Oracle® Fusion Middleware

Developer's Guide for Oracle Data Integrator

12*c* (12.1.2)

E39359-03

January 2014

ORACLE®

Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator, 12*c* (12.1.2)

E39359-03

Primary Authors: Laura Hofman Miquel, Joshua Stanley

Contributing Authors: Alex Kotopoulis, Michael Reiche, Jon Patt, Alex Prazma, Gail Risdal

Contributors: David Allan, Linda Bittarelli, Sophia Chen, Pratima Chennupati, Victor Chow, Sowmya Dhandapani, Daniel Gallagher, Gary Hostetler, Kevin Hwang, Aslam Khan, Sebu T. Koleth, Christian Kurz, Venkata Lakkaraju, Thomas Lau, Deekshit Mantampady, Kevin McDonough, Luda Mogilevich, Ale Paez, Suresh Pendap, Sandrine Riley, Julien Testut, Sachin Thatte, Julie Timmons, Jay Turner, Vikas Varma, Robert Velisar, Winnie Wan, Geoff Watters, Jennifer Waywell

Contents

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 and functionalities 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 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 Oracle resources:

- *Installing and Configuring Oracle Data Integrator*
- *Upgrading Oracle Data Integrator*
- *Connectivity and Knowledge Modules Guide for Oracle Data Integrator*

- *Application Adapters Guide for Oracle Data Integrator*
- *Knowledge Module Developer's Guide for Oracle Data Integrator*
- *Release Notes 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.
italic	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 document describes the new and enhanced features introduced with Oracle Data Integrator 12*c* (12.1.2).

New Features in Oracle Data Integrator 12*c* (12.1.2)

Oracle Data Integrator 12*c* (12.1.2) introduces the following enhancements:

- Declarative Flow-Based User Interface
- Reusable Mappings
- Multiple Target Support
- Step-by-Step Debugger
- Runtime Performance Enhancements
- Oracle GoldenGate Integration Improvements
- Standalone Agent Management with WebLogic Management Framework
- Integration with OPSS Enterprise Roles
- XML Improvements
- Oracle Warehouse Builder Integration
- Unique Repository IDs
- Oracle Warehouse Builder to Oracle Data Integrator Migration Utility

Declarative Flow-Based User Interface

The new declarative flow-based user interface combines the simplicity and ease-of-use of the declarative approach with the flexibility and extensibility of configurable flows. Mappings (the successor of the Interface concept in Oracle Data Integrator 11*g*) connect sources to targets through a flow of components such as Join, Filter, Aggregate, Set, Split, and so on.

Reusable Mappings

Reusable Mappings can be used to encapsulate flow sections that can then be reused in multiple mappings. A reusable mapping can have input and output signatures to connect to an enclosing flow; it can also contain sources and targets that are encapsulated inside the reusable mapping.

Multiple Target Support

A mapping can now load multiple targets as part of a single flow. The order of target loading can be specified, and the Split component can be optionally used to route rows into different targets, based on one or several conditions.

Step-by-Step Debugger

Mappings, Packages, Procedures, and Scenarios can now be debugged in a step-by-step debugger. Users can manually traverse task execution within these objects and set breakpoints to interrupt execution at pre-defined locations. Values of variables can be introspected and changed during a debugging session, and data of underlying sources and targets can be queried, including the content of uncommitted transactions.

Runtime Performance Enhancements

The runtime execution has been improved to enhance performance. Various changes have been made to reduce overhead of session execution, including the introduction of blueprints, which are cached execution plans for sessions.

Performance is improved by loading sources in parallel into the staging area. Parallelism of loads can be customized in the physical view of a map.

Users also have the option to use unique names for temporary database objects, allowing parallel execution of the same mapping.

Oracle GoldenGate Integration Improvements

The integration of Oracle GoldenGate as a source for the Change Data Capture (CDC) framework has been improved in the following areas:

- Oracle GoldenGate source and target systems are now configured as data servers in Topology. Extract and replicate processes are represented by physical and logical schemas. This representation in Topology allows separate configuration of multiple contexts, following the general context philosophy.
- Most Oracle GoldenGate parameters can now be added to extract and replicate processes in the physical schema configuration. The UI provides support for selecting parameters from lists. This minimizes the need for the modification of Oracle GoldenGate parameter files after generation.
- A single mapping can now be used for journalized CDC load and bulk load of a target. This is enabled by the Oracle GoldenGate JKM using the source model as opposed to the Oracle GoldenGate replication target, as well as configuration of journalizing in mapping as part of a deployment specification. Multiple deployment specifications can be used in a single mapping for journalized load and bulk load.
- Oracle GoldenGate parameter files can now be automatically deployed and started to source and target Oracle GoldenGate instances through the JAgent technology.

Standalone Agent Management with WebLogic Management Framework

Oracle Data Integrator standalone agents are now managed through the WebLogic Management Framework. This has the following advantages:

- UI-driven configuration through Configuration Wizard
- Multiple configurations can be maintained in separate domains
- Node Manager can be used to control and automatically restart agents

Integration with OPSS Enterprise Roles

Oracle Data Integrator can now use the authorization model in Oracle Platform Security Services (OPSS) to control access to resources. Enterprise roles can be mapped into Oracle Data Integrator roles to authorize enterprise users across different tools.

XML Improvements

The following XML Schema constructs are now supported:

- list and union - List or union-based elements are mapped into VARCHAR columns.
- substitutionGroup - Elements based on substitution groups create a table each for all types of the substitution group.
- Mixed content - Elements with mixed content map into a VARCHAR column that contains text and markup content of the element.
- Annotation - Content of XML schema annotations are stored in the table metadata.

Oracle Warehouse Builder Integration

Oracle Warehouse Builder (OWB) jobs can now be executed in Oracle Data Integrator through the `OdiStartOwbJob` tool. The OWB repository is configured as a data server in Topology. All the details of the OWB job execution are displayed as a session in the Operator tree. For more information about this feature, see "OdiStartOwbJob" on page A-76.

Unique Repository IDs

Master and Work Repositories now use unique IDs following the GUID convention. This avoids collisions during import of artifacts and allows for easier management and consolidation of multiple repositories in an organization.

Oracle Warehouse Builder to Oracle Data Integrator Migration Utility

ODI 12c supports an easier mapping between Oracle Warehouse Builder 11gR2 (11.2.0.4) concepts and objects and their ODI 12c (12.1.2) counterparts. A migration utility is provided that automatically translates many OWB objects and mappings into their ODI equivalents.

The migration utility requires ODI patch 17053768 and the ODI 12.1.2.0.1 bundle patch (patch number 17836908), and OWB patch 17830453. For more information about the migration utility, see *Migrating from Oracle Warehouse Builder to Oracle Data Integrator*.

Part I

Understanding 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:

1 Introduction to Oracle Data Integrator

This chapter introduces Oracle Data Integrator.

This chapter includes the following sections:

Introduction to Data Integration with Oracle Data Integrator

Data Integration ensures that information is timely, accurate, and consistent across complex systems. This section provides an introduction to data integration and describes how Oracle Data Integrator provides support for Data Integration.

Data Integration

Integrating data and applications throughout the enterprise, and presenting them in a unified view is a complex proposition. Not only are there broad disparities in technologies, data structures, and application functionality, but there are also fundamental differences in integration architectures. Some integration needs are Data Oriented, especially those involving large data volumes. Other integration projects lend themselves to an Event Driven Architecture (EDA) or a Service Oriented Architecture (SOA), for asynchronous or synchronous integration.

Data Integration ensures that information is timely, accurate, and consistent across complex systems. Although it is still frequently referred as Extract-Load-Transform (ETL) - Data Integration was initially considered as the architecture used for loading Enterprise Data Warehouse systems - data integration now includes data movement, data synchronization, data quality, data management, and data services.

