objects from or to the VCS repository.
- AES Encryption support for cipher content in the ODI objects to be stored in the VCS repository.

20.2 Selecting the VCS to use with ODI

Perform the following steps to select the VCS that you want ODI to use as the team versioning application.

To select a VCS to use with ODI:

1. Click **Team >Select Versioning Application**.
2. On the Select Version Control Management System dialog, select **Subversion** and then click **OK**.

Note: Currently ODI supports only Apache™ Subversion®.

3. On the Confirmation dialog, click **Yes**. The Disconnect confirmation dialog appears.
4. On the Disconnect confirmation dialog, click **Disconnect** to disable the Legacy Version Control System and to disconnect from the ODI repository.
5. Reconnect to the ODI repository. ODI will now use Subversion as the team versioning application.

20.3 Creating an SVN Connection

You need to connect to the Subversion repository from ODI Studio. ODI provides the following types of authentication options when connecting to the Subversion repository:

To create an Subversion connection from the ODI Studio:

1. Click **Team > Subversion > Edit Connection**.
2. From the **Authentication Type** drop-down menu, select one of the following authentication types:
 - HTTP Basic Authentication
 - Subversion Basic Authentication
 - SSH Authentication
 - SSL Authentication
 - File Based Authentication

Based on the selected authentication type, different options are displayed in the SVN Connection dialog.

3. Set the required options and click **OK**.

For more information, see the following sections:

["HTTP Basic Authentication Options"](#)

["Subversion Basic Authentication Options"](#)

["SSH Authentication Options"](#)

["SSL Authentication Options"](#)

["File Based Authentication Options"](#)

See ["Guidelines for Choosing the Authentication Type"](#).

20.3.1 HTTP Basic Authentication Options

The following tables describes the options specific to HTTP Basic Authentication.

Table 20–1 *HTTP Basic Authentication Options*

Option	Description
SVN URL	Subversion repository URL.
User	User name for authentication.
Password	Password to connect to the Subversion repository.
Use Proxy Server	Select to use an HTTP or HTTPS proxy server. Note: If this check box is selected, the Host and Port properties are enabled.
Host	Proxy server to connect to the Subversion repository.
Port	Proxy server port.
Proxy Server Requires Authentication	Select to authenticate the proxy server. Note: If this check box is selected, the User and Password properties are enabled.
User	User name to connect to the proxy server.

Table 20–1 (Cont.) HTTP Basic Authentication Options

Option	Description
Password	Password to connect to the proxy server.

20.3.2 Subversion Basic Authentication Options

The following table describes the options specific to Subversion Basic Authentication.

Table 20–2 Subversion Basic Authentication

Option	Description
SVN URL	Subversion repository URL.
User	User name for authentication.
Password	Password to connect to the Subversion repository.

20.3.3 SSH Authentication Options

The following table describes the options specific to SSH Authentication.

Table 20–3 SSH Authentication Options

Options	Descriptions
Subversion URL	Subversion repository URL.
User	User name for authentication.
Password	Password to connect to the Subversion repository.
Private Key File	Select to establish an SSH connection using SSH keys. Note: If this check box is selected, the Key File and Passphrase properties are enabled.
Key File	Path of the Private Key File.
Passphrase	Passphrase for the selected Private Key File.

20.3.4 SSL Authentication Options

The following table describes the options specific to SSL Authentication.

Table 20–4 SSL Authentication Options

Options	Description
SVN URL	Subversion repository URL.
User	User name for authentication.
Password	Password to connect to the Subversion repository.
Enable Client Authentication	Select to enable client authentication. Note: If this check box is selected, the Certificate File and Passphrase properties are enabled.
Certificate File	Path of the Certificate File.
Passphrase	Passphrase for the selected Certificate File.

20.3.5 File Based Authentication Options

The following table describes the options specific to File Based Authentication.

Table 20–5 File Based Authentication

Options	Description
SVN URL	Subversion repository URL.

20.4 Editing an SVN Connection

Perform the following steps to edit an existing Subversion connection.

To edit an Subversion connection:

1. Click **Team> Subversion> Edit Connection**.
2. Edit the options as required.

For more information, see the following sections:

["HTTP Basic Authentication Options"](#)

["Subversion Basic Authentication Options"](#)

["SSH Authentication Options"](#)

["SSL Authentication Options"](#)

["File Based Authentication Options"](#)

3. Click **OK**.

See ["Guidelines for Choosing the Authentication Type"](#).

20.5 Configuring Subversion Settings

You need to configure the Subversion specific settings in ODI. These settings include the working folder path and the merge working folder.

To configure the Subversion settings:

1. Click **Tools> Preferences> ODI> User Interface> Versioning> Subversion**.
2. Specify the Subversion Settings.

For more information, see ["Subversion Settings"](#).

3. Click **OK**.

20.5.1 Subversion Settings

The following table describes the options that you need to set on the Subversion Settings dialog.

Table 20–6 Subversion Settings Options

Options	Description
Working Folder Path	The working folder contains artifacts exported from the Subversion repository to execute various version management operations like export, commit, restore, and so on. ODI replicates the same folder structure present in the Subversion repository while storing these artifacts in the working folder.
Merge Working Folder	This folder contains the ODI artifacts exported from the Subversion repository during the Merge operation. Note: ODI deletes the artifacts present in the merge working folder once the merge operation is complete.

20.6 Configuring Subversion Repository with ODI

Perform the following steps to configure the Subversion repository with ODI.

To configure Subversion repository with ODI:

1. Click **Team> Subversion> Configure**.
2. Set the configuration options.
For more information, see ["Options to Configure Subversion Repository with ODI"](#).
3. Click **OK**.

Note: After configuring the Subversion repository with ODI, you must disconnect from the ODI repository and connect again.

20.6.1 Options to Configure Subversion Repository with ODI

The following table describes the options that are required to configure the Subversion Repository with ODI.

Table 20–7 Options to configure Subversion repository with ODI

Options	Description
SVN Repository Name	Name of the existing Subversion repository.
SVN Project Name	List of the Subversion projects available in the Subversion repository.
Create Default Project Structure	Click to create a default project structure in the Subversion repository. For more information, see Section 20.7, "Creating a Default Subversion Project Structure" .
Select Trunk or Branch	Select a trunk or an available branch.
New Branch	Click to create a new branch from a tag.
Auto Version	Select to enable auto versioning at the ODI repository level. Note: If Auto Version is enabled, a new version is created for an already versioned object whenever changes are made to the object and the changes are saved.
VCS Key	Select to create a VCS Key. The VCS key can be a minimum of 8 characters and a maximum of 100 characters long.
Enter VCS Key	Click to enter the VCS Key.

20.7 Creating a Default Subversion Project Structure

You need to create a default project structure in the Subversion repository. This structure helps to identify the trunk, branch, and tag folders present in the Subversion project created in the Subversion repository.

To create a default Subversion project structure:

1. Click **Team> Subversion> Configure**.
2. Click **Create Default Project Structure** button.
3. Enter a name for the Subversion Project in the **SVN Project Name** field.
4. Enter a description in the **Comments** field.

5. Click OK.

20.8 Populating a New ODI Repository from a Subversion Branch/Trunk

You can populate a new ODI Repository with the ODI objects from a trunk/branch that exists in the Subversion repository.

To populate a new ODI repository from a Subversion trunk/branch:

1. Click **Team> Subversion> Populate Repository from Branch/Trunk**.
2. Click **Yes** on the Confirmation dialog.
3. View the Import Report and close the report.
4. Verify that the version controlled ODI objects are populated in the ODI Studio.

20.9 Populating a Restored ODI Repository from a Subversion Branch/Trunk

You can restore a new ODI Repository from the database backup and then populate the restored ODI Repository from a trunk/branch that exists in the Subversion Repository.

Note: Only ODI VCS Administrators can populate a restored ODI Repository from a Subversion trunk/branch.

To populate a restored ODI repository from a Subversion branch/trunk:

1. Create a new ODI repository using ODI Studio or Repository Creation Utility (RCU).
2. Restore the new ODI repository from the database backup.
3. Configure the restored ODI Repository with the last configured Subversion trunk/branch.

For more information, see "[Configuring Subversion Repository with ODI](#)".

4. Click **Team> Subversion> Populate Restored Repository from Branch/Trunk**.
5. Click **Yes** on the Confirmation dialog.
6. View the Import Report and click **Close**.
7. Verify that the version controlled ODI objects are populated in the ODI Studio.

20.10 Understanding Generic Profiles in ODI

ODI provides out-of-the-box generic profiles. These profiles contain a set of privileges to work with version management and version administration operations.

ODI includes the following generic profiles:

- **Designer**

Contains a set of privileges to work with version management operations such as add a non version controlled ODI Object to the VCS, create a new version of a version controlled ODI Object, restore a version controlled ODI Object from one of its previous versions, etc. on all the design objects provided by ODI.

- **Security Admin**

Contains a set of privileges to work with version management operations such as add a non version controlled ODI Object to the VCS, create a new version of a version controlled ODI Object, restore a version controlled ODI Object from one of its previous versions, etc. on all the security objects provided by ODI.

- **Topology Admin**

Contains a set of privileges to work with version management operations such as add a non version controlled ODI Object to the VCS, create a new version of a version controlled ODI Object, restore a version controlled ODI Object from one of its previous versions, etc. on all the topology objects provided by ODI.

- **VCS Version Admin**

Contains a set of privileges to work with version administration operations such as select the version control system to be integrated with ODI, configure ODI Repository with the Version Control System, create a label, merge development branch with trunk, etc. along with a set of privileges given to DESIGNER, SECURITY ADMIN, and TOPOLOGY ADMIN profile.

20.11 Creating a Full Tag in the Subversion Repository

A tag is identification text that you can assign to identify a set of consistent objects versions, or the entire repository in Subversion.

You can create a full tag from all the objects present in the branch or the trunk in the ODI repository.

This will enable you to create a consistent set of ODI artifacts in the Subversion Repository, which can be shared with other users who can create a new repository from a full tag.

To create a full tag in the Subversion repository:

1. Click **Team> Subversion> Create Full Tag**.
2. Specify the options on the Create Full Tag dialog.
3. In the **Tag** field, enter a name for the tag.
4. Verify the name of the Subversion branch or trunk.
Note: This is a read-only field.
5. In the **Comments** field, enter a description for the tag.
6. Click **OK** to create the tag.

See "[General Tagging Guidelines](#)".

20.12 Creating a Partial Tag in the Subversion Repository

You can create a partial tag from the subset of ODI artifacts present in the trunk or branch of the Subversion repository.

This will enable you to create a consistent set of ODI artifacts in the Subversion Repository, which can be shared with other users who can create a new repository from a partial tag.

To create a partial tag in the Subversion repository:

1. Click **Team> Subversion> Create Partial Tag**.

2. Specify the options on the Create Partial Tag dialog.
For more information, see ["Create Partial Tag Options"](#).
3. Click **OK** to create the tag.
See ["General Tagging Guidelines"](#).

20.12.1 Create Partial Tag Options

The following table describes the options that you need to specify on the Create Partial Tag dialog.

Table 20–8 Create Partial Tag Options

Options	Description
Tag	Name of the tag.
Branch/Trunk	Name of the branch or trunk to which ODI is configured.
Include all security objects	Select to include all security objects.
Comments	Log message, which describes the changes you are committing.
Selected Objects	Objects to be added. To add objects, drag-and-drop the supported objects from the navigator tree or search for an object using the Search option in the toolbar and add it.
Dependent Objects	Objects dependent on the selected object.

20.13 Creating a Branch from a Tag

You can create a new Subversion branch from a tag that exists in the Subversion repository.

To create a branch from a tag:

1. Click **Team > Subversion > Create Branch from Tag**.
2. Specify the options on the Create Branch from Tag dialog.
For more information, see ["Create Branch from Tag Options"](#).
3. Click **OK**. A success Information dialog appears.
4. Click **OK**.

See ["General Branching Guidelines"](#).

See ["Branching Guidelines for Single Development Team"](#).

See ["Branching Guidelines for Parallel Development Teams"](#).

See ["Guidelines for Release Branches for Parallel Development Teams"](#).

20.13.1 Create Branch from Tag Options

You need to specify the following options on the Create Branch from Tag dialog box.

Table 20–9 Create Branch from Tag Options

Option	Description
Branch Name	Name of the branch.
Tag	List of tags for the ETL project configured in the ODI repository.

Table 20–9 (Cont.) Create Branch from Tag Options

Option	Description
Comments	Log message, which describes the changes you are committing.

20.14 Unlocking the ODI Repository

The ODI Repository may get locked during the VCS Tag creation. You cannot make any changes to the ODI Repository in a locked state. Sometimes due to unexpected errors while Tag creation, the ODI Repository may remain in the locked state.

In such situations, you can run the `OdiUnlockOdiRepository` tool to unlock the ODI Repository. For more information about running a tool from a command line, see section *Using a Tool From a Command Line*.

For more information about the `OdiUnlockOdiRepository` tool, see section *OdiUnlockODIRepository*.