Oracle Data Integrator

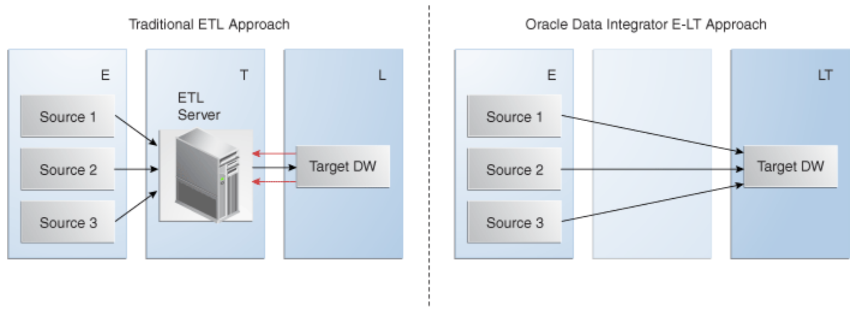
Oracle Data Integrator provides a fully unified solution for building, deploying, and managing complex data warehouses or as part of data-centric architectures in a SOA or business intelligence environment. In addition, it combines all the elements of data integration—data movement, data synchronization, data quality, data management, and data services—to ensure that information is timely, accurate, and consistent across complex systems.

Oracle Data Integrator (ODI) features an active integration platform that includes all styles of data integration: data-based, event-based and service-based. ODI unifies silos of integration by transforming large volumes of data efficiently, processing events in real time through its advanced Changed Data Capture (CDC) framework, and providing data services to the Oracle SOA Suite. It also provides robust data integrity control features, assuring the consistency and correctness of data. With powerful core differentiators - heterogeneous E-LT, Declarative Design and Knowledge Modules - Oracle Data Integrator meets the performance, flexibility, productivity, modularity and hot-pluggability requirements of an integration platform.

E-LT

Traditional ETL tools operate by first *E*xtracting the data from various sources, *T*ransforming the data in a proprietary, middle-tier ETL engine that is used as the staging area, and then *L*oading the transformed data into the target data warehouse or integration server. Hence the term *ETL* represents both the names and the order of the operations performed, as shown in Figure 1–1.

Figure 1–1 Traditional ETL versus ODI E-LT



The data transformation step of the ETL process is by far the most compute-intensive, and is performed entirely by the proprietary ETL engine on a dedicated server. The ETL engine performs data transformations (and sometimes data quality checks) on a row-by-row basis, and hence, can easily become the bottleneck in the overall process. In addition, the data must be moved over the network twice – once between the sources and the ETL server, and again between the ETL server and the target data warehouse. Moreover, if one wants to ensure referential integrity by comparing data flow references against values from the target data warehouse, the referenced data must be downloaded from the target to the engine, thus further increasing network traffic, download time, and leading to additional performance issues.

In response to the issues raised by ETL architectures, a new architecture has emerged, which in many ways incorporates the best aspects of manual coding and automated code-generation approaches. Known as *E-LT*, this new approach changes where and how data transformation takes place, and leverages existing developer skills, RDBMS engines and server hardware to the greatest extent possible. In essence, E-LT moves the data transformation step to the target RDBMS, changing the order of operations to: Extract the data from the source tables, Load the tables into the destination server, and then Transform the data on the target RDBMS using native SQL operators. Note, with E-LT there is no need for a middle-tier engine or server as shown in Figure 1–1.

Oracle Data Integrator supports both ETL- and E-LT-Style data integration. See "Designing E-LT and ETL-Style Mappings" on page 11-33 for more information.

Oracle Data Integrator Concepts

This section provides an introduction to the main concepts of Oracle Data Integrator.

Introduction to Declarative Design

To design an integration process with conventional ETL systems, a developer needs to design each step of the process: Consider, for example, a common case in which sales figures must be summed over time for different customer age groups. The sales data comes from a sales management database, and age groups are described in an age distribution file. In order to combine these sources then insert and update appropriate records in the customer statistics systems, you must design each step, which includes:

1. Load the customer sales data in the engine
2. Load the age distribution file in the engine
3. Perform a lookup between the customer sales data and the age distribution data
4. Aggregate the customer sales grouped by age distribution
5. Load the target sales statistics data into the engine
6. Determine what needs to be inserted or updated by comparing aggregated information with the data from the statistics system
7. Insert new records into the target
8. Update existing records into the target

This method requires specialized skills, depending on the steps that need to be designed. It also requires significant efforts in development, because even repetitive succession of tasks, such as managing inserts/updates in a target, need to be developed into each task. Finally, with this method, maintenance requires significant effort. Changing the integration process requires a clear understanding of what the process does as well as the knowledge of how it is done. With the conventional ETL method of design, the logical and technical aspects of the integration are intertwined. Declarative Design is a design method that focuses on "What" to do (the Declarative Rules) rather than "How" to do it (the Process). In our example, "What" the process does is:

- Relate the customer age from the sales application to the age groups from the statistical file
- Aggregate customer sales by age groups to load sales statistics

"How" this is done, that is the underlying technical aspects or technical strategies for performing this integration task – such as creating temporary data structures or calling loaders – is clearly separated from the declarative rules.

Declarative Design in Oracle Data Integrator uses the well known relational paradigm to declare in the form of a mapping the declarative rules for a data integration task, which includes designation of sources, targets, and transformations.

Declarative rules often apply to metadata to transform data and are usually described in natural language by business users. In a typical data integration project (such as a Data Warehouse project), these rules are defined during the specification phase in documents written by business analysts in conjunction with project managers. They can very often be implemented using SQL expressions, provided that the metadata they refer to is known and qualified in a metadata repository.

The four major types of Declarative Rules are mappings, joins, filters and constraints:

- A mapping is a business rule implemented as an SQL expression. It is a transformation rule that maps source attributes (or fields) onto one of the target attributes. It can be executed by a relational database server at run-time. This server can be the source server (when possible), a middle tier server or the target server.
- A join operation links records in several data sets, such as tables or files. Joins are used to link multiple sources. A join is implemented as an SQL expression linking the attributes (fields) of two or more data sets. Joins can be defined regardless of the physical location of the source data sets involved. For example, a JMS queue can be joined to an Oracle table. Depending on the technology performing the join, it can be expressed as an inner join, right outer join, left outer join and full outer join.
- A filter is an expression applied to source data sets attributes. Only the records matching this filter are processed by the data flow.
- A constraint is an object that defines the rules enforced on data sets' data. A constraint ensures the validity of the data in a given data set and the integrity of the data of a model. Constraints on the target are used to check the validity of the data before integration in the target.

Table 1–1 gives examples of declarative rules.

Table 1–1 Examples of declarative rules

Declarative Rule	Type	SQL Expression
Sum of all amounts or items sold during October 2005 multiplied by the item price	Mapping	`SUM( CASE WHEN SALES.YEARMONTH=200510 THEN SALES.AMOUNT*product.item_PRICE ELSE 0 END )`
Products that start with 'CPU' and that belong to the hardware category	Filter	`Upper(PRODUCT.PRODUCT_NAME)like 'CPU%' And PRODCUT.CATEGORY = 'HARDWARE'`
Customers with their orders and order lines	Join	`CUSTOMER.CUSTOMER_ID = ORDER.ORDER_ID And ORDER.ORDER_ID = ORDER_LINE.ORDER_ID`
Reject duplicate customer names	Unique Key Constraint	`Unique key (CUSTOMER_NAME)`
Reject orders with a link to an non-existent customer	Reference Constraint	`Foreign key on ORDERS(CUSTOMER_ID) references CUSTOMER(CUSTOMER_ID)`

Introduction to Knowledge Modules

Knowledge Modules (KM) implement "how" the integration processes occur. Each Knowledge Module type refers to a specific integration task:

- Reverse-engineering metadata from the heterogeneous systems for Oracle Data Integrator (RKM). Refer to Chapter 5, "Creating and Using Data Models and Datastores" for more information on how to use the RKM.
- Handling Changed Data Capture (CDC) on a given system (JKM). Refer to Chapter 6, "Using Journalizing" for more information on how to use the Journalizing Knowledge Modules.

- Loading data from one system to another, using system-optimized methods (LKM). These KMs are used in mappings. See Chapter 11, "Creating and Using Mappings" for more information on how to use the Loading Knowledge Modules.
- Integrating data in a target system, using specific strategies (insert/update, slowly changing dimensions) (IKM). These KMs are used in mappings. See Chapter 11, "Creating and Using Mappings" for more information on how to use the Integration Knowledge Modules.
- Controlling Data Integrity on the data flow (CKM). These KMs are used in data model's static check and mappings flow checks. See Chapter 5, "Creating and Using Data Models and Datastores" and Chapter 11, "Creating and Using Mappings" for more information on how to use the Check Knowledge Modules.
- Exposing data in the form of web services (SKM). Refer to Chapter 8, "Creating and Using Data Services" for more information on how to use the Service Knowledge Modules.

A Knowledge Module is a code template for a given integration task. This code is independent of the Declarative Rules that need to be processed. At design-time, a developer creates the Declarative Rules describing integration processes. These Declarative Rules are merged with the Knowledge Module to generate code ready for runtime. At runtime, Oracle Data Integrator sends this code for execution to the source and target systems it leverages in the E-LT architecture for running the process.

Knowledge Modules cover a wide range of technologies and techniques. Knowledge Modules provide additional flexibility by giving users access to the most-appropriate or finely tuned solution for a specific task in a given situation. For example, to transfer data from one DBMS to another, a developer can use any of several methods depending on the situation:

- The DBMS loaders (Oracle's SQL*Loader, Microsoft SQL Server's BCP, Teradata TPump) can dump data from the source engine to a file then load this file to the target engine
- The database link features (Oracle Database Links, Microsoft SQL Server's Linked Servers) can transfer data directly between servers

These technical strategies amongst others corresponds to Knowledge Modules tuned to exploit native capabilities of given platforms.

Knowledge modules are also fully extensible. Their code is opened and can be edited through a graphical user by technical experts willing to implement new integration methods or best practices (for example, for higher performance or to comply with regulations and corporate standards). Without having the skill of the technical experts, developers can use these custom Knowledge Modules in the integration processes.

For more information on Knowledge Modules, refer to the *Connectivity and Modules Guide for Oracle Data Integrator* and the *Knowledge Module Developer's Guide for Oracle Data Integrator*.

Introduction to Mappings

A mapping is an Oracle Data Integrator object stored that enables the loading of target datastores with data transformed from source datastores, based on declarative rules implemented as joins, filters and constraints.

A mapping also references the Knowledge Modules (code templates) that will be used to generate the integration process.

Datastores

A datastore is a data structure that can be used as a source or a target in a mapping. It can be:

- a table stored in a relational database
- an ASCII or EBCDIC file (delimited, or fixed length)
- a node from a XML file
- a JMS topic or queue from a Message Oriented Middleware
- a node from a enterprise directory
- an API that returns data in the form of an array of records

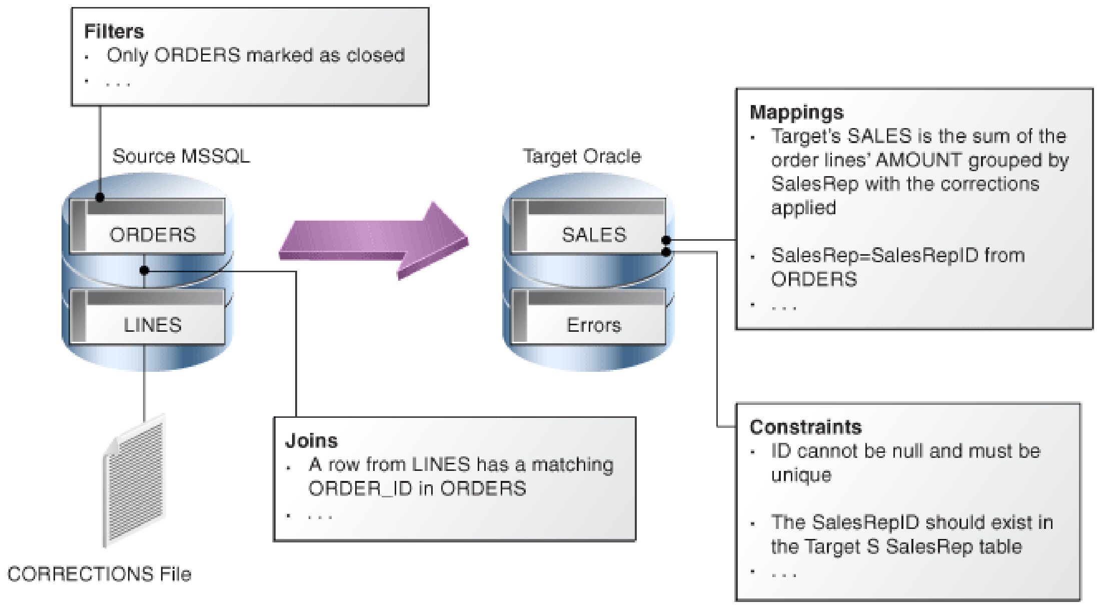
Regardless of the underlying technology, all data sources appear in Oracle Data Integrator in the form of datastores that can be manipulated and integrated in the same way. The datastores are grouped into data models. These models contain all the declarative rules –metadata - attached to datastores such as constraints.

Declarative Rules

The declarative rules that make up a mapping can be expressed in human language, as shown in the following example: Data is coming from two Microsoft SQL Server tables (ORDERS joined to ORDER_LINES) and is combined with data from the CORRECTIONS file. The target SALES Oracle table must match some constraints such as the uniqueness of the ID column and valid reference to the SALES_REP table.

Data must be transformed and aggregated according to some mappings expressed in human language as shown in Figure 1–2.

Figure 1–2 Example of a business problem


Filters
• Only ORDERS marked as closed
• . . .
Source MSSQL
ORDERS
LINES
CORRECTIONS File
Target Oracle
SALES
Errors
Mappings
• Target's SALES is the sum of the order lines' AMOUNT grouped by SalesRep with the corrections applied
• SalesRep=SalesRepID from ORDERS
• . . .
Joins
• A row from LINES has a matching ORDER_ID in ORDERS
• . . .
Constraints
• ID cannot be null and must be unique
• The SalesRepID should exist in the Target S SalesRep table
• . . .

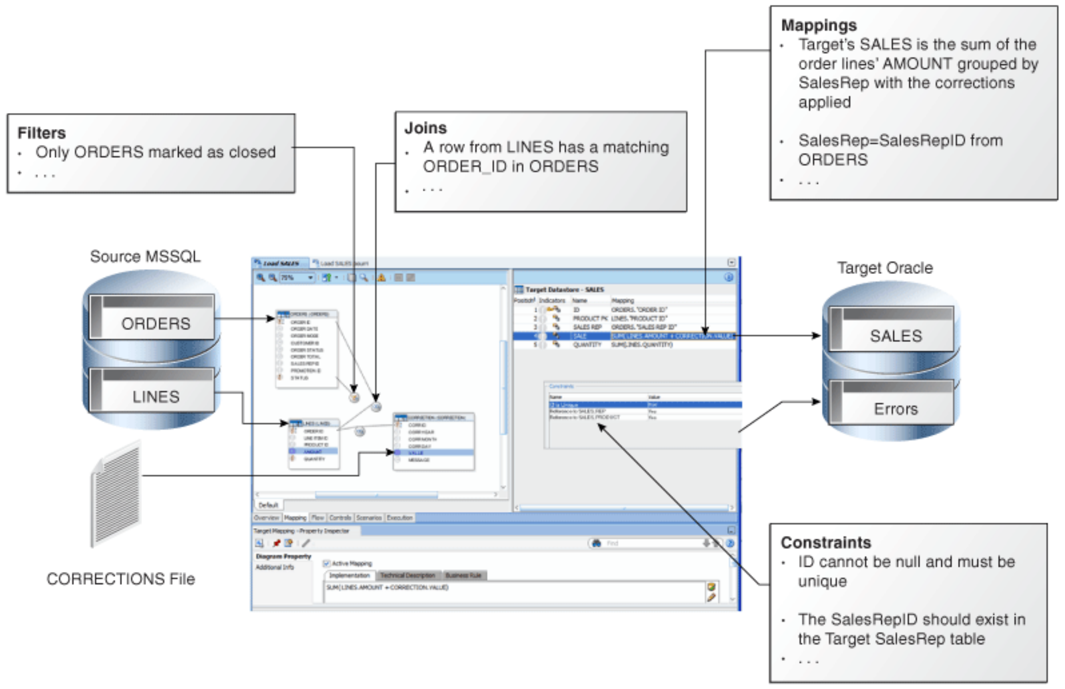
Translating these business rules from natural language to SQL expressions is usually straightforward. In our example, the rules that appear in Figure 1–2 could be translated as shown in Table 1–2.