Note: You must have VCS Administrator privileges to run the `OdiUnlockOdiRepository` command.

20.15 Adding Non-versioned ODI Objects to the Subversion Repository

You can add all or multiple non-versioned ODI objects to the Subversion repository.

To add non-versioned ODI objects to the Subversion repository:

1. Click **Team > Subversion > Add Non-versioned Objects**.
A list of all the non-versioned ODI objects is displayed.
2. Select the ODI Objects that you want to add to the Subversion repository.
Select multiple ODI objects or all ODI objects listed on the Add Non-Versioned Objects dialog.
3. In the **Comments** field, enter a description for the ODI objects that you are adding to the Subversion repository.
4. Click **OK**. A success Information dialog appears.
5. Click **OK**.

See "[Guidelines for Versioning During Development](#)".

20.16 Adding a Single Non-versioned ODI Object to the Subversion Repository

You can add a single non-versioned ODI object to the Subversion repository.

To add a single non-versioned ODI object to the Subversion repository:

1. Right-click the ODI object (example, folder, model, datastore, packages, etc.) that you want to add to the Subversion repository.
2. Select **Version > Subversion > Add to VCS**. The Add ODI Objects to Subversion dialog appears.

Note: The Add ODI Objects to Subversion dialog lists the parent ODI objects that need to be added along with the selected ODI object.

3. In the **Comments** field, enter a description for the ODI objects that you are adding to the Subversion repository.
4. Click **OK**. A success Information dialog appears.
5. Click **OK**.

See ["Guidelines for Versioning During Development"](#).

20.17 Creating versions of a version controlled ODI Object

Whenever you modify a version controlled ODI object, you can create a new version of it. You can create versions of both, the first class objects and the parent container objects. When you create version of a parent container object, all the child objects are also versioned.

To create versions of a version controlled ODI object:

1. Right-click the ODI object of which you want to create a version.
2. Select **Version> Subversion> Create VCS Version**.
3. Verify the information on the Create Version dialog.

When creating a version of a version controlled parent container object, information regarding the child objects are also displayed in the dialog. This information includes the Name, Type, Path, and if the child object was added, modified, or deleted.

4. In the **Comments** field, enter a description of the version that you are creating.
5. Click **OK**. A success Information dialog appears.
6. Click **OK**.

See ["Guidelines for Versioning During Development"](#).

20.18 Restoring a Version Controlled ODI Object from its Previous Version

You can restore a version controlled ODI object from its previous version or revision in the Subversion repository.

To restore a version controlled ODI object from its previous version:

1. Right-click the ODI object that you want to restore from the Subversion repository.
2. Select **Version> Subversion> Restore from VCS**.
3. Specify the options on the Restore Object from Subversion Repository dialog.
For more information, see ["Restore Object from Subversion Options"](#).
4. Click **OK**.
5. If you choose to restore with merge, you need to perform additional steps that are mentioned in ["Performing a Merge"](#).

20.18.1 Restore Object from Subversion Options

The following table describes the options that you need to specify on the Restore Object from Subversion dialog.


Table 20–10 Restore Object from Subversion Options

Option	Description
Name	Name of the ODI object. This is a read-only field.
Type	Type of ODI object to be restored from the Subversion repository. This is a read-only field.
Path	Path of the ODI object present in the ODI repository. This is a read-only field.
Select Version	Click to access the Version Selection dialog and select a particular version of the ODI object present in the Subversion repository.
Restore with Merge	Select to restore an object using the Merge option. For more details about performing a merge, see " Performing a Merge ".
Restore child objects with Merge	Select to restore an object along with its child objects using the Merge option.

20.19 Restoring a Version Controlled ODI Object Deleted in ODI Repository

If you accidentally delete a version controlled object in the ODI repository, you can restore it from the Subversion repository.

To restore a version controlled ODI object deleted in the ODI repository:

1. Click **Team > Subversion > Restore Deleted Object**.
The Restore Deleted ODI Object dialog appears.
2. Under **Version Search Criteria**, specify the criteria to search versions of the object in the Subversion repository.
For more information, see "[Version Search Criteria](#)".
3. Click **Apply**. All the versions of the object that are available in the Subversion repository are listed under **Versions**.
4. Review the details of each of the versions of the ODI object that are listed.
5. Select the version that you want to restore and click the **Restore Object** icon ()
6. See the Import Report to confirm if the object has been imported successfully and close the report.
7. Click **Close** on the Restore Deleted ODI Object dialog.

20.20 Viewing the Version History of a Version Controlled ODI Object

You can view the version history of a version controlled ODI object across trunk, multiple branches, and tags of the Subversion project in the Subversion repository.

To view the version history of an ODI object:

1. Right-click the ODI object whose version history you want to view.
2. Select **Version > Subversion > Version History**.

3. Under **Version Search Criteria**, specify the criteria to search the versions of the selected ODI object.

For more information, see "[Version Search Criteria](#)".

4. Click **Apply**.
5. Under **Versions**, all the versions of the selected ODI object that exist in the Subversion repository are listed.
6. Review the version history details for each of the versions of the ODI object.
7. Click **Close**.

You can compare versions of the ODI object using the steps mentioned in "[Comparing Versions of an ODI Object from the Version History Dialog](#)".

20.20.1 Version Search Criteria

The following table describes the options that you need to specify on the Version Search Criteria dialog.

Table 20–11 *Version Search Criteria Options*



Option	Description
Branch/Tag/Trunk	Select Branch, Tag, or Trunk depending on what the ODI repository is mapped to.
Branch/Tag	Select the appropriate Branch or Tag from the list. Note: This list is populated only if the Branch or Tag check box is selected.
Date Range	Select to enter the range of dates within which the version was created.
Version Range	Enter the first and last version to view all the versions within this range. Note: If the Version Range fields are left empty, all versions of the object are displayed. If the lower limit of the version range is not specified, all the versions that are lesser than the specified upper limit are displayed. Similarly, if the upper limit of the version range is not specified, all the versions that are greater than the specified lower limit are displayed.
Path	Path of the ODI object in the ODI repository. Note: You can use * as a wildcard character.
Comments	Enter the log message, which describes the changes that were committed in the selected version of the object. Note: You can use * as a wildcard character.

20.21 Comparing Versions of an ODI Object from the Version History Dialog

You can compare versions of an ODI object from the Version History dialog to see the difference between them. You can either compare two versions of an object in the Subversion repository, or compare one version of the object in the Subversion repository with the current object in the ODI repository.

To compare versions of an ODI object:

1. Right-click the ODI object whose versions you want to compare.
2. Select **Version> Subversion> Version History**.

3. Under **Version Search Criteria**, specify the criteria to search the versions of the selected ODI object.
For more information, see "[Version Search Criteria](#)".
4. Click **Apply**. All the versions of the selected ODI object that exist in the Subversion repository are listed under **Versions**.
5. Do one of the following:
 - To compare two versions of the object in the Subversion repository:
Select the two versions that you want to compare, click the **Compare** icon () , and then select **Compare Selected Versions**.
 - To compare a version of the object in the Subversion repository with the current object in the ODI repository:
Select the version that you want to compare with the current object in the ODI repository, click the **Compare** icon () , and then select **Compare With Repository Object**.
6. On the Version Compare Results dialog, view the differences between the two versions.

The Version Compare Results dialog provides icons to navigate through the differences, filter the compare results, change the display options (color coding), generate report, expand or collapse the nodes in the tree, and perform merge.

For more information, see "[Icons on the Version Compare Results dialog](#)".
7. (Optional) Perform a merge using the steps mentioned in "[Performing a Merge](#)".
8. Click **Close**.

20.21.1 Icons on the Version Compare Results dialog

The following table describes the icons that are available on the Version Compare Results dialog.

Table 20–12 *Icons on the Version Compare Results dialog*












Icon	Name	Description
	Refresh	Refreshes the results.
	Go to First Difference	Jumps to the first difference.
	Go to Previous Difference	Jumps to the previous difference.
	Go to Next Difference	Jumps to the next difference.
	Go to Last Difference	Jumps to the last difference.
	Coloring and Filtering	Displays the Display Options dialog, where colors can be assigned to fields and objects.
	Reset all Filters	Resets all filters to the default.
	Generate Report	Generates the report.

Table 20–12 (Cont.) Icons on the Version Compare Results dialog

Icon	Name	Description
	Expand All	Expands all the nodes in the tree.
	Collapse All	Collapses all the nodes in the tree.
	Perform Merge	Merges the changes. For more information, see "Performing a Merge" and "Performing a Branch Merge" .

20.22 Viewing Version Tree of a Version Controlled ODI Object

You can view the version history of a version controlled ODI objects across multiple branches, tags, or trunk present in the Subversion repository in the form a version tree graph.

Each revision node in the version tree graph represents a revision in the Subversion repository. The nodes are distinguished by different colors. Added items are in yellow color, deleted items in red color, modified items in white color, merged items in orange color, and restored items in green color.

To view version tree of a version controlled ODI object:

1. Right-click the ODI object and select **Version> Subversion> Version Tree**.

The Version Tree Editor appears showing different versions, tags, and branches for the object in the Subversion repository.

2. Scroll horizontally to view the entire version tree.

Note: Version nodes are displayed as ovals enclosed in rectangles and the action nodes are displayed as ovals.

3. Click the version nodes to view the version attributes under **Version Properties**.
4. Close the Version Tree Editor.

You can compare versions of the ODI object using the steps mentioned in ["Comparing Versions of an ODI Object from the Version Tree Editor"](#).


20.23 Comparing Versions of an ODI Object from the Version Tree Editor

You can compare versions of an ODI object from the Version Tree Editor to see the difference between them. You can either compare two versions of an object in the Subversion repository, or compare one version of the object in the Subversion repository with the current object in the ODI repository.


To compare versions of an ODI object:

1. Right-click the ODI object whose versions you want to compare.
2. Select **Version> Subversion> Version Tree**.
3. Do one of the following:

- To compare two versions of the object in the Subversion repository:

Select two version nodes that you want to compare, click the **Compare** icon () , and then select **Compare Selected Versions**.

- To compare a version of the object in the Subversion repository with the current object in the ODI repository:

Select the version node that you want to compare with the current object in the ODI repository, click the **Compare** icon (), and then select **Compare With Repository Object**.


4. On the Version Compare Results dialog, view the differences between the two versions.

The Version Compare Results dialog provides icons to navigate through the differences, filter the compare results, change the display options (color coding), generate report, expand or collapse the nodes in the tree, and perform merge.

For more information, see ["Icons on the Version Compare Results dialog"](#).

5. (Optional) Perform a merge using the steps mentioned in ["Performing a Merge"](#).
6. Click **Close**.

20.24 Performing a Merge

When you compare two versions of an ODI object, the differences between them are shown on the Version Compare Results dialog. You can use the **Perform Merge** icon () on the Version Compare Results dialog to perform a merge.

To perform a merge:


1. Compare versions of an ODI object.

For more information, see ["Comparing Versions of an ODI Object from the Version History Dialog"](#) or ["Comparing Versions of an ODI Object from the Version Tree Editor"](#).


2. On the Version Compare Results dialog, click the **Perform Merge** icon ()

The Merge Results dialog appears with a list of Merge Objects.


3. On the Merge Object Selection tab, select the filters as appropriate. For example, you may filter the results to see only the conflicting objects.
4. Perform the following steps to resolve any conflicts:

- a. On the Merge Object Selection tab, select the conflicting object and click the **Fix Merge Conflict** icon ()

The Merge Conflict Resolution tab appears showing the differences between the two versions of the object.

- b. Inspect the differences and click the **Edit Repository Object** icon ()

The ODI object opens in an editor.

- c. Modify the ODI object to resolve the conflict and save the changes.
- d. Click the **Conflict Resolved** icon () to mark the object as resolved.
- e. Perform steps a. to d. to resolve all the conflicting objects.

5. Close the Merge Results dialog.

20.25 Performing a Branch Merge

You can perform a branch merge to merge the changes done in the branch to the current ODI repository.

To perform a branch merge:

1. Click **Team > Subversion > Merge**.
2. Select **Branch** as the **Merge Type** as **Branch**.
3. Select a source from the **Source** drop-down list.
4. Specify a name for the merge in the **Merge Name** field.

You can choose to go ahead with the default merge name.

5. Click **OK**. The merge is performed and the Merge Summary is displayed.
6. Inspect the conflicts in the objects that are modified in the branch.
7. Close the Merge Summary. The Merge Results dialog appears with the **Merge Objects** list.

You may filter the **Merge Objects** list to show only the conflicting objects.

8. Perform the following steps to resolve the conflicts:
 - a. On the Merge Object Selection tab, select the conflicting object and click the **Fix Merge Conflict** icon (🔗).
The Merge Conflict Resolution tab appears showing the differences between the two versions of the object.
 - b. Inspect the differences and click the **Edit Repository Object** icon (✏️).
The ODI object opens in an editor.
 - c. Modify the ODI object to resolve the conflict and save the changes.
 - d. Click the **Conflict Resolved** icon (✅) to mark the object as resolved.
 - e. Perform steps a. to d. to resolve all the conflicting objects.
9. Close the Merge Results tab.