Table 1–2 Business rules translated

Type	Rule	SQL Expression/Constraint
Filter	Only ORDERS marked as closed	`ORDERS.STATUS = 'CLOSED'`
Join	A row from LINES has a matching ORDER_ID in ORDERS	`ORDERS.ORDER_ID = LINES.ORDER_ID`
Mapping	Target's SALES is the sum of the order lines' AMOUNT grouped by sales rep, with the corrections applied	`SUM(LINES.AMOUNT + CORRECTIONS.VALUE)`
Mapping	Sales Rep = Sales Rep ID from ORDERS	`ORDERS.SALES_REP_ID`
Constraint	ID must not be null	ID is set to `not null` in the data model
Constraint	ID must be unique	A unique key is added to the data model with (ID) as set of columns
Constraint	The Sales Rep ID should exist in the Target SalesRep table	A reference (foreign key) is added in the data model on `SALES.SALES_REP = SALES_REP.SALES_REP_ID`

Implementing this business problem using Oracle Data Integrator is a very easy and straightforward exercise. It is done by simply translating the business rules into a mapping. Every business rule remains accessible from the mapping's diagram, as shown in Figure 1–3.

Figure 1–3 Implementation using Oracle Data Integrator


Filters
• Only ORDERS marked as closed
• . . .
Joins
• A row from LINES has a matching ORDER_ID in ORDERS
• . . .
Mappings
• Target's SALES is the sum of the order lines' AMOUNT grouped by SalesRep with the corrections applied
• SalesRep=SalesRepID from ORDERS
• . . .
Source MSSQL
ORDERS
LINES
CORRECTIONS File
Target Oracle
SALES
Errors
Constraints
• ID cannot be null and must be unique
• The SalesRepID should exist in the Target SalesRep table
• . . .

Data Flow

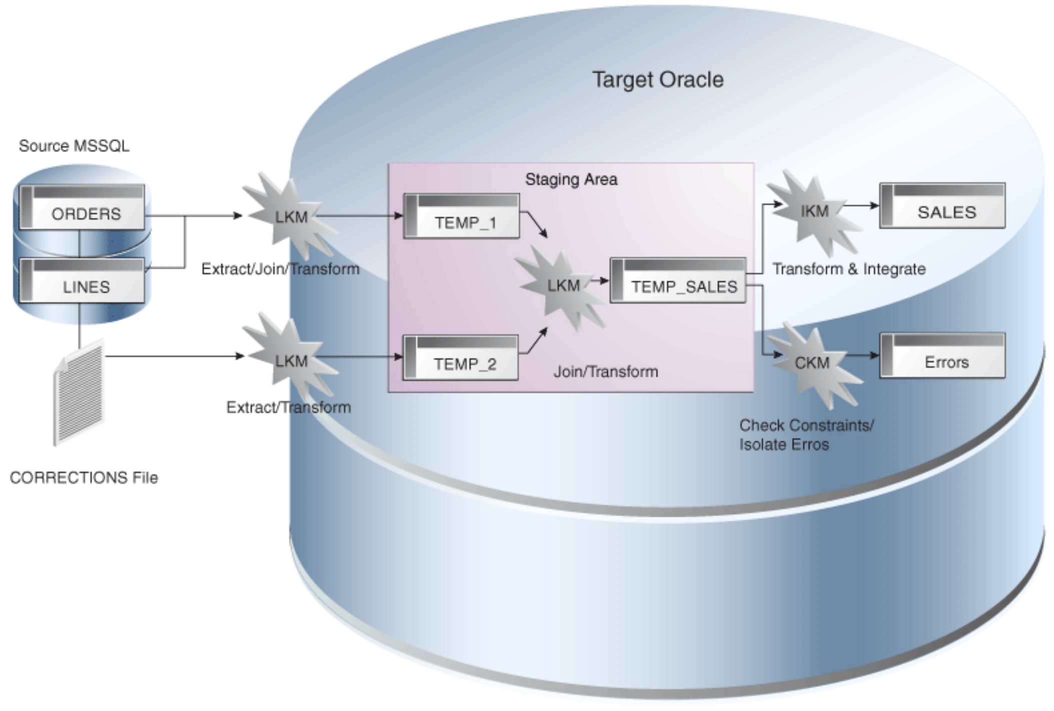
Business rules defined in the mapping are automatically converted into a data flow that will carry out the joins filters, mappings, and constraints from source data to target tables.

By default, Oracle Data Integrator will use the Target RDBMS as a staging area for loading source data into temporary tables and applying all the required mappings, staging filters, joins and constraints. The staging area is a separate area in the RDBMS (a user/database) where Oracle Data Integrator creates its temporary objects and executes some of the rules (mapping, joins, final filters, aggregations etc.). When performing the operations this way, Oracle Data Integrator behaves like an E-LT as it first extracts and loads the temporary tables and then finishes the transformations in the target RDBMS.

In some particular cases, when source volumes are small (less than 500,000 records), this staging area can be located in memory in Oracle Data Integrator's in-memory relational database – In-Memory Engine. Oracle Data Integrator would then behave like a traditional ETL tool.

Figure 1–4 shows the data flow automatically generated by Oracle Data Integrator to load the final SALES table. The business rules will be transformed into code by the Knowledge Modules (KM). The code produced will generate several steps. Some of these steps will extract and load the data from the sources to the staging area (Loading Knowledge Modules - LKM). Others will transform and integrate the data from the staging area to the target table (Integration Knowledge Module - IKM). To ensure data quality, the Check Knowledge Module (CKM) will apply the user defined constraints to the staging data to isolate erroneous records in the Errors table.

Figure 1–4 Oracle Data Integrator Knowledge Modules in action



Oracle Data Integrator Knowledge Modules contain the actual code that will be executed by the various servers of the infrastructure. Some of the code contained in the Knowledge Modules is generic. It makes calls to the Oracle Data Integrator Substitution API that will be bound at run-time to the business-rules and generates the final code that will be executed.

At design time, declarative rules are defined in the mappings and Knowledge Modules are only selected and configured.

At run-time, code is generated and every Oracle Data Integrator API call in the Knowledge Modules (enclosed by <% and %>) is replaced with its corresponding object name or expression, with respect to the metadata provided in the Repository. The generated code is orchestrated by Oracle Data Integrator run-time component - the Agent – on the source and target systems to make them perform the processing, as defined in the E-LT approach.

Refer to Chapter 11, "Creating and Using Mappings" for more information on how to work with mappings.

Typical ODI Integration Projects

Oracle Data Integrator provides a wide range of integration features. This section introduces the most typical ODI Integration Projects.

Batch Oriented Integration

ODI is a comprehensive data integration platform with a built-in connectivity to all major databases, data warehouse and analytic applications providing high-volume and high-performance batch integration.

The main goal of a data warehouse is to consolidate and deliver accurate indicators to business users to help them make decisions regarding their everyday business. A typical project is composed of several steps and milestones. Some of these are:

- Defining business needs (Key Indicators)
- Identifying source data that concerns key indicators; specifying business rules to transform source information into key indicators
- Modeling the data structure of the target warehouse to store the key indicators
- Populating the indicators by implementing business rules
- Measuring the overall accuracy of the data by setting up data quality rules
- Developing reports on key indicators
- Making key indicators and metadata available to business users through adhoc query tools or predefined reports
- Measuring business users' satisfaction and adding/modifying key indicators

Oracle Data Integrator will help you cover most of these steps, from source data investigation to metadata lineage, and through loading and data quality audit. With its repository, ODI will centralize the specification and development efforts and provide a unique architecture on which the project can rely to succeed.

Scheduling and Operating Scenarios

Scheduling and operating scenarios is usually done in the Test and Production environments in separate Work Repositories. Any scenario can be scheduled by an ODI Agent or by any external scheduler, as scenarios can be invoked by an operating system command.

When scenarios are running in production, agents generate execution logs in an ODI Work Repository. These logs can be monitored either through the Operator Navigator or through any web browser when Oracle Data Integrator Console is setup. Failing jobs can be restarted and ad-hoc tasks submitted for execution.

E-LT

ODI uses a unique E-LT architecture that leverages the power of existing RDBMS engines by generating native SQL and bulk loader control scripts to execute all transformations.

Event Oriented Integration

Capturing events from a Message Oriented Middleware or an Enterprise Service Bus has become a common task in integrating applications in a real-time environment. Applications and business processes generate messages for several subscribers, or they consume messages from the messaging infrastructure.

Oracle Data Integrator includes technology to support message-based integration and that complies with the Java Message Services (JMS) standard. For example, a transformation job within Oracle Data Integrator can subscribe and source messages from any message queue or topic. Messages are captured and transformed in real time and then written to the target systems.

Other use cases of this type of integration might require capturing changes at the database level. Oracle Data Integrator Changed Data Capture (CDC) capability identifies and captures inserted, updated, or deleted data from the source and makes it available for integration processes.