20.25.1 Viewing Merge Summary

You can see the Merge Summary report for the previous merges that you have performed.

Tip: The Merge Summary report can also be accessed from the Merge Results dialog, which is displayed when you perform a merge.

To view the merge summary:

1. Click **Team > Subversion > Merge Summary**.
2. Select a merge status to view merge summary for all, completed, or in progress merges.
3. Select a merge name to display the associated merge summary.
4. Click **OK**.

Release Management

This chapter describes the features added to ODI to support release management.

This chapter includes the following sections:

- [Managing ODI Releases](#)
- [Types of Deployment Archives](#)
- [Creating a Deployment Archive from a VCS Label](#)
- [Creating an Initial Deployment Archive from the ODI Repository](#)
- [Creating a Patch Deployment Archive from the ODI Repository](#)
- [Viewing Available Deployment Archives](#)
- [Initializing an ODI Repository Using an Initial Deployment Archive](#)
- [Updating an ODI Repository Using a Patch Deployment Archive](#)
- [Viewing Deployment Archives Applied in an ODI Repository](#)
- [Rolling Back a Patch Deployment Archive](#)

21.1 Managing ODI Releases

You can manage ODI releases using deployment archives. A deployment archive is an archived file (zip file) that contains a set of ODI objects in the form of XML files and metadata. You can create deployment archives that can be used to either initialize an ODI repository or to update a deployed ODI repository.

If ODI is integrated with a VCS, deployment archives can be created from the VCS labels. If ODI is not integrated with a VCS, deployment archives can be created from the current ODI repository.

See also, "[Types of Deployment Archives](#)".

21.2 Types of Deployment Archives

You can create the following types of deployment archives in ODI:

- **Initial Deployment Archives**

Initial deployment archives contain all the ODI objects that are necessary to initialize an ODI repository. You can create an initial deployment archive and use it to deploy an ODI repository in an environment where the ODI objects are not modified, for example, in a testing or a production environment.
- **Patch Deployment Archives**

Patch deployment archives contain only the ODI objects that need to be updated in an ODI repository. You can create a patch deployment archive and use it to update an ODI repository that is already deployed. For example, when you update any ODI objects in a development environment, the updates can be applied in a testing or a production environment using a patch deployment archive.

See also:

["Creating a Deployment Archive from a VCS Label"](#).

["Creating an Initial Deployment Archive from the ODI Repository"](#).

["Creating a Patch Deployment Archive from the ODI Repository"](#).

21.3 Creating a Deployment Archive from a VCS Label

In an ODI environment that is integrated with VCS, you can create a deployment archive from a VCS label. The deployment archive will be created with the ODI objects that are included in the VCS label.

To create a deployment archive from a VCS label:

1. Click **Team > Deployment Archives > Create From VCS Label**.
2. Select the type of deployment archive that you want to create. You can choose one of the following types:
 - **Initial Deployment Archive**
Contains all the objects that are necessary to initialize an ODI repository.
 - **Patch Deployment Archive**
Contains only the ODI objects that need to be updated in an ODI repository.For more information, see ["Types of Deployment Archives"](#).
3. In the **Deployment Archive Name** field, enter a name for the deployment archive.
4. In the **Deployment Archive Folder Path** field, specify a folder path where you want to store the deployment archive.
5. From the **Select VCS Label** drop down list, select the VCS label that you want to use to create the deployment archive.
6. In the **Description** field, enter a description for the deployment archive.
7. In the **Export Key** field, set a export key.
8. In the **Confirm Export Key** field, re-enter the export key for confirmation.
9. Click **OK**.
A success Information dialog appears.
10. Click **OK**.

See ["Guidelines for Initial Deployment and Patching"](#).

21.4 Creating an Initial Deployment Archive from the ODI Repository

In an ODI environment that is not integrated with VCS, you can create an initial deployment archive from the current state of the ODI repository.

This section describes how to create an initial deployment archive, which will include all the ODI objects that are required to initialize an ODI repository.

For more information, see "[Types of Deployment Archives](#)".

To create an initial deployment archive from the ODI repository:

1. Click **Team > Deployment Archives > Create From Repository > Full Repository**.
2. In the **Deployment Archive Name** field, enter a name for the deployment archive.
3. In the **Deployment Archive Folder Path** field, specify a folder path where you want to store the deployment archive.
4. In the **Description** field, enter a description for the deployment archive.
5. In the **Export Key** field, set a export key.
6. In the **Confirm Export Key** field, re-enter the export key for confirmation.
7. Click **OK**.

A success Information dialog appears.

8. Click **OK**.

See "[Guidelines for Initial Deployment and Patching](#)".

21.5 Creating a Patch Deployment Archive from the ODI Repository

In an ODI environment that is not integrated with VCS, you can create a patch deployment archive with selected ODI objects from the current ODI repository.

Note: When you create a patch deployment archive from the ODI Repository, an execution deployment archive is also created. The execution deployment archive includes only the **Scenarios** and **Load Plans** corresponding to the ODI objects that are included in the patch deployment archive.

The execution deployment archive names are prefixed with EXEC_.

This section describes how to create a patch deployment archive, which will include only the ODI objects that need to be updated in an ODI repository.

For more information, see "[Types of Deployment Archives](#)".

To create a patch deployment archive from the ODI repository:

1. Click **Team > Deployment Archives > Create From Repository > Selected Objects**.
2. Select **Patch Deployment Archive** option.

Note: It is necessary to select the **Patch Deployment Archive** option.
3. In the **Deployment Archive Name** field, enter a name for the deployment archive.
4. In the **Deployment Archive Folder Path** field, specify a folder path where you want to store the deployment archive.
5. In the **Description** field, enter a description for the deployment archive.
6. Select **Include all security objects** option if you want to include all the security objects in the deployment archive.
7. In the **Export Key** field, set a export key.

8. In the **Confirm Export Key** field, re-enter the export key for confirmation.
9. Drag and drop the required ODI objects from the Designer Navigator into the **Selected Objects** field under **Objects to be added to Deployment Archive**.
The dependent objects that will be included in the deployment archive are listed under **Dependent Objects**.
10. Click **OK**.
A success Information dialog appears.
11. Click **OK**.
See "[Guidelines for Initial Deployment and Patching](#)".

21.6 Viewing Available Deployment Archives

You can view a list of deployment archives that are available in a folder path along with their details. Double-clicking any of the deployment archives lists the contents of the deployment archive.

To view available deployment archives:

1. Click **Team > Deployment Archives > View Details**.
2. In the **Deployment Archive Folder Path** field, browse to the folder path where the deployment archives were created. All the deployment archives in the specified folder path are listed under **List of Deployment Archives**.
Note: You cannot type the folder path, you must use the **Browse** button to specify the folder path.
3. Double-click any of the deployment archives to see its contents. A list of ODI objects contained in the deployment archive is displayed.
4. Click **Close**.

See also, "[Initializing an ODI Repository Using an Initial Deployment Archive](#)"

21.7 Initializing an ODI Repository Using an Initial Deployment Archive

You can initialize an ODI repository that is not connected to VCS with the contents of an initial deployment archive. If VCS connectivity is configured then it is assumed to be a development repository and thus the options to apply deployment archive are disabled.

Before you initialize an ODI repository, ensure that the Work repository is empty.

Note: In situations where the ODI repository is initialized with deployment archives from multiple Work repositories, the Master repository will not be empty. In such situations, when you load the deployment archive from the second Work repository, the contents of the Master repository are overwritten.

To initialize an ODI repository using an initial deployment archive:

1. Click **Team > Deployment Archives > Initialize Deployment Archive**.

2. In the **Deployment Archive Folder Path** field, browse to the folder path where the deployment archive was created. All the deployment archives in the specified folder path are listed under **List of Deployment Archives**.

Note: You cannot type the folder path, you must use the **Browse** button to specify the folder path.

3. Select the initial deployment archive that you want to use to initialize the ODI repository.
4. Click **OK**.

If the selected deployment archive contains cipher data, you are prompted to enter the export key.

5. Do one of the following:
 - Select **Enter Export Key** option and enter the export key that was used when creating the deployment archive.
 - Select **Import file without cipher data** option if you want to import the file without the cipher data.

6. Click **OK**.

A success Information dialog appears.

7. Click **OK**.

See also, "[Updating an ODI Repository Using a Patch Deployment Archive](#)".

See also, "[Guidelines for Deployment in Testing and Production Environments](#)".

21.8 Updating an ODI Repository Using a Patch Deployment Archive

You can update an ODI repository that is not connected to VCS with the updated ODI objects in a patch deployment archive. If VCS connectivity is configured then it is assumed to be a development repository and thus the options to apply deployment archive are disabled.

To update an ODI repository using a patch deployment archive:

1. Click **Team > Deployment Archives > Patch Deployment Archive**.
2. In the **Deployment Archive Folder Path** field, browse to the folder path where the deployment archive was created. All the deployment archives in the specified folder path are listed under **List of Deployment Archives**.

Note: You cannot type the folder path, you must use the **Browse** button to specify the folder path.

3. Select the patch deployment archive that you want to use to update the ODI repository.
4. Select **Create Rollback Deployment Archive** option if you want to create a rollback deployment archive.

Note: It is recommended to create a rollback deployment archive. If required, you can revert the changes applied by the patch deployment archive using this rollback deployment archive.

5. In the **Rollback Deployment Archive Path** field, browse to a folder path where you want to create the rollback deployment archive.

Note: The **Rollback Deployment Archive Path** field is enabled only if the **Create Rollback Deployment Archive** option is selected.

6. Click **OK**.

If the selected deployment archive contains cipher data, you are prompted to enter the export key.

7. Do one of the following:

- Select **Enter Export Key** option and enter the export key that was used when creating the deployment archive.
- Select **Import file without cipher data** option to import the file without the cipher data.

8. Click **OK**.

A success Information dialog appears.

9. Click **OK**.

See also, "[Viewing Deployment Archives Applied in an ODI Repository](#)".

See also, "[Guidelines for Deployment in Testing and Production Environments](#)".

21.9 Viewing Deployment Archives Applied in an ODI Repository

You can view the list of deployment archives applied in an ODI repository.

To view deployment archives applied in an ODI repository:

1. Click **Team > Deployment Archives > Inventory**.

The Inventory dialog appears with the list of deployment archives applied in the ODI repository.

2. View the details of the deployment archives listed on the Inventory dialog.
3. Click **Close**.

See also, "[Updating an ODI Repository Using a Patch Deployment Archive](#)".

21.10 Rolling Back a Patch Deployment Archive

If you want to revert the changes applied to an ODI repository using a patch deployment archive, you can rollback the patch deployment archive.

Multiple patch deployment archives can be rolled back, provided the rollback order is maintained. For example, if you had applied patch1 and then patch 2, you must first rollback patch 2 and then patch 1.

To rollback a patch deployment archive:

1. Click **Team > Deployment Archives > Rollback**.

2. In the **Deployment Archive Folder Path** field, browse to the folder path where the rollback archive was created.

Note: You cannot type the folder path, you must use the **Browse** button to specify the folder path.

3. From the **List of Deployment Archives**, select the appropriate rollback deployment archive.
4. Click **OK**.

If the selected deployment archive contains cipher data, you are prompted to enter the export key.

5. Do one of the following:
 - Select **Enter Export Key** option and enter the export key that was used when creating the deployment archive.
 - Select **Import file without cipher data** option to import the file without the cipher data.
6. Click **OK**.

A success Information dialog appears.
7. Click **OK**.

See also, "[Updating an ODI Repository Using a Patch Deployment Archive](#)".

Life Cycle Management Guidelines

This chapter provides the guidelines to be followed for using the Life Cycle Management features.

This chapter includes the following sections:

- [Guidelines for Choosing the Authentication Type](#)
- [General Branching Guidelines](#)
- [General Tagging Guidelines](#)
- [Branching Guidelines for Single Development Team](#)
- [Branching Guidelines for Parallel Development Teams](#)
- [Guidelines for Release Branches for Parallel Development Teams](#)
- [Guidelines for Versioning During Development](#)
- [Guidelines for Deployment in Testing and Production Environments](#)
- [Guidelines for Initial Deployment and Patching](#)

22.1 Guidelines for Choosing the Authentication Type

Please consider the following points before you choose the authentication type.

- **File Based Authentication:**

It provides direct access to repository through file system.

It should be avoided as it removes any layer of protection between users and the repository. Users may accidentally or intentionally corrupt the repository database.
- **HTTP Basic Authentication:**

Provides basic authentication through user name and password. Not secured.

Passwords are sent over the network in plain text. It should be avoided where security is concern.
- **SSL Authentication:**

Same as HTTP Authentication while allowing to secure all network communication through secured socket layer (SSL).

It should be used when VCS operations are performed over the Internet.
- **SVN Basic Authentication:**

Basic user name and password based authentication over custom SVN protocol supported by snvserve server.

Not secured. Passwords are sent over network in plain text. It should be avoided where security is concern.

- **SSH Authentication:**

Same as SVN authentication while sending all network communication over SSH.

Useful if you have an existing infrastructure that is heavily based on SSH accounts, and users already have system accounts on your server machine.

22.2 General Branching Guidelines

The following are the general branching guidelines that you must follow.

- Use Trunk for the main code line where the future development takes place.
- Use Branches for releasing code, for example, allowing stabilization, bug fixes, etc.
- Use Branches for parallel development, for example, when you have geographically separated teams or functionally separated teams.
- To minimize merges, keep branches to minimum and do not create personal branches.
- Always merge between the Trunk and the development branches, or between the release main branch and the release development branches.

Note: Never merge between development branches directly. This will create unmanageable code tracking.

- Remember the VCS KEY configured for your repository with Trunk. The same VCS Key must be configured for all development branches so that encrypted content can be populated and merged effectively.

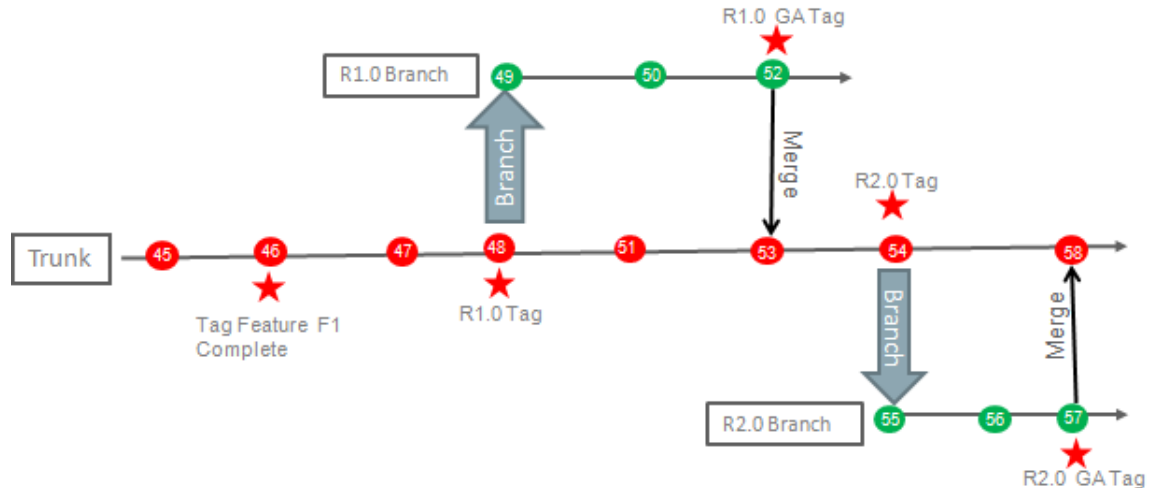
22.3 General Tagging Guidelines

The following are the general tagging guidelines that you must follow.

- When development of a project feature is completed, create a partial tag with all relevant artifacts.
- Create a tag for objects being released to QA or production through deployment archive. It will be useful for referring back to the released objects while bug fixing and debugging.
- Create Full tags on your development trunk or branch periodically (weekly, fortnightly, or monthly). If something goes wrong with your current code you can revert back to the last available full tag.
- If parallel development team is suppose to work on a subset of repository objects, then create a partial tag with only the relevant objects so that the development branch can be created only with the subset of objects.
- Create a full tag on the Trunk or branch where changes are being merged so that pre-merge state can be restored if something goes wrong during merge.
- Avoid merging branch artifacts directly. Rather perform branch merges through tags. Create a tag with the objects that are ready to be merged and then merge the changes to other branch or trunk using that tag.

- Create a tag before detaching or erasing any existing work repository. The VCS copy of the work repository objects will be deleted as part of these operations. So creating the tag will allow you to resurrect the objects if needed.

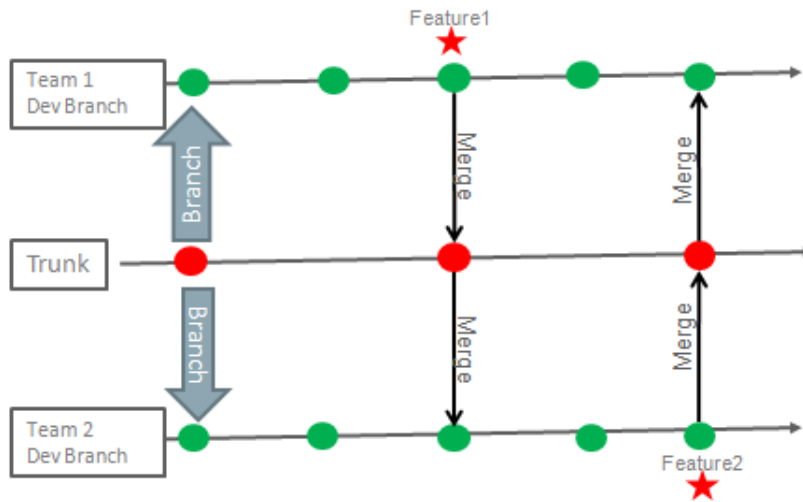
22.4 Branching Guidelines for Single Development Team



The following are the branching guidelines for single development teams.

- Use Trunk for development until close to the release.
Create Tags at various logical points.
- Create dev branch for release when you are close to the release.
 - Use release branch for stabilizing code and bug fixing.
 - Use Trunk for developing next release items.
 - Merge changes from release branch to Trunk at various logical points.
- At General Availability (GA).
 - Create Release Tag on the release branch.
 - Create Deployment Archives from the GA tag.
 - Merge changes from GA tag to the Trunk.

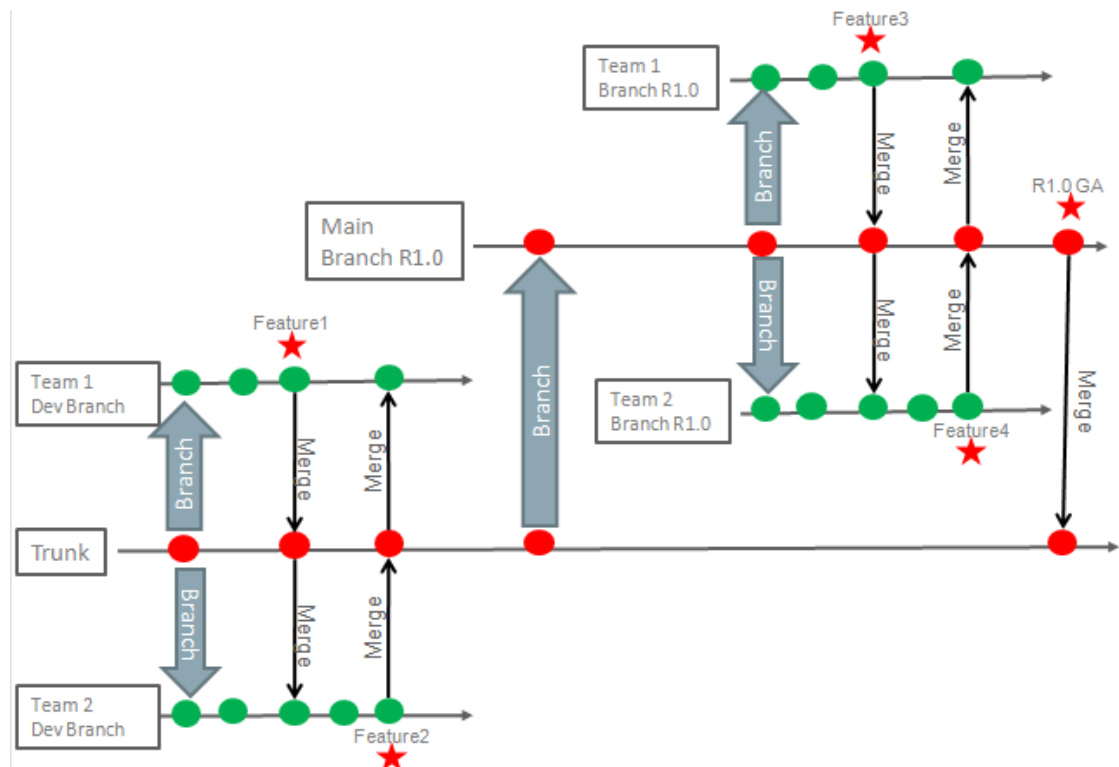
22.5 Branching Guidelines for Parallel Development Teams



The following are the branching guidelines for parallel development teams.

- Create separate development branch from Trunk for each parallel team.
- At various logical points:
 - Create Tags on the team specific dev branch.
 - Merge code to Trunk.
 - Notify other team to merge the changes from Trunk.

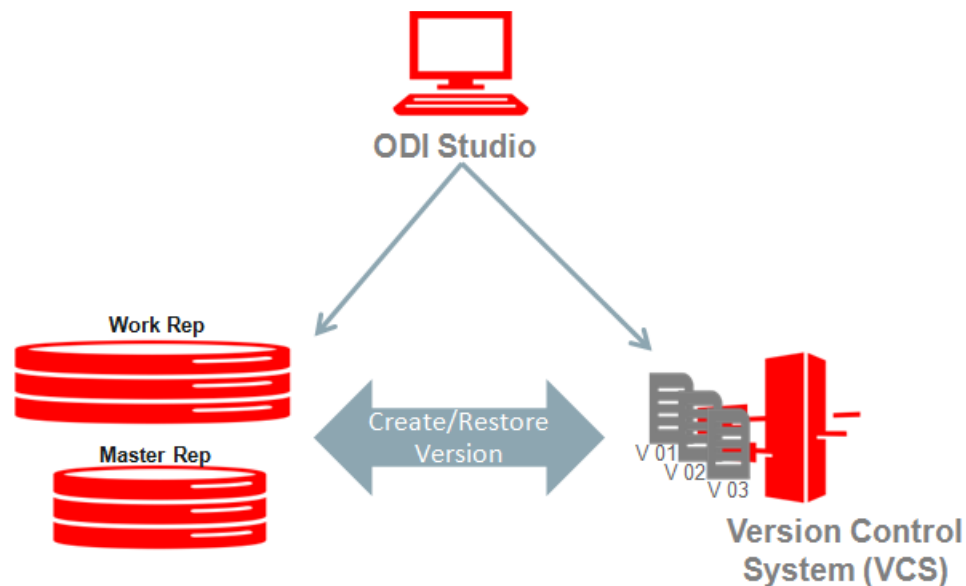
22.6 Guidelines for Release Branches for Parallel Development Teams



The following are the guidelines for working with release branches for parallel development teams.

- When close to the Release.
 - Create main development branch for the release from the Trunk.
 - Create a development branch for each team from the release's main branch.
- At various logical points.
 - Create tags on the team specific release branch.
 - Merge code from teams release branch to release main.
 - Notify other team to merge changes from the Release main.
- At General Availability (GA).
 - Create GA tag on Main Release branch.
 - Create Deployment Archive from GA tag.
 - Merge stabilized release code from GA tag into Trunk.

22.7 Guidelines for Versioning During Development



The following are the versioning guidelines that you must following during development.

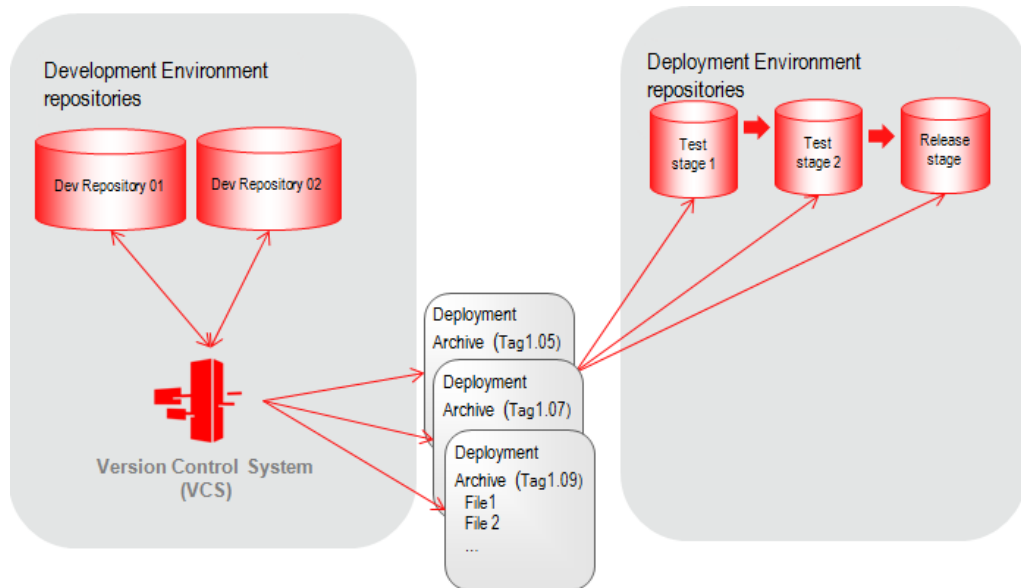
- Individual object version
 - Create version at stable check point. Discourage auto versioning since it gives you more control.
 - Periodically check stale versioned or non-versioned objects for the folder one is working on and create versions.
- Folder Version
 - Create folder to have a snapshot of entire folder for:
 - * Folder comparison
 - * Sync deleted, moved or renamed objects state with VCS

* Restore folder with children later

Note: Individual object versioning is independent of folder versioning.