ODI provides two methods for tracking changes from source datastores to the CDC framework: triggers and RDBMS log mining. The first method can be deployed on most RDBMS that implement database triggers. This method is optimized to minimize overhead on the source systems. For example, changed data captured by the trigger is not duplicated, minimizing the number of input/output operations, which slow down source systems. The second method involves mining the RDBMS logs—the internal change history of the database engine. This has little impact on the system's transactional performance and is supported for Oracle (through the Log Miner feature).

The CDC framework used to manage changes, based on Knowledge Modules, is generic and open, so the change-tracking method can be customized. Any third-party change provider can be used to load the framework with changes.

Changes frequently involve several data sources at the same time. For example, when an order is created, updated, or deleted, both the orders table and the order lines table are involved. When processing a new order line, it is important that the new order, to which the line is related, is taken into account too. ODI provides a mode of change tracking called Consistent Set CDC. This mode allows for processing sets of changes for which data consistency is guaranteed.

For example, incoming orders can be detected at the database level using CDC. These new orders are enriched and transformed by ODI before being posted to the appropriate message queue or topic. Other applications such as Oracle BPEL or Oracle Business Activity Monitoring can subscribe to these messages, and the incoming events will trigger the appropriate business processes.

For more information on how to use the CDC framework in ODI, refer to Chapter 6, "Using Journalizing".

Service-Oriented Architecture

Oracle Data Integrator can be integrated seamlessly in a Service Oriented Architecture (SOA) in several ways:

Data Services are specialized Web services that provide access to data stored in database tables. Coupled with the Changed Data Capture capability, data services can also provide access to the changed records for a given subscriber. Data services are automatically generated by Oracle Data Integrator and deployed as Web services to a Web container, usually a Java application server. For more information on how to set up, generate and deploy data services, refer to Chapter 8, "Creating and Using Data Services".

Oracle Data Integrator can also expose its transformation processes as Web services to enable applications to use them as integration services. For example, a LOAD_SALES batch process used to update the CRM application can be triggered as a Web service from any service-compliant application, such as Oracle BPEL, Oracle Enterprise Service Bus, or Oracle Business Activity Monitoring. Transformations developed using ODI can therefore participate in the broader Service Oriented Architecture initiative.

Third-party Web services can be invoked as part of an ODI workflow and used as part of the data integration processes. Requests are generated on the fly and responses processed through regular transformations. Suppose, for example, that your company subscribed to a third-party service that exposes daily currency exchange rates as a Web service. If you want this data to update your multiple currency data warehouse, ODI automates this task with a minimum of effort. You would simply invoke the Web service from your data warehouse workflow and perform any appropriate transformation to the incoming data to make it fit a specific format. For more information on how to use web services in ODI, refer to Chapter 16, "Using Web Services".

Data Quality with ODI

With an approach based on declarative rules, Oracle Data Integrator is the most appropriate tool to help you build a data quality framework to track data inconsistencies.

Oracle Data Integrator uses declarative data integrity rules defined in its centralized metadata repository. These rules are applied to application data to guarantee the integrity and consistency of enterprise information. The Data Integrity benefits add to the overall Data Quality initiative and facilitate integration with existing and future business processes addressing this particular need.

Oracle Data Integrator automatically retrieves existing rules defined at the data level (such as database constraints) by a reverse-engineering process. ODI also allows developers to define additional, user-defined declarative rules that may be inferred from data discovery and profiling within ODI, and immediately checked.

Oracle Data Integrator provides a built-in framework to check the quality of your data in two ways:

- Check data in your data servers, to validate that this data does not violate any of the rules declared on the datastores in Oracle Data Integrator. This data quality check is called a static check and is performed on data models and datastores. This type of check allows you to profile the quality of the data against rules that are not enforced by their storage technology.
- Check data while it is moved and transformed by a mapping, in a flow check that checks the data flow against the rules defined on the target datastore. With such a check, correct data can be integrated into the target datastore while incorrect data is automatically moved into error tables.

Both static and flow checks are using the constraints that are defined in the datastores and data models, and both use the Check Knowledge Modules (CKMs). For more information refer to "Flow Control and Static Control" on page 11-31.

Managing Environments

Integration projects exist in different environments during their lifecycle (development, test, product) and may even run in different environments in production (multiple site deployment). Oracle Data Integrator makes easier the definition and maintenance of these environments, as well as the lifecycle of the project across these environments using the Topology

The Topology describes the physical and logical architecture of your Information System. It gives you a very flexible way of managing different servers, environments and agents. All the information of the Topology is stored in the master repository and is therefore centralized for an optimized administration. All the objects manipulated within Work Repositories refer to the Topology. That's why it is the most important starting point when defining and planning your architecture.

The Topology is composed of data servers, physical and logical schemas, and contexts.

Data servers describe connections to your actual physical application servers and databases. They can represent for example:

- An Oracle Instance
- An IBM DB2 Database
- A Microsoft SQL Server Instance
- A File System
- An XML File
- and so forth.

At runtime, Oracle Data Integrator uses the connection information you have described to connect to the servers.

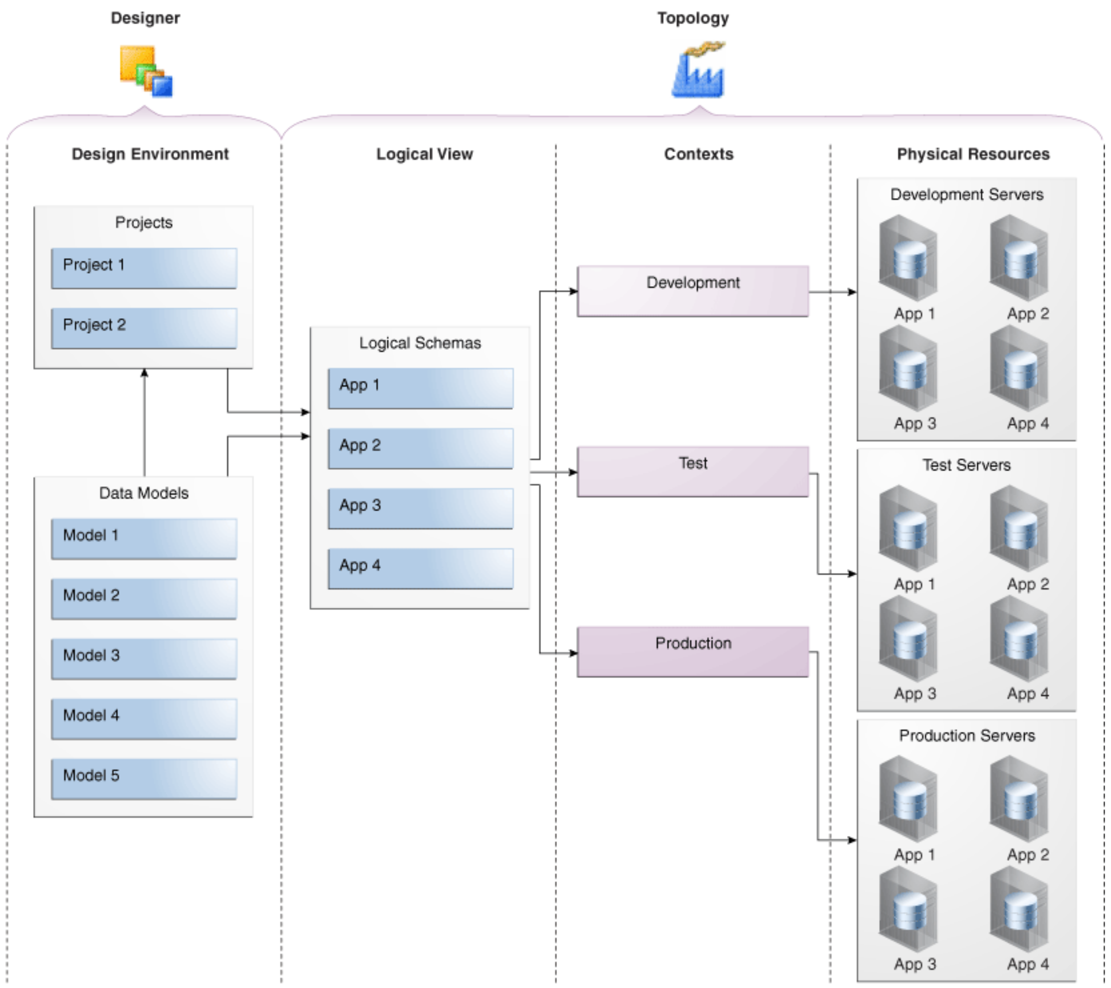
Physical schemas indicate the physical location of the datastores (tables, files, topics, queues) inside a data server. All the physical schemas that need to be accessed have to be registered under their corresponding data server, physical schemas are used to prefix object names and access them with their qualified names. When creating a physical schema, you need to specify a temporary, or work schema that will store temporary or permanent object needed at runtime.

A logical schema is an alias that allows a unique name to be given to all the physical schemas containing the same datastore structures. The aim of the logical schema is to ensure the portability of procedures and models on different design-time and run-time environments.

A Context represents one of these environments. Contexts are used to group physical resources belonging to the same environment.

Typical projects will have separate environments for Development, Test and Production. Some projects will even have several duplicated Test or Production environments. For example, you may have several production contexts for subsidiaries running their own production systems (Production New York, Production Boston, and so forth). There is obviously a difference between the logical view of the information system and its physical implementation as described in Figure 1–5.

Figure 1–5 Logical and Physical View of the Infrastructure



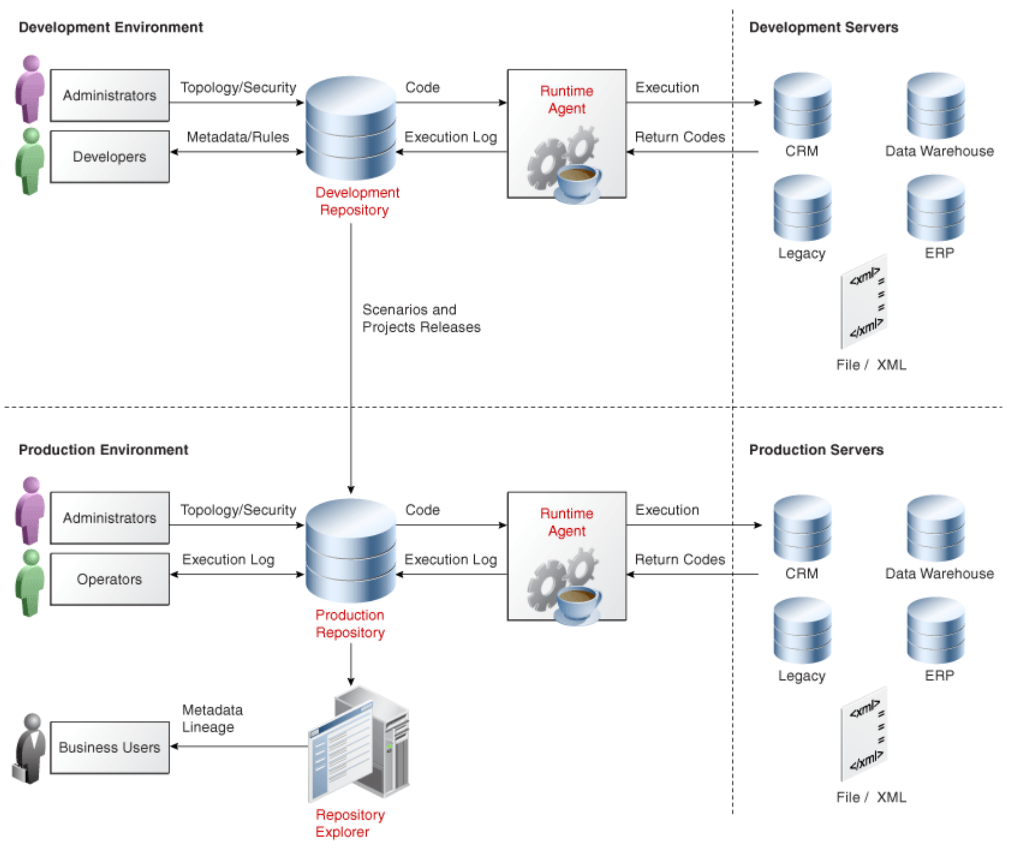
The logical view describes logical schemas that represent the physical schemas of the existing applications independently of their physical implementation. These logical schemas are then linked to the physical resources through contexts.

Designers always refer to the logical view defined in the Topology. All development done therefore becomes independent of the physical location of the resources they address. At runtime, the logical information is mapped to the physical resources, given the appropriate contexts. The same scenario can be executed on different physical servers and applications simply by specifying different contexts. This brings a very flexible architecture where developers don't have to worry about the underlying physical implementation of the servers they rely on.

Oracle Data Integrator Architecture

The architecture of Oracle Data Integrator relies on different components that collaborate together, as described in Figure 1–6.

Figure 1–6 Functional Architecture Overview



Repositories

The central component of the architecture is the Oracle Data Integrator Repository. It stores configuration information about the IT infrastructure, metadata of all applications, projects, scenarios, and the execution logs. Many instances of the repository can coexist in the IT infrastructure. The architecture of the repository is designed to allow several separated environments that exchange metadata and scenarios (for example: Development, Test, Maintenance and Production environments). In the figure above, two repositories are represented: one for the development environment, and another one for the production environment. The repository also acts as a version control system where objects are archived and assigned a version number. The Oracle Data Integrator Repository can be installed on an OLTP relational database.

The Oracle Data Integrator Repository is composed of a master repository and several Work Repositories. Objects developed or configured through the user s are stored in one of these repository types.

There is usually only one master repository that stores the following information:

- Security information including users, profiles and rights for the ODI platform
- Topology information including technologies, server definitions, schemas, contexts, languages etc.
- Versioned and archived objects.

The Work Repository is the one that contains actual developed objects. Several work repositories may coexist in the same ODI installation (for example, to have separate environments or to match a particular versioning life cycle). A Work Repository stores information for:

- Models, including schema definition, datastores structures and metadata, fields and attributes definitions, data quality constraints, cross references, data lineage etc.
- Projects, including business rules, packages, procedures, folders, Knowledge Modules, variables etc.
- Scenario execution, including scenarios, scheduling information and logs.

When the Work Repository contains only the execution information (typically for production purposes), it is then called an Execution Repository.

For more information on how to manage ODI repositories, refer to Chapter 3, "Administering Repositories".

Users

Administrators, Developers and Operators use the Oracle Data Integrator Studio to access the repositories. This Fusion Client Platform (FCP) based UI is used for administering the infrastructure (security and topology), reverse-engineering the metadata, developing projects, scheduling, operating and monitoring executions.

Business users (as well as developers, administrators and operators), can have read access to the repository, perform topology configuration and production operations through a web based UI called *Oracle Data Integrator Console*. This Web application can deployed in a Java EE application server such as Oracle WebLogic.

ODI Studio provides four Navigators for managing the different aspects and steps of an ODI integration project:

- Topology Navigator
- Designer Navigator
- Operator Navigator
- Security Navigator

Topology Navigator

Topology Navigator is used to manage the data describing the information system's physical and logical architecture. Through Topology Navigator you can manage the topology of your information system, the technologies and their datatypes, the data servers linked to these technologies and the schemas they contain, the contexts, the language and the agents, as well as the repositories. The site, machine, and data server descriptions will enable Oracle Data Integrator to execute the same mappings in different environments.

Designer Navigator

Designer Navigator is used to design data integrity checks and to build transformations such as for example:

- Automatic reverse-engineering of existing applications or databases
- Graphical development and maintenance of transformation and mappings
- Visualization of data flows in the mappings
- Automatic documentation generation
- Customization of the generated code

The main objects you handle through Designer Navigator are Models and Projects.

Operator Navigator

Operator Navigator is the production management and monitoring tool. It is designed for IT production operators. Through Operator Navigator, you can manage your executions in the sessions, as well as the scenarios in production.

Security Navigator

Security Navigator is the tool for managing the security information in Oracle Data Integrator. Through Security Navigator you can create users and profiles and assign user rights for methods (edit, delete, etc) on generic objects (data server, datatypes, etc), and fine-tune these rights on the object instances (Server 1, Server 2, etc).

Design-time Projects

A typical project is composed of several steps and milestones.