- Create Versions of ODI objects at various logical points.
- Ensure to create versions of all relevant objects to maintain consistency of ODI objects the VCS.
- Have tags created at various logical points during development.

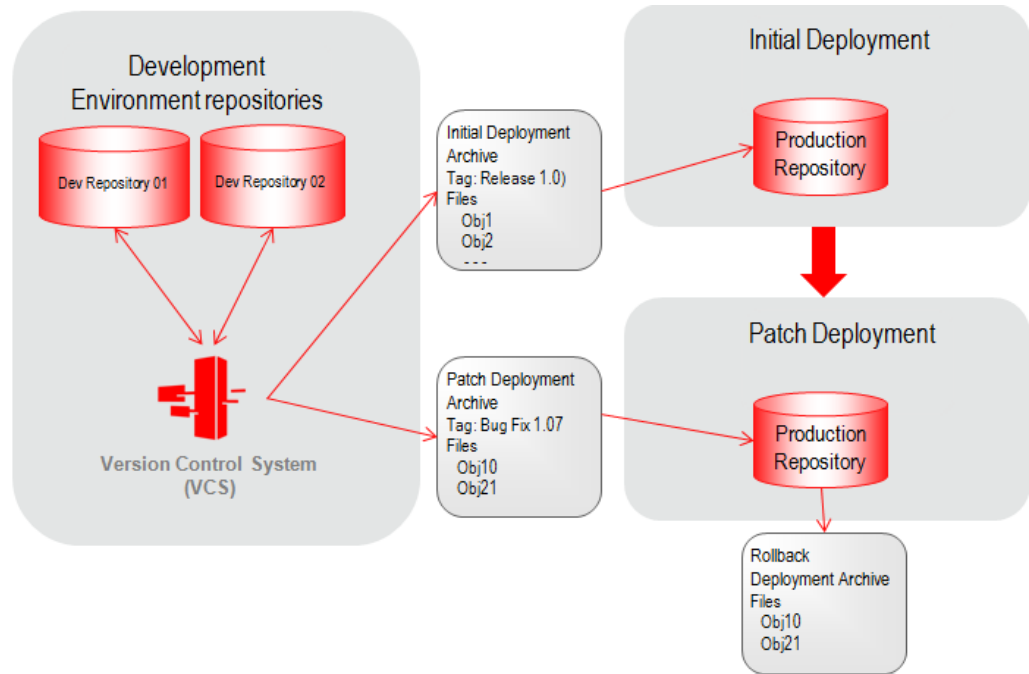
22.8 Guidelines for Deployment in Testing and Production Environments



The following are the guidelines for deployment in testing and production environments.

- Create Tag when objects are ready for testing.
- Create Deployment Archive (DA) from the Tag.
- Apply DA in the Testing environment.
 - File bugs.
 - Dev team to fix bugs and create new DA.
 - Continue testing with the new DA.
- DA is ready for Production environment when all the bugs are resolved.

22.9 Guidelines for Initial Deployment and Patching



The following are the guidelines that you must follow for initial deployment and patching.

Initial Deployment Archive

- Use for applying released artifacts.
- Must contain all released objects.
- Deploy initially on an empty repository.

Patch Deployment Archive

- Use for deploying bug fixes and enhancements.
- Must contain only the objects that are relevant to the fix or the enhancement.
- Create a Rollback Deployment Archive while applying a Patch Deployment Archive, so that the patch can be rolled back if something goes wrong.

Exporting and Importing

This chapter describes how to manage export and import operations in Oracle Data Integrator. An introduction to the import and export concepts is provided.

This chapter includes the following sections:

- [Import and Export Concepts](#)
- [Exporting and Importing Objects](#)
- [Repository-Level Export/Import](#)
- [Exporting the Technical Environment](#)
- [Exporting and Importing the Log](#)

23.1 Import and Export Concepts

This section introduces you to the fundamental concepts of export and import operations in Oracle Data Integrator. All export and import operations require a clear understanding of the concepts introduced in this section:

- [Global Identifiers \(GUIDs\)](#)
- [Export Keys](#)
- [Relationships between Objects](#)
- [Import Modes](#)
- [Tips for Import/Export](#)

23.1.1 Global Identifiers (GUIDs)

Oracle Data Integrator 12c introduces the use of globally-unique object identifiers. Unlike previous versions of ODI, in ODI 12c, object uniqueness across multiple work repositories is guaranteed by assigning GUIDs to all objects. In order to provide backward compatibility, Internal Identifiers are still available; however, they are only maintained across repositories when using ODI in 11g compatibility mode.

For more information about 11g compatibility mode, see: [Chapter 10, "Using Compatibility Mode."](#)

When creating an ODI entity, a GUID is automatically assigned to the object using the Java random UUID implementation. The only exception is when importing export files from releases previous to 12c. In order to ensure that ODI 11g objects when imported have reproducible, universally unique IDs, a Global Upgrade Key is required during the repository upgrade process. The upgrade key allows ODI to consistently calculate the same GUID for an 11g object. This key identifies uniquely the set of repositories

that were working together before an upgrade. An "Import Upgrade Key" must be specified when importing a pre-12c export file. This import upgrade key may be the same as the Global Upgrade Key (it usually should be), but is not required to be the same.

For more information, see: "Selecting the ODI Upgrade Key" in *Upgrading Oracle Data Integrator*.

23.1.2 Export Keys

Oracle Data Integrator 12c (12.1.3) introduces the use of an *Export Key* whenever exporting ODI objects which could contain sensitive data which should not be stored in plaintext. All ODI export dialogues will prompt you to enter an Export Key, which is used to encrypt sensitive (cipher) data using AES symmetric block two-way encryption. If you choose not to supply an Export Key, sensitive data will be stripped from the export file as it is saved.

The AES KEY for any sensitive data encryption needed during the export must be at least 8 characters and no more than 100 characters long. It should have at least one special character (@#%+/=) or digit, and at least one alphabetic lower or upper case character.

Dialogues which prompt for an Export Key also provide an option to "Save export key." If you select this prompt, ODI will remember (but never display) the Export Key you have just used, by saving it (obfuscated) in the current repository. This may be convenient when you want to use the same key for many subsequent export operations. As soon as you start typing new content into the Export Key field, however, this option is automatically deselected and any previously-remembered Export Key is lost, unless you cancel the export.

When importing objects, ODI presents a dialogue prompting you to either enter an Export Key, or import the file without cipher data. You must provide the identical Export Key that was used when the object was exported, or, you can opt to import the file without any sensitive data that is encrypted in the file.

When performing an import of multiple files, you will be prompted as needed for Export Keys for each file if they were exported with different Export Keys.

By default, ODI uses AES-128 encryption. You can select AES-128 or AES-256 (where available) during master repository creation/import, and you can specify the version used by ODI Studio in the `odi.conf` file.

23.1.3 Relationships between Objects

Oracle Data Integrator stores all objects in a relational database schema (the Repository) with dependencies between objects. Repository tables that store these objects maintain these dependencies as references using the Internal IDs and GUIDs. When you drag and drop a target datastore into a mapping, the reference to the GUID of this datastore is stored in the mapping object, along with the Internal ID and the fully-qualified name of the referenced object.

If you want to export this mapping, and import it in *Synonym mode* into another work repository, a datastore with the same GUID must already exist in this other work repository; otherwise, the mapping will have an unresolved reference. The unresolved references can be resolved either by fixing the imported object directly or by importing the missing object.

Therefore, the Model or Sub-model holding this datastore needs to be exported and imported in *Synonym mode* prior to importing the mapping.

You can use the *Smart export and import feature* or *solutions* to export and import sets of dependent objects.

- Use solutions in combination with versioning to maintain the dependencies when doing export/import. See [Chapter 19, "Using Version Control \(Legacy Mode\)."](#)
- It is recommended to use the Smart export and import feature because the dependencies are determined automatically.

There are also dependencies between work repository objects and master repository objects. Most references from work repository objects to master repository objects are made using Codes or Names. This means that only the Code of the objects (for example ORACLE is the code of the Oracle technology in the master) of the master repository objects are referenced in the work repository. There are some exceptions in the Mapping framework, such as in SnpMapRef, that also use Internal ID and GUID.

Dependencies within a work repository are ID-based.

It is important to import objects in the appropriate order. You can also use the Smart export and import feature to preserve these dependencies. [Table 23–1](#) lists the dependencies of a mapping to other objects when importing the mapping in synonym mode. Note that a Smart export automatically includes these dependent objects when exporting a mapping.

Table 23–1 Dependencies of a mapping in the work and Master Repository

Dependencies on other objects of Work Repository when importing in Synonym Mode	Dependencies on objects of the Master Repository
<ul style="list-style-type: none"> ■ (Parent/Child) Folder: Folder holding this mapping needs to be imported first. ■ (Reference) Model/Sub-Model: all Models/Sub-Models holding Datastore definitions referenced by the mapping need to be imported first. Datastore definitions including Attributes, Data Types, Primary Keys, Foreign Keys (references), Conditions must be exactly the same as the ones used by the exported mapping ■ (Reference) Global Variables, Sequences and Functions used within the mapping need to imported first ■ (Reference) Local Variables, Sequences and Function used within the mapping need to imported first ■ (Reference) Knowledge Modules referenced within the mapping need to be imported first ■ (Reference) Any mapping used as source in the current mapping needs to be imported first 	<ul style="list-style-type: none"> ■ Technology Codes ■ Context Codes ■ Logical Schema Names ■ Data Type Codes ■ Physical Server Names of the Optimization Contexts of Mappings

23.1.4 Import Modes

Oracle Data Integrator can import objects, the topology or repositories using several modes.

Read carefully this section in order to determine the import type you need.

Table 23–2 Import Types

Import Type	Description
Duplication	<p>This mode creates a new object (with a new GUID and internal ID) in the target Repository, and inserts all the elements of the export file.</p> <p>In Repositories in legacy compatible mode, the ID of this new object will be based on the ID of the Repository in which it is to be created (the target Repository). This does not apply to normal 12c Repositories.</p> <p>Dependencies between objects which are included into the export such as parent/child relationships are recalculated to match the new parent IDs. References to objects which are not included into the export are not recalculated.</p> <p>Note that this mode is designed to insert only 'new' elements.</p> <p>The Duplication mode is used to duplicate an object into the target repository. To transfer objects from one repository to another, with the possibility to ship new versions of these objects, or to make updates, it is better to use the three Synonym modes.</p> <p>This import type is not available for importing master repositories. Creating a new master repository using the export of an existing one is performed using the master repository Import wizard.</p>
Synonym Mode INSERT	<p>Tries to insert the same object (with the same GUID) into the target repository. The original object GUID is preserved.</p> <p>If an object of the same type with the same internal ID already exists then nothing is inserted.</p> <p>Dependencies between objects which are included into the export such as parent/child relationships are preserved. References to objects which are not included into the export are not recalculated.</p> <p>If any of the incoming attributes violates any referential constraints, the import operation is aborted and an error message is thrown.</p> <p>Note that sessions can only be imported in this mode.</p>
Synonym Mode UPDATE	<p>Tries to modify the same object (with the same GUID) in the repository.</p> <p>This import type updates the objects already existing in the target Repository with the content of the export file.</p> <p>If the object does not exist, the object is not imported. This applies to child objects also. So if new child objects are added in the source, and the parent object is exported and imported into the target using Synonym Mode UPDATE, then the new child objects will not be added in the target.</p> <p>Note that this import type does NOT delete child objects that exist in the repository but are not in the export file. For example, if the target repository contains a project with some variables and you want to replace it with one that contains no variables, this mode will update for example the project name but will not delete the variables under this project. The Synonym Mode INSERT_UPDATE should be used for this purpose.</p>