Some of these are:

- Define the business needs
- Identify and declare the sources and targets in the Topology
- Design and Reverse-engineer source and target data structures in the form of data models
- Implement data quality rules on these data models and perform static checks on these data models to validate the data quality rules
- Develop mappings using datastores from these data models as sources and target
- Develop additional components for tasks that cannot be achieved using s, such as Receiving and sending e-mails, handling files (copy, compress, rename and such), executing web services
- Integrate mappings and additional components for building Package workflows
- Version your work and release it in the form of scenarios
- Schedule and operate scenarios.

Oracle Data Integrator will help you cover most of these steps, from source data investigation to metadata lineage, and through loading and data quality audit. With its repository, Oracle Data Integrator will centralize the specification and development efforts and provide a unique architecture on which the project can rely to succeed.

Chapter 2, "Overview of an Integration Project" introduces you to the basic steps of creating an integration project with Oracle Data Integrator. Chapter 9, "Creating an Integration Project" gives you more detailed information on the several steps of creating an integration project in ODI.

Run-Time Agent

At design time, developers generate scenarios from the business rules that they have designed. The code of these scenarios is then retrieved from the repository by the Run-Time Agent. This agent then connects to the data servers and orchestrates the code execution on these servers. It retrieves the return codes and messages for the execution, as well as additional logging information – such as the number of processed records, execution time etc. - in the Repository.

The Agent comes in two different flavors:

- The Java EE Agent can be deployed as a web application and benefit from the features of an application server.
- The Standalone Agent runs in a simple Java Machine and can be deployed where needed to perform the integration flows.

Both these agents are multi-threaded java programs that support load balancing and can be distributed across the information system. This agent holds its own execution schedule which can be defined in Oracle Data Integrator, and can also be called from an external scheduler. It can also be invoked from a Java API or a web service. Refer to Chapter 4, "Setting Up a Topology" for more information on how to create and manage agents.

ODI Domains

An ODI domain contains the Oracle Data Integrator components that can be managed using Oracle Enterprise Manager Cloud Control (EMCC). An ODI domain contains:

- Several Oracle Data Integrator Console applications. An Oracle Data Integrator Console application is used to browse master and work repositories.
- Several Run-Time Agents attached to the Master Repositories. These agents must be declared in the Master Repositories to appear in the domain. These agents may be Standalone Agents or Java EE Agents. See Chapter 4, "Setting Up a Topology" for information about how to declare Agents in the Master Repositories.

In EMCC, the Master Repositories and Agent pages display both application metrics and information about the Master and Work Repositories. You can also navigate to Oracle Data Integrator Console from these pages, for example to view the details of a session. In order to browse Oracle Data Integrator Console in EMCC, the connections to the Work and Master repositories must be declared in Oracle Data Integrator Console. See *Installing and Configuring Oracle Data Integrator* for more information.

2 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.

Oracle Data Integrator QuickStart List

To perform the minimum required steps of an Oracle Data Integrator integration project follow the ODI QuickStart list and go directly to the specified section of this guide.

Prerequisites

Before performing the QuickStart 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*.

ODI QuickStart list

Step 1 of the following "QuickStart" list describes setting up the ODI Studio repository architecture. This means creating repositories to store metadata for the applications involved in transformation and integration processing, developed project versions, and all of the information required for their use (planning, scheduling and execution reports).

Steps 2 through 4 describes setting up the topology of your information system by defining the data servers, the schemas they contain, and the contexts. Refer to Chapter 1, "Introduction to Oracle Data Integrator" if you are not familiar with these concepts.

Step 5 consists of creating a model. A model is a set of datastores corresponding to data structures contained in a physical schema: tables, files, JMS messages, elements from an XML file are represented as datastores.

Steps 6 through 8 describe creating your integration project. In this project you create mappings to load data from source datastores to target datastores.

Steps 9 and 10 consist of executing the mappings you have created in step 8, and viewing and monitoring the execution results.

Step 11 describes how to creating sequential loading operations using packages.

1. Set up the Oracle Data Integrator repository architecture:

   a. You need to create one master repository containing information about the topology, security, and version management of projects and data models. Refer to "Creating the Master Repository" on page 3-4 for more details.

      To test your master repository connection, refer to "Connecting to the Master Repository" on page 3-5.

   b. You need to create at least one Work Repository containing information about data models, projects, and their operations. Refer to "Creating a Work Repository" on page 3-5 for more details.

      To test your work repository connection and access this repository through Designer and Operator, refer to the section "Connecting to a Work Repository" on page 3-7.

2. To connect source and target systems you need to declare data servers. A data server can be a database, a MOM, a connector or a file server and is always linked with one specific technology. Creating a data server corresponding to the servers used in Oracle Data Integrator is covered in "Creating a Data Server" on page 4-6.

3. A physical schema is a defining component of a data server. It allows the datastores to be classified and the objects stored in the data server to be accessed. For each data server, create the physical schemas as described in "Creating a Physical Schema" on page 4-12. Use the default Global context.

4. In Oracle Data Integrator, you perform developments on top of a logical topology. Refer to Chapter 1, "Introduction to Oracle Data Integrator," if you are not familiar with the logical architecture. Create logical schemas and associate them with physical schemas in the Global context. See "Creating a Logical Schema" on page 4-13 for more information.

5. Mappings use data models containing the source and target datastores. Data Models are usually reverse-engineered from your data server's metadata into an Oracle Data Integrator repository. Create a model according to "Creating and Reverse-Engineering a Model" on page 5-3.

6. The developed integration components are stored in a project. Creating a new project is covered in "Creating a New Project" on page 9-3.

7. Mappings use Knowledge Modules to generate their code. For more information refer to the E-LT concept in Chapter 1, "Introduction to 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 20-9. Which Knowledge Modules you need to import is discussed in *Connectivity and Knowledge Modules Guide for Oracle Data Integrator*.

8. 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 11-5.

9. Once you have finished creating a mapping, you can run it, as described in "Running Mappings" on page 11-9. Select **Local (No Agent)** to execute the mapping directly by Oracle Data Integrator.

10. You can view and monitor the execution results in the Operator navigator. Follow a mapping's execution using the Operator navigator is described in Chapter 23, "Monitoring Integration Processes."

11. 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 10-4.

Part II

Administering Oracle Data Integrator Architecture

This part describes the Oracle Data Integrator Architecture.

This part contains the following chapters:

3 Administering Repositories

This chapter describes how to create and administer Oracle Data Integrator repositories. An overview of the repositories used in Oracle Data Integrator is provided.

This chapter includes the following sections:

See Also:

- For more information about creating ODI Repositories in RCU, see *Installing and Configuring Oracle Data Integrator*.
- For more information about upgrading ODI Repositories, see *Upgrading Oracle Data Integrator*.

Introduction to Oracle Data Integrator Repositories

There are two types of repositories in Oracle Data Integrator:

- **Master Repository:** This is a data structure containing information on the topology of the company's IT resources, on security and on version management of projects and data models. This repository is stored on a relational database accessible in client/server mode from the different Oracle Data Integrator modules. In general, you need only one master repository. However, it may be necessary to create several master repositories in one of the following cases:
  - Project construction over several sites not linked by a high-speed network (off-site development, for example).
  - Necessity to clearly separate the operating environments (development, test, production), including on the database containing the master repository. This may be the case if these environments are on several sites.

- **Work Repository:** This is a data structure containing information about data models, projects, and their use. This repository is stored on a relational database accessible in client/server mode from the different Oracle Data Integrator modules. Several work repositories can be created with several master repositories if necessary. However, a work repository can be linked with only one master repository for version management purposes.

The standard method for creating repositories is using the Repository Creation Utility (RCU). The RCU automatically manages storage space as well as repository creation. However, if you want to create repositories manually, it is possible to manually create and configure the repositories using ODI Studio.

The steps needed to create and configure repositories are detailed in the following sections:

- Creating Repository Storage Spaces
- Creating the Master Repository
- Connecting to the Master Repository
- Creating a Work Repository
- Connecting to a Work Repository

Note: Oracle recommends that you regularly perform the following maintenance operations: purge the execution logs in order to reduce the work repository size, and back up the Oracle Data Integrator repositories on the database.

Advanced actions for administering repositories are detailed in "Advanced Actions for Administering Repositories" on page 3-8.

Creating Repository Storage Spaces

Oracle Data Integrator repositories can be installed on database engines supported by Oracle Fusion Middleware 12*c*. For the latest list of supported databases versions as well as the requirements for each database, see:

`http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html`

For each database that will contain a repository, a storage space must be created.

Caution: For reasons of maintenance and back-up, Oracle strongly recommends that repositories be stored separately from your application data (for example, in a different schema for an Oracle database, or in a different database for Microsoft SQL Server).