Table 23–2 (Cont.) Import Types

Import Type	Description
	<p><i>Example 1: (Scenario for a package)</i></p> <p>The same scenario exists in the source and target repositories.</p> <p>Source Repository:</p> <p>SCENARIO_1 (096bb382-5413-442f-97c5-aedbc3aa8caf)</p> <ul style="list-style-type: none"> - ODIBEEP_SCEN_STEP_1 (579c3271-ab52-45e8-bad2-826d9ba3c056) - ODIBEEP_SCEN_TASK_1 (4a0e1924-dfbc-49e6-a91b-e7fe8698de42) <p>Target Repository:</p> <p>SCENARIO_1 (096bb382-5413-442f-97c5-aedbc3aa8caf)</p> <ul style="list-style-type: none"> - ODIBEEP_SCEN_STEP_1 (579c3271-ab52-45e8-bad2-826d9ba3c056) - ODIBEEP_SCEN_TASK_1 (4a0e1924-dfbc-49e6-a91b-e7fe8698de42) <p>In the source repository, another step is added to the package and the scenario is regenerated.</p> <p>Source Repository:</p> <p>SCENARIO_1 (096bb382-5413-442f-97c5-aedbc3aa8caf)</p> <ul style="list-style-type: none"> - ODIBEEP_SCEN_STEP_1 (26da9aa1-f213-4c38-964e-5ea0d5c3d744) <p>Note: A new GUID is assigned by regeneration.</p> <ul style="list-style-type: none"> - ODIBEEP_SCEN_TASK_1 (a3621fa7-ebb5-46ed-928d-3f4bafcf47a3) <p>Note: A new GUID is assigned by regeneration.</p> <ul style="list-style-type: none"> - ODIDELETESCEN_SCEN_STEP_2 (5c356613-ea73-4b39-8269-890af79c944c) - ODIDELETESCEN_SCEN_TASK_2 (6d2fbb37-4498-40aa-80f6-829a86c8d70a) <p>Result if source exported and imported into target using Synonym Mode UPDATE.</p> <p>SCENARIO_1 (096bb382-5413-442f-97c5-aedbc3aa8caf)</p> <ul style="list-style-type: none"> - ODIBEEP_SCEN_STEP_1 (579c3271-ab52-45e8-bad2-826d9ba3c056) - ODIBEEP_SCEN_TASK_1 (4a0e1924-dfbc-49e6-a91b-e7fe8698de42) <p>The existing step and task are not updated since they were assigned a new global id during scenario regeneration. The new steps and tasks are not added since new objects are not inserted when using Synonym Mode UPDATE. So we can see that when adding steps / tasks and regenerating a scenario, that using Synonym Mode UPDATE to import into the target will not give desirable results. Due to the nature of Scenarios, Synonym Mode INSERT_UPDATE should be used to move Scenarios between repositories.</p>

Table 23–2 (Cont.) Import Types

Import Type	Description
	<p><i>Example 2:</i></p> <p>Source Repository: TABLE_1 (863b3cc1-3a3c-4011-b3bc-0ce236e41f22) - COL_1_RENAMED (210a73ef-4919-4eab-9e39-a3153300f8b0) - COL_NEW (0cca9009-cb11-4e4d-892e-01014b0a1592)</p> <p>Target Repository: TABLE_1 (863b3cc1-3a3c-4011-b3bc-0ce236e41f22) - COL_1 (210a73ef-4919-4eab-9e39-a3153300f8b0) - COL_DEL (0cca9009-cb11-4e4d-892e-01014b0a1592)</p> <p>Result if source is exported and imported into the target using Synonym Mode UPDATE: TABLE_1 (863b3cc1-3a3c-4011-b3bc-0ce236e41f22) - COL_1_RENAMED (210a73ef-4919-4eab-9e39-a3153300f8b0) - COL_DEL (0cca9009-cb11-4e4d-892e-01014b0a1592)</p> <p>Note the following:</p> <ul style="list-style-type: none"> ■ COL_1 is updated to COL_1_RENAMED. ■ COL_DEL remains as it was before the import as objects are not deleted when using Synonym Mode Update. ■ COL_NEW is not added as new objects are not inserted when using Synonym Mode Update.
Synonym Mode INSERT_UPDATE	<p>If no ODI object exists in the target Repository with an identical GUID, this import type will create a new object with the content of the export file. Already existing objects (with an identical GUID) will be updated; the new ones, inserted.</p> <p>Existing child objects will be updated, non-existing child objects will be inserted, and child objects existing in the repository but not in the export file will be deleted.</p> <p>Dependencies between objects which are included into the export such as parent/child relationships are preserved. References to objects which are not included into the export are not recalculated.</p> <p>This import type is not recommended when the export was done without the child components. This will delete all sub-components of the existing object.</p>

Table 23–2 (Cont.) Import Types

Import Type	Description
Import Replace	<p>This import type replaces an already existing object in the target repository by one object of the same object type specified in the import file.</p> <p>This import type is only supported for scenarios, Knowledge Modules, actions, and action groups and replaces all children objects with the children objects from the imported object.</p> <p>Note the following when using the Import Replace mode:</p> <p>If your object was currently used by another ODI component like for example a KM used by a mapping, this relationship will not be impacted by the import, the mappings will automatically use this new KM in the project.</p> <p>Warnings:</p> <ul style="list-style-type: none"> ▪ When replacing a Knowledge module by another one, Oracle Data Integrator sets the options in the new module using option name matching with the old module's options. New options are set to the default value. It is advised to check the values of these options in the mappings. ▪ Replacing a KM by another one may lead to issues if the KMs are radically different. It is advised to check the mapping's design and execution with the new KM.

23.1.5 Tips for Import/Export

This section provides tips for the import and export operations.

Repository IDs

When importing ODI 11g objects, use an Upgrade Key to compute a GUID that is based on the legacy Internal ID. When importing from ODI Studio, an Upgrade Key is prompted for when the import is started and it is determined that the import file is from before 12c. When import is not interactive (that is, run from a command line), then an error is thrown if the import needs an Upgrade Key and one has not been specified. For more information, see [Chapter 10, "Using Compatibility Mode."](#)

When importing, objects are matched by GUID. If a match is found, then that object will use the Internal ID of the matching object from the target repository. If a match is not found, then the behavior is as follows:

- If the target repository is not legacy ID compatible, then a new ID is assigned.
- If the target repository is legacy ID compatible, then the ID of the source object from the import file is used.
- If the import is in `DUPLICATION` mode, then a new Internal ID is always assigned.

Export/Import Reports

A report is displayed after every export or import operation. It is advised to read it carefully in order to determine eventual errors of the import process.

Depending on the export or import operation performed, this report gives you details on, for example, the:

- **Import type**
- **Imported Objects.** For every imported object the object type, the original object name, the object name used for the import, the original ID, and the new, recalculated ID/GUID after the import is given.
- **Deleted Objects.** For every deleted object the object type, the object name, and the original ID/GUID is given.

- **Created Missing References** lists the missing references detected after the import.
- **Fixed Missing References** lists the missing references fixed during the import.

The reports displayed after a smart export or smart import operation contain additional details to describe what happened to the objects during the export or import, for example which objects have been ignored, merged, overwritten and so forth.

You can save the import report as an .xml or .html file. Click **Save...** to save the import report.

Missing References

In order to avoid missing references, use either the Smart Export and Import feature or solutions to manage dependencies. For more information, see "[Smart Export and Import](#)" on page 23-14 and "[Working with Labels](#)" on page 19-7.

Import Type

Choose the import type carefully. See "[Import Modes](#)" on page 23-3 for more information.

23.2 Exporting and Importing Objects

Exporting and importing Oracle Data Integrator objects means transferring objects between different repositories.

When exporting an Oracle Data Integrator object, an XML export file is created. ODI objects have dependencies, as described in "[Relationships between Objects](#)" on page 23-2. These dependencies will be exported in the XML export file.

The content of this XML file will depend on the export method you will use:

- [Exporting an Object with its Child Components](#)
- [Exporting an Object without its Child Components](#)

The choice will depend on your goal, if you need to do a partial export then the **Export Without Child Components** is the one to use.

The [Export Multiple ODI Objects](#) feature is useful when you need to regularly export the same set of Objects.

Once the export has been performed, it is very important to choose the import strategy to suite your requirements.

The [Smart Export and Import](#) feature is a lightweight and consistent export and import mechanism. It supports the export and import of one or multiple ODI objects. It is recommended to use this feature to avoid most of the common issues that are encountered during an export or import.

This section contains the following topics:

- [Exporting an Object with its Child Components](#)
- [Exporting an Object without its Child Components](#)
- [Partial Export/Import](#)
- [Exporting one ODI Object](#)
- [Export Multiple ODI Objects](#)
- [Importing Objects](#)

- [Smart Export and Import](#)

23.2.1 Exporting an Object with its Child Components

This option is the most common when you want to export an object. It allows you to export all subcomponents of the current object along with the object itself.

When an Object is exported with its child components, all container-dependent Objects – those which possess a direct parent/child relationship - are also exported. Referenced Objects are not exported.

For example, when you choose to export a Project with its child components, the export will contain the Project definition as well as all objects included in the Project, such as Folders, Mappings, Procedures, Packages, Knowledge Modules, Variables, Sequences, Functions, etc. However, this export will not contain dependent objects referenced which are outside of the Project itself, such as Datastores and Attributes, as defined previously in "[Relationships between Objects](#)" on page 23-2. The numeric Internal ID references of these Objects will be exported. Additionally, the GUID of the referenced object is also exported, using a special `SnpFKXRef` object in the export file.

23.2.2 Exporting an Object without its Child Components

This option can be useful in some particular situations where you would want to take control of the import process. It allows you to export only the top-level definition of an object without any of its sub-objects.

For example, if you choose to export a Model without its children, it will only contain the Model definition but not the underlying Sub-models and Datastores when you import this model to a new repository.

23.2.3 Partial Export/Import

If you have a very large project that contains thousands of mappings and you only want to export a subset of these to another work repository, you can either export the entire Project and then import it, or choose to do a partial manual export/import as follows:

1. Export all Models referenced by the sub-items of your project and import them in **Synonym mode** in the new repository to preserve their GUIDs
2. Export the Project without its children and import it in *Synonym mode*. This will simply create the empty Project in the new repository (with the same GUIDs as in the source).
3. Export every first level Folder you want, without its children, and import them in *Synonym mode*. The empty Folders will be created in the new repository.
4. Export and Import all Markers, Knowledge Modules, Variables, Sequences, and so forth that are referenced by every object you plan to export, and import them in *Synonym mode*. See "[Import Modes](#)" on page 23-3 for more information on the Synonym or Duplication mode and the impact on Object GUIDs and Internal IDs for special caution regarding the import of Knowledge Modules in *Synonym mode*.
5. Finally, export the mappings you are interested in and import them in *Synonym mode* in the new repository.

23.2.4 Exporting one ODI Object

Exporting one Oracle Data Integrator Object means export one single ODI object in order to transfer it from one repository to another.

To export an object from Oracle Data Integrator:

1. Select the object to be exported in the appropriate Oracle Data Integrator Navigator.

2. Right-click the object, and select **Export...**

If this menu item does not appear, then this type of object does not have the export feature.

3. In the Export dialog, set the Export parameters as indicated in [Table 23-3](#).

Table 23-3 Object Export Parameters

Properties	Description
Export Directory	Directory in which the export file will be created.
Export Name	Name given to the export
Child Components Export	<p>If this option is checked, the objects linked to the object to be exported will be also exported. These objects are those visible under the exported object in the tree. It is recommended to leave this option checked. Refer to "Exporting an Object with its Child Components" on page 23-9 for more details.</p> <p>Note that when you are exporting a Load Plan, scenarios will not be exported even if you check this option.</p>
Replace exiting files without warning	If this option is checked, the existing file will be replaced by the ones of the export. If a file with the same name as the export file already exists, it will be overwritten by the export file.
Encryption	These fields allow you to provide an Export Key, used to encrypt any sensitive data that is contained in the exported object. See: "Export Keys" on page 23-2 for details.
Export Key	<p>Specifies the AES KEY for any sensitive data encryption needed during the export.</p> <p>The export key string is minimum 8 characters long and maximum 100 characters long. It should have at least one special character (@#\$\$%+/=) or digit, and at least one alphabetic lower or upper case character.</p>
Confirm Export Key	Enter your Export Key again.
Save Export Key	If checked, your Export Key is saved for all future exports.
Advanced options	This set of options allow to parameterize the XML output file format. It is recommended that you leave the default values.
XML Version	<p>XML Version specified in the export file. Parameter .xml version in the XML file header.</p> <pre><?xml version="1.0" encoding="ISO-8859-1"?></pre>
Character Set	<p>Encoding specified in the export file. Parameter encoding in the XML file header.</p> <pre><?xml version="1.0" encoding="ISO-8859-1"?></pre>
Java Character Set	Java character set used to generate the file

You must at least specify the **Export Name**.

4. Click **OK**.

The object is exported as an XML file in the specified location.

23.2.5 Export Multiple ODI Objects

You can export one or more objects at once, using the **Export Multiple Objects** action. This lets you export ODI objects to a zip file or a directory, and lets you re-use an existing list of objects to export.

More powerful mechanisms for doing this are Solutions and also the Smart Export and Import. For more information, see ["Working with Labels"](#) on page 19-7 or ["Smart Export and Import"](#) on page 23-14.

To export multiple objects at once:

1. Select **Export...** from the Designer, Topology, Security or Operator Navigator toolbar menu.
2. In the Export Selection dialog, select **Export Multiple Objects**.
3. Click **OK**.
4. In the Export Multiple Objects dialog, specify the export parameters as indicated in [Table 23-3](#).

The objects are either exported as .xml files directly into the directory, or as a zip file containing .xml files. If you want to generate a zip file, you need to select **Export as zip file** and enter the name of the zip file in the **Zip file name** field.

5. Specify the list of objects to export:
 1. Drag and drop the objects from the Oracle Data Integrator Navigators into the Export list. Note that you can export objects from different Navigators at once.
 2. Click **Load a list of objects...** to load a previously saved list of objects. This is useful if you regularly export the same list of objects.
 3. To save the current list of objects, click **Save Export List** and specify a file name. If the file already exists, it will be overwritten without any warning.
6. Click **OK** to start the export.

To import multiple objects at once, you must use a Label or the Smart Import. See ["Working with Labels"](#) on page 19-7 and ["Smart Export and Import"](#) on page 23-14 for more information.

23.2.6 Importing Objects

Importing and exporting allows you to transfer objects (Mappings, Knowledge Modules, Models, and so on) from one repository to another.

When importing Knowledge Modules choose carefully your import strategy which may depend on the knowledge module's scope. See ["Project and Global Knowledge Modules"](#) on page 6-4 for more information.

This section includes the following topics:

- [Importing an ODI object](#)
- [Importing 11g Objects into a 12c Environment](#)
- [Importing a Project KM](#)
- [Importing a KM in Replace Mode](#)
- [Importing a Global Knowledge Module](#)

Importing an ODI object

To import an object in Oracle Data Integrator:

1. In the Navigator, select the object or object node under which you want to import the object.
2. Right-click the object, and select **Import**, then the type of the object you wish to import.
3. In the Import dialog:
 1. Select the **Import Type**. See "[Import Modes](#)" on page 23-3 for more information.
 2. Enter the **File Import Directory**.
 3. Select the file(s) to import from the list.
4. Click **OK**.

The XML-formatted files are imported into the work repository, and the imported objects appear in the Oracle Data Integrator Navigators.

Note that the parent or node under which objects are imported is dependent on the import type used. When using DUPLICATION mode, the objects will be imported into where the Import option was selected. For Synonym imports, the objects will be imported under the parent specified by the objects parent ID in the import file.

Importing 11g Objects into a 12c Environment

Any versionable 11g object can be imported into a 12c ODI environment.

The following objects can be checked in as versions and can be imported:

- Project, Folder
- Package, Scenario
- Interface, Procedure, Knowledge Module
- Sequence, User Function, Variable
- Model, Model Folder
- Label
- Load Plan

When importing objects, you must define an Upgrade Key. ODI uses this key to generate a unique GUID for the objects.

Note: 11g interfaces can only be imported into a 12c repository in either SYNONYM INSERT mode or DUPLICATION mode. That is because of the complex transformation taking place when interfaces are converted into mappings.

See Also: For more information about upgrading repositories and the Upgrade Key, see "Selecting the ODI Upgrade Key" in *Upgrading Oracle Data Integrator*.

Importing a Project KM

To import a Knowledge Module into an Oracle Data Integrator project:

1. In Designer Navigator, select the project into which you want to import the KM.
2. Right-click the project, and select **Import > Import Knowledge Modules...**
3. In the Import dialog:
 1. The **Import Type** is set to Duplication. Refer to "[Import Modes](#)" on page 23-3 for more information.
 2. Enter the **File Import Directory**.
 3. Select the Knowledge Module file(s) to import from the list.
4. Click **OK**.

The Knowledge Modules are imported into the work repository and appear in your project under the Knowledge Modules node.

Importing a KM in Replace Mode

Knowledge modules are usually imported into new projects in Duplication mode. See "[Import Modes](#)" on page 23-3 for more information.

When you want to replace a global KM or a KM in a project by another one and have all mappings automatically use the new KM, you must use the Import Replace mode. See "[Import Modes](#)" on page 23-3 for more information.

To import a Knowledge Module in Replace mode:

1. Select the Knowledge Module you wish to replace.
2. Right-click the Knowledge Module and select **Import Replace...**
3. In the Replace Object dialog, specify the Knowledge Module export file.
4. Click **OK**.

The Knowledge Module is now replaced by the new one.

WARNING: Replacing a Knowledge module by another one in Oracle Data Integrator sets the options in the new module using the option name similarities with the old module's options. New options are set to the default value.

It is advised to check the values of these options in the mappings as well as the mappings' design and execution with the new KM.

Refer to the Import Replace mode description in "[Import Modes](#)" on page 23-3 for more information.

Importing a Global Knowledge Module

To import a global knowledge module in Oracle Data Integrator:

1. In the Navigator, select the Global Knowledge Modules node in the Global Objects accordion.
2. Right-click and select **Import Knowledge Modules**.
3. In the Import dialog:
 1. Select the **Import Type**. See "[Import Modes](#)" on page 23-3 for more information.
 2. Enter the **File Import Directory**.

3. Select the file(s) to import from the list.
4. Click **OK**.
5. If prompted, enter the Export Key used when this object was exported. If you do not enter an Export Key, any encrypted sensitive (cipher) data will be stripped from the imported object. For more information about the Export Key, see: "[Export Keys](#)" on page 23-2.

The global KM is now available in all your projects.

23.2.7 Smart Export and Import

It is recommended to use the *Smart Export and Import* feature to avoid most of the common issues that are encountered during an export or import such as broken links or ID conflicts. The Smart Export and Import feature is a lightweight and consistent export and import mechanism providing several smart features.

The *Smart Export* automatically exports an object with all its object dependencies. It is particularly useful when you want to move a consistent lightweight set of objects from one repository to another and when you want to include only a set of modified objects, for example in a patching use case, because Oracle Data Integrator manages all object dependencies automatically while creating a consistent sub-set of the repository.

The *Smart Import* provides:

- Automatic and customizable object matching rules between the objects to import and the objects already present in the repository
- A set of actions that can be applied to the object to import when a matching object has been found in the repository
- Proactive issue detection and resolution that suggests a default working label for every broken link or conflict detected during the Smart Import

23.2.7.1 Performing a Smart Export

To perform a Smart Export:

1. Select **Export...** from the Designer, Topology, Security or Operator Navigator toolbar menu.
2. In the Export Selection dialog, select **Smart Export**.

Note: This option is only available if you are connected to a Work repository.

3. Click **OK**.
4. In the Smart Export dialog, specify the export parameters as follows:
 - In the Export Name field, enter the name given to the export (mandatory). Default is `SmartExport.xml`.
 - The objects are either exported into a single `.xml` file directly in the directory, or as a zip file containing a single `.xml` file. If you want to generate a zip file, you need to select **Export as zip file** and enter the name of the zip file in the Zip file name field.
 - Optionally, enter an **Export key** used to encrypt sensitive data. For more information about the Export Key, see: "[Export Keys](#)" on page 23-2.

- Optionally, customize the XML output file format in the **Encoding Options** section. It is recommended that you leave the default values.

Properties	Description
XML Character Set	Encoding specified in the export file. Parameter encoding in the XML file header. <?xml version="1.0" encoding="ISO-8859-1"?>
Java Character Set	Java character set used to generate the file

- In the **Dependencies** section, drag and drop the objects you want to add to the Smart Export from the Oracle Data Integrator Navigators into the **Selected Objects** list on the left. Note that you can export objects from different Navigators at once.

The object to export appears in a tree with all its related parent and child objects that are required to support.

Repeat this step according to your needs.

Notes:

- If your export contains *shortcuts*, you will be asked if you want to materialize the shortcuts. If you select **No**, both the shortcuts and the base objects will be exported.
 - A **bold** object name indicates that this object has been specifically added to the Smart Export. Only objects that appear in **bold** can be removed. Removing an object also removes its child objects and the dependent objects of the removed object. Note that child objects of a specifically added object also appear in **bold** and can be removed. To remove an object from the export tree, right-click the object and select **Remove Object**. If the removed object is dependent of another object, it will remain in the tree but will be shown in normal (non-bold) typeface.
 - A **grayed out** object name indicates that this object is an dependent object that will not be exported, as for example a technology.
-
-
- Optionally, modify the list of objects to export. You can perform the following actions: Remove one object, remove all objects, add objects by release tag, and add shortcuts. See "[Change the List of Objects to Export](#)" on page 23-16 for more information.
 - If there are any cross reference objects, including shortcuts, they are displayed in the **Dependencies** list on the right. Parent objects will not be shown under the *Uses* node and child objects will not be shown under *Used By* node.

- Click **Export** to start the export process.






The Smart export generates a single file containing all the objects of the **Selected Objects** list. You can use this export file as the input file for the Smart Import. See "[Performing a Smart Import](#)" on page 23-17 for more information.

You can review the results of the Smart Export in the Smart Export report.

The Smart Export Toolbar

The *Smart Export toolbar* provides tools for managing the objects to export and for viewing dependencies. [Table 23–4](#) details the different toolbar components.

Table 23–4 *Smart Export Toolbar*

Icon	Name	Description
	Search	Searches for a object in the Selected Objects or Dependencies list.
	Expand All	Expands all tree nodes in the Selected Objects or Dependencies list.
	Collapse All	Collapses all tree nodes in the Selected Objects or Dependencies list.
	Clear All	Deletes all objects from the list. Warning: This also deletes Release Tags and Materialization selections.
	Add Objects by Release Tag	Adds all objects that have the same release tag as the object already in the Selected Objects list.

Change the List of Objects to Export

You can perform the following actions to change the list of objects to export:

- *Remove one object from the list*

Only objects that have been explicitly added to the Smart Export (objects in bold) can be removed from the **Selected Objects** list.

To remove one object:

1. In the **Selected Objects** list, select the object you wish to remove.
2. Right-click and select **Remove Object**.

The object and its dependencies are removed from the Selected Objects list and will not be included in the Smart Export.

Note: If the object you wish to remove is a dependent object of another object to export, it remains in the list but becomes un-bold.

- *Remove all objects from the list*

To delete all objects from the **Selected Objects** list, select **Clear All** in the Smart Export Toolbar.

Caution: This also deletes Release Tags and Materialization selections.

- *Add objects by release tag*

To add a folder or model folder of a certain release:

1. Select **Add Objects by Release Tag** in the Smart Export Toolbar.

This opens the Release Tag Selection dialog.

2. In the Release Tag Selection dialog, select a release tag from the Release Tag list. All objects of this release tag will be added to the Smart Export. You don't need to add them individually to the Smart Export.

The Release Tag Selection dialog displays the list of release tags that have been already added to the Smart Export.

3. Click **OK** to add the objects of the selected release tag to the Smart Export.

The release tag name is displayed in the **Selected object** list after the object name.

Note: When you add a folder or model folder to the **Selected Objects** list that has a release tag, you can choose to automatically add all objects of the given release to the Smart Export by clicking **OK** in the Confirmation dialog.

- *Add shortcuts*

If you add shortcuts to the Smart Export, you can choose to materialize the shortcut. If you choose *not* to materialize a shortcut added to the Smart Export, then the shortcut is exported with all its dependent objects, including the base object. If you choose to materialize the shortcut, the shortcut is materialized and the base object referenced through the shortcut is not included.

23.2.7.2 Performing a Smart Import

Note: When performing a Smart Import of ODI 11g objects, you must specify an Upgrade Key to be used to generate a new GUID for the object. ODI Studio will prompt you for this Upgrade Key if it detects that you are importing a pre-12c export file. For more information, see: [Chapter 10, "Using Compatibility Mode."](#)

To perform a Smart Import:

1. Select **Import...** from the Designer, Topology, Security or Operator Navigator toolbar menu.
2. In the Import Selection dialog, select **Smart Import**.
3. Click **OK**.

The Smart Import wizard opens.

4. On the first screen, *Step 1 - File Selection*, specify the import settings as follows:
 - a. In the **File Selection** field, enter the location of the Smart Export file to import.
 - b. Optionally, select a response file to replay a previous Smart Import wizard execution by presetting all fields from the **Response File** field.

Note: Actions from the Response File are used only when Oracle Data Integrator detects a conflict during the import. If there is no conflict, the default action is used.

- c. Click **Next** to move to the next step of the Smart Import Wizard.

If an exported file has sensitive (cipher) data and was exported with an Export Key, then after clicking **Next**, the Enter Export Key dialog is shown. Provide an Export Key in order to include the encrypted cipher data in your import. Alternatively, enable **Import File without cipher data** and leave the field for the Export Key empty, to import only non-cipher data. For more information about the Export Key, see: "[Export Keys](#)" on page 23-2.

Oracle Data Integrator launches a matching process that verifies whether the repository contains matching objects for each of the potential objects to import.

- 5. On the second screen, *Step 2 - Import Actions*, verify the result of the matching process and fix eventual issues. The number of detected issues is displayed in the first line of this screen.

Note that the Smart Import wizard suggests default values for every field.

- a. In the **Object Match Details** section, expand the nodes in the **Import Object** column to navigate to the objects available to import during this Smart Import.
- b. In the **Action** column, select the action to perform on the object during the import operation. Possible values are listed in [Table 23-5](#).

Table 23-5 Actions during Import

Action	Description
Merge	For containers, this means overwrite the target container with the source container, and then loop over the children for merging. Each child may have a different action. Child FCOs that are not in the import file will not be deleted. The Merge action may also be used for Datastores, which will be merged at the SCO level.
Overwrite	Overwrite target object with source object. Any child objects remaining after import come from the source object. Note that this applies to all the child objects (If a project overwrites another, all the folders in this project will be replaced and any extra folders will be removed).
Create Copy	Create source object including renaming or modifying any fields needed to avoid conflict with existing objects of same name/id/code. This action preserves the consistency and relations from and to the imported objects.
Reuse	Do not import the object, yet preserve the ability import all objects related to it and link them to the target object. Basically, this corresponds to overwriting the source object with the matched target object.
Ignore	Do not process the source object.

- c. In the **Repository Object** column, select the required repository object. This is the repository object that matches the best the import object.
- d. If an issue, such as a broken link or a code conflict, has been detected during the matching process, a warning icon is displayed in the **Issues** column. View the **Issue Details** section for more details on the issue.

Note: The **Next** button is disabled until all critical issues are fixed.

- e. The table in the **Issue Details** section lists the issues that have been detected during the matching process. To fix an issue, select the action to perform in the **Action** column. [Table 23-6](#) describes the possible actions.

Table 23–6 Possible actions to fix an issue

Action	Description
Ignore	Not possible on critical issues
Change	If a name or code collision is detected, specify the new value in the Fix field. Note: If ODI displays a Different Context is already set as the default context message, set the Fix value to 0.
Do not change	For value changed issues, the value in the matching target object will be kept.
Fix Link	For broken links, click Search in the Fix field.

Note: Oracle Data Integrator provides a default working label for every issue. However, missing references may still result during the actual import process depending on the choices you made for the import actions.

- f. In the **Fix** column, specify the fix. For example, for broken links, click **Search** and select the target object in the Broken Link Target Object Selection dialog.

Note:

- g. Click **Next** to move to the next step of the Smart Import Wizard.
6. On the third screen, *Step 3 - Summary*, review the import file name and eventual issues.
 - a. In the **File Selection** field, verify the import file name.
 - b. If the Smart Import still contains unresolved warnings, they are displayed on this screen. Note that critical issues are not displayed here. To fix them, click **Back**.
 - c. Optionally, select **Save Response File** to create a response file that you can reuse in another import to replay this Smart Import wizard execution by presetting all fields.
 - d. Click **Finish** to launch the Smart Import and to finalize of the Smart Import Wizard.

You can review the results of the Smart Import in the Smart Import report.

23.3 Repository-Level Export/Import

At repository level you can export and import the master repository and the work repositories.

23.3.1 Exporting and Importing the Master Repository

The master repository export/import procedure allows you to transfer the whole repository (Topology and Security domains included) from one repository to another.

It can be performed in Topology Navigator, to import the exported objects in an existing repository, or while creating a new master repository.

Exporting the Master Repository in Topology Navigator

The objects that are exported when exporting the master repository are objects, methods, profiles, users, languages, versions (if option selected), solutions (if option selected), open tools, password policies, entities, links, fields, lookups, technologies, datatypes, datatypes conversions, logical agents, contexts and the child objects.

To export a master repository:

1. Select **Export...** from the Designer, Topology, Security or Operator Navigator toolbar menu.
2. In the Export Selection dialog, select **Export the Master Repository**.
3. Click **OK**.
4. In the Export Master Repository dialog, set the Export parameters as indicated in [Table 23–3, "Object Export Parameters"](#).

The master repository and its topology and security settings are either exported as .xml files directly into the directory, or as a zip file containing .xml files. If you want to generate a zip file, you need to select **Export to zip file** and enter the name of the zip file in the **Zip File Name** field.

5. Select **Export versions**, if you want to export all stored versions of objects that are stored in the repository. You may wish to unselect this option in order to reduce the size of the exported repository, and to avoid transferring irrelevant project work.
6. Select **Export solutions**, if you want to export all stored solutions that are stored in the repository. You may wish to unselect this option for similar reasons.
7. Click **OK**.

The export files are created in the specified export directory.

Importing the Master Repository

To import the exported master repository objects into an existing master repository:

1. Select **Import...** from the Designer, Topology, Security or Operator Navigator toolbar menu.
2. In the Import Selection dialog, select **Import the Master Repository**.
3. Click **OK**.
4. In the Import dialog:
 - a. Select the **Import Mode**. Refer to ["Import Modes"](#) on page 23-3 for more information.
 - b. Select whether you want to import the files **From a Folder** or **From a ZIP file**.
 - c. Enter the file import folder or zip file.
5. Click **OK**.
6. If prompted, enter the Export Key used when this object was exported. If you do not enter an Export Key, any encrypted sensitive (cipher) data will be stripped from the imported object. For more information about the Export Key, see: ["Export Keys"](#) on page 23-2.

The master repository contains now the objects you have imported.

Note: The import is not allowed if the source and target repositories have the same Internal ID and have different repository timestamps. This can only happen with 11g-compatible repositories, because 12c repositories have a unique Global ID.

If the target 11g-compatible repository has the same Internal ID as the source repository, you can *renumber* the repository. This operation should be performed with caution. See "[About Internal Identifiers \(IDs\)](#)" on page 10-2 for more information on the risks, and how to renumber a repository is described in "[Renumbering Repositories](#)" on page 10-3.

Creating a new Master Repository using a previous Master export

To create a new master repository using an export of another master repository:

1. Open the New Gallery by choosing **File > New**.
2. In the New Gallery, in the Categories tree, select **ODI**.
3. Select from the Items list the **Master Repository Import Wizard**.
4. Click **OK**.

The Master Repository Import Wizard appears.

5. Specify the **Database Connection** parameters as follows:
 - **Login:** User ID/login of the owner of the tables you have created for the master repository
 - **JDBC Driver:** The driver used to access the technology, which will host the repository.
 - **JDBC URL:** The complete path for the data server to host the repository.
Note that the parameters **JDBC Driver** and **URL** are synchronized and the default values are technology dependent.
 - **User:** The user id/login of the owner of the tables.
 - **Password:** This user's password.
 - **DBA User:** The database administrator's username
 - **DBA Password:** This user's password
6. Specify the **Repository Configuration** parameters as follows:
 - **Id** (Importing legacy ID-compatible repositories only): When importing an ODI 11g repository, you must specify the ID of the repository. You do not need to specify an ID when importing repositories that are not legacy ID-compatible.
 - **Use a Zip File:** If using a compressed export file, check the **Use a Zip File** box and select in the **Export Zip File field** the file containing your master repository export.
 - **Export Path:** If using an uncompressed export, select the directory containing the export in the **Export Path** field.
 - **Technology:** From the list, select the technology your repository will be based on.
7. Click **Test Connection** to test the connection to your master repository.

The Information dialog opens and informs you whether the connection has been established.

8. Click **Next**.
9. Specify the password storage details:
 - Select **Use Password Storage Configuration specified in Export** if you want to use the configuration defined in the export.
 - Select **Use New Password Storage Configuration** if you do not want to use the configuration defined in the export and select
 - **Internal Password Storage** if you want to store passwords in the Oracle Data Integrator repository
 - **External Password Storage** if you want use JPS Credential Store Framework (CSF) to store the data server and context passwords. Indicate the **MBean Server Parameters** to access the credential store as described in "MBean Server Parameters" in *Administering Oracle Data Integrator*.

Refer to "Setting Up External Password Storage" in *Administering Oracle Data Integrator* for more information on password storage details.

10. In the Master Repository Import Wizard click **Finish** to validate your entries.

If an exported file for the Master Repository has sensitive (cipher) data and was exported with an Export Key, then after clicking **Next**, the Enter Export Key dialog is shown. Provide an Export Key in order to include the encrypted cipher data in your import.

If a file for the Master Repository import was exported without cipher data, or if you enable **Import File without cipher data** and leave the field for the Export Key empty, another dialog for creating a new password for the ODI SUPERVISOR user will be displayed. You will need to create a new password.

For more information about the Export Key, see: "[Export Keys](#)" on page 23-2.

A new repository is created and the exported components are imported in this master repository.

23.3.2 Export/Import Topology and Security Settings

Exporting and then importing the topology or security settings allows you to transfer a domain from one master repository to another.

Exporting the Topology and Security Settings

The domains that can be exported are given below:

- **Topology:** the full topology (logical and physical architectures including the local repository, data servers, hosts, agents, generic actions, technologies, datatypes, logical schemas, and contexts).
- **Logical Topology:** technologies (connection, datatype or language information), logical agents, logical schemas, actions and action groups.
- **Security:** objects, methods, users, profiles, privileges, password policies and hosts.
- **Execution Environment:** technologies, data servers, contexts, generic actions, load balanced agents, physical schemas and agents.

To export the topology/security:

1. Select **Export...** from the Designer, Topology, Security or Operator Navigator toolbar menu.
2. In the Export Selection dialog, select one of the following:
 - **Export the Topology**
 - **Export the Logical Topology**
 - **Export the Security Settings**
 - **Export the Execution Environment**
3. Click **OK**.
4. In the Export dialog, specify the export parameters as indicated in [Table 23–3, "Object Export Parameters"](#).

The topology and security settings are either exported as .xml files directly into the directory, or as a zip file containing .xml files. If you want to generate a zip file, you need to select **Export to zip file** and enter the name of the zip file in the **Zip File Name** field.

5. Click **OK**.

The export files are created in the specified export directory.

Importing the Topology and Security Settings

To import a topology export:

1. Select **Import...** from the Designer, Topology, Security or Operator Navigator toolbar menu.
2. In the Import Selection dialog, select one of the following:
 - **Import the Topology**
 - **Import the Logical Topology**
 - **Import Security Settings**
 - **Import the Execution Environment**
3. Click **OK**.
4. In the Import dialog:
 1. Select the **Import Mode**. Refer to ["Import Modes"](#) on page 23-3 for more information.
 2. Select whether to import the topology export from a **Folder** or a **Zip File**.
 3. Enter the file import directory.
5. Click **OK**.
6. If prompted, enter the Export Key used when this object was exported. If you do not enter an Export Key, any encrypted sensitive (cipher) data will be stripped from the imported object. For more information about the Export Key, see: ["Export Keys"](#) on page 23-2.

The specified files are imported into the master repository.

23.3.3 Exporting and Importing a Work Repository

Importing or exporting a work repository allows you to transfer all work repository objects from one repository to another.

Exporting a Work Repository

To export a work repository:

1. Select **Export...** from the Designer, Topology, Security or Operator Navigator toolbar menu.
2. In the Export Selection dialog, select **Export the Work Repository**.
3. Click **OK**.
4. In the Export dialog, set the Export parameters as indicated in [Table 23-3, "Object Export Parameters"](#).

The work repository with its models and projects are either exported as `.xml` files directly into the directory, or as a zip file containing `.xml` files. If you want to generate a zip file, you need to select **Export to zip file** and enter the name of the zip file in the **Zip File Name** field

5. Click **OK**.

The export files are created in the specified export directory.

Importing a Work Repository

To import a work repository:

1. Select **Import...** from the Designer, Topology, Security or Operator Navigator toolbar menu.
2. In the Import Selection dialog, select **Import the Work Repository**.
3. Click **OK**.
4. In the Import dialog:
 1. Select the **Import mode**. Refer to ["Import Modes"](#) on page 23-3 for more information.
 2. Select whether to import the work repository from a **Folder** or a **Zip File**.
 3. Enter the file import directory.
5. Click **OK**.
6. If prompted, enter the Export Key used when this object was exported. If you do not enter an Export Key, any encrypted sensitive (cipher) data will be stripped from the imported object. For more information about the Export Key, see: ["Export Keys"](#) on page 23-2.

The specified files are imported into the work repository.

23.4 Exporting the Technical Environment

This feature produces a comma separated (`.csv`) file in the directory of your choice, containing the details of the technical environment. The export includes a description of your work environment. It contains info about the ODI version being used, master and work repositories, and information about agents and technologies. This export may be required for troubleshooting or support issues.

You can customize the format of this file.

To produce the technical environment file:

1. Select **Export...** from the Designer, Topology, Security or Operator Navigator toolbar menu.

2. In the Export Selection dialog, select **Export the Technical Environment**.
3. Click **OK**.
4. In the Technical environment dialog, specify the export parameters as indicated in [Table 23-7](#):

Table 23-7 Technical Environment Export Parameters

Properties	Description
Export Directory	Directory in which the export file will be created.
File Name	Name of the .cvs export file
Advanced options	This set of options allow to parameterize the XML output file format. It is recommended that you leave the default values.
Character Set	Encoding specified in the export file. Parameter encoding in the XML file header. <?xml version="1.0" encoding="ISO-8859-1"?>
Field codes	The first field of each record produced contains a code identifying the kind of information present on the row. You can customize these codes as necessary. <ul style="list-style-type: none"> ■ Oracle Data Integrator Information Record Code: Code used to identify rows that describe the current version of Oracle Data Integrator and the current user. This code is used in the first field of the record. ■ Master, Work, Agent, and Technology Record Code: Code for rows containing information about the master repository, the work repositories, the running agents, or the the data servers, their version, etc.
Record Separator and Field Separator	These separators define the characters used to separate records (lines) in the file, and fields within one record.

5. Click **OK**.

23.5 Exporting and Importing the Log

You can export and import log data for archiving purposes. See "Exporting and Importing Log Data" in *Administering Oracle Data Integrator* for more information.