Your master repository can be stored in the same schema as one of your work repositories. However, you cannot create two different work repositories in the same schema.

The examples in the following table are supplied as a guide:

Technology	**Steps to follow**

Oracle	Create a schema `odim` to host the Master repository and a schema `odiw` to host the work repository.
	The schemas are created by the following SQL commands:
	`SQL> create user MY_SCHEMA identified by MY_PASS`
	`       default tablespace MY_TBS`
	`       temporary tablespace MY_TEMP;`
	`SQL> grant connect, resource to MY_SCHEMA;`
	`SQL> grant execute on dbms_lock to MY_SCHEMA;`
	Where:
	MY_SCHEMA corresponds to the name of the schema you want to create, such as *odim* and *odiw*
	MY_PASS corresponds to the password you have given it <MY_TBS> the Oracle tablespace where the data will be stored
	MY_TEMP temporary default tablespace

Microsoft SQL Server	Create a database `db_odim` to host the master repository and a database `db_odiw` to host the work repository. Create two logins `odim` and `odiw` which have these databases by default.
	Use Oracle Enterprise Manager to create the two databases *db_odim* and *db_odiw*.
	Use Query Analyzer or I-SQL to launch the following commands:
	`CREATE LOGIN mylogin`
	`     WITH PASSWORD = 'mypass',`
	`     DEFAULT_DATABASE = defaultbase,`
	`     DEFAULT_LANGUAGE = us_english;`
	`USE defaultbase;`
	`CREATE USER dbo FOR LOGIN mylogin;`
	`GO`
	Where:
	mylogin corresponds to odim or odiw.
	mypass corresponds to a password for these logins.
	defaultbase corresponds to db_odim and db_odiw respectively.
	Note: Oracle recommends configuring the Microsoft SQL Server databases that store the repository information with a case-sensitive collation. This enables reverse-engineering and creating multiple objects with the same name but a different case (for example: *tablename* and *TableNAme*).

DB2/400	Create a library *odim* to host the Master repository and a schema *odiw* to host the work repository. Create two users *odim* and *odiw* who have these libraries by default. **Note**: The libraries must be created in the form of SQL collections.

DB2/UDB	**Prerequisites**:
	■ Master and work repository users must have access to tablespaces with minimum 16k pagesize
	■ The database must have a temporary tablespace with minimum 16 k
	For example:
	`CREATE  LARGE  TABLESPACE ODI16 PAGESIZE 16 K  MANAGED BY`
	`AUTOMATIC STORAGE ;`
	`GRANT USE OF TABLESPACE ODI16 TO USER ODIREPOS;`

Creating the Master Repository

Creating the master repository creates an empty repository structure and seeds metadata (for example, technology definitions, or built-in security profiles) into this repository structure.

To create the master repository:

1. In ODI Studio, open the New Gallery dialog by choosing **File** > **New**.
2. In the New Gallery dialog, in the Categories tree, select **ODI**.
3. Select from the Items list the **Master Repository Creation Wizard**.
4. Click **OK**.

   The Master Repository Creation Wizard opens.
5. Specify the **Database Connection** parameters as follows:
   - **Technology**: From the list, select the technology that will host your master repository. Default is *Oracle*.
   - **JDBC Driver**: The driver used to access the technology, that will host the repository.
   - **JDBC Url**: The URL used to establish the JDBC connection to the database.

     Note that the parameters **JDBC Driver** and **JDBC Url** are synchronized and the default values are technology dependent.
   - **User**: The user ID / login of the owner of the tables (for example, `odim`).
   - **Password**: This user's password.
   - **DBA User**: The database administrator's username.
   - **DBA Password**: This user's password.
6. Specify **Repository Configuration** parameters as needed.
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. If the connection fails, fix the connection to your master repository before moving to next step.
8. Click **Next**.
9. Do one of the following:
   - Select **Use ODI Authentication** to manage users using ODI's internal security system, and enter the following supervisor login information:

Properties	Description
Supervisor User	User name of the ODI supervisor. The value must be `SUPERVISOR`.
Supervisor Password	This user's password
Confirm Password	This user's password

   - Select **Use External Authentication** to use an external enterprise identity store, such as Oracle Internet Directory, to manage user authentication, and enter the following supervisor login information:

Properties	Description
Supervisor User	User name of the ODI supervisor
Supervisor Password	This user's password

Note: In order to use the external authentication option, ODI Studio has to be configured for external authentication. See "Configuring External Authentication" on page 25-17 for more information.

10. Click **Next**.
11. Specify the password storage details for your data servers (that is, sources and targets, which are defined in the Topology):
    - Select **Internal Password Storage** if you want to store passwords in the Oracle Data Integrator master repository
    - Select **External Password Storage** if you want use JPS Credential Store Framework (CSF) to store the data server and context passwords in a remote credential store. Indicate the **MBean Server Parameters** to access the credential store. Refer to Chapter 25, "Managing Security in Oracle Data Integrator," for more information.
12. In the Master Repository Creation Wizard click **Finish** to validate your entries.

Oracle Data Integrator begins creating your master repository. You can follow the procedure on your Messages – Log. To test your master repository, refer to "Connecting to the Master Repository" on page 3-5.

Connecting to the Master Repository

To connect to the Master repository, follow the instructions in "Connecting to the Master Repository," in the *Installation Guide for Oracle Data Integrator.*

Creating a Work Repository

A master repository can have one or more work repositories associated with it; a work repository, however, can only be associated with one master repository.

To create a new work repository:

1. In the Topology Navigator, open the **Repositories** panel.
2. Right-click the Work Repositories node and select **New Work Repository**.

   The **Work Repository Creation Wizard** opens.
3. Specify the Oracle Data Integrator work repository connection details as follows:
    - **Technology:** Choose the technology of the server to host your work repository. Default is *Oracle*.
    - **JDBC Driver**: The driver used to access the technology, that will host the repository.
    - **JDBC Url:** The complete path of the data server to host the work repository.

      Note that the parameters **JDBC Driver** and **JDBC Url** are synchronized and the default values are technology dependent.

Oracle recommends using the full machine name instead of `localhost` in the JDBC URL to avoid connection issues. For example, for remote clients, the client (ODI Studio or SDK) is on a different machine than the work repository and `localhost` points to the current client machine instead of the one hosting the work repository.

- **User:** User ID / login of the owner of the tables you are going to create and host of the work repository.
- **Password:** This user's password.

4. Click **Test Connection** to verify that the connection is working.
5. Click **Next.**

   Oracle Data Integrator verifies whether a work repository already exists on the connection specified in step 3:

   - If an existing work repository is detected on this connection, the next steps will consist in attaching the work repository to the master repository. Refer to "Specify the Password of the Oracle Data Integrator work repository to attach." on page 3-9 for further instructions.
   - If no work repository is detected on this connection, a new work repository is created. Continue with the creation of a new work repository and provide the work repository details in step 6.
6. Specify the Oracle Data Integrator work repository properties:
   - **Name**: Give a unique name to your work repository (for example: *DEVWORKREP1*).
   - **Password**: Optional. Enter a password required for attaching this work repository to a different master. If you leave this option blank, no password is required for this operation.
   - **Type**: Select the type for the work repository:
     - **Development**: This type of repository allows management of design-time objects such as data models and projects (including mappings, procedures, and so on). A development repository also includes the run-time objects (scenarios and sessions). This type of repository is suitable for development environments.
     - **Execution**: This type of repository only includes run-time objects (scenarios, schedules and sessions). It allows launching and monitoring of data integration jobs in Operator Navigator. Such a repository cannot contain any design-time artifacts. Designer Navigator cannot be used with it. An execution repository is suitable for production environments.
7. Click **Finish**.
8. The Create Work Repository login dialog opens. If you want to create a login for the work repository, click **Yes** and you will be asked to enter the **Login Name** in a new dialog. If you do not want to create a work repository login, click **No**.
9. Click **Save** in the toolbar.

For more information, refer to "Connecting to a Work Repository" on page 3-7.

Connecting to a Work Repository

To connect to a work repository, click the "Connect to repository" button in ODI Studio, and enter the credentials you specified in Step8 of "Creating a Work Repository" on page 3-5.

If you did not create a work repository Login Name in Step 8 of Creating a Work Repository, in order to connect, you must create a login by performing the following steps:

1. In ODI Studio, open the New Gallery dialog by choosing **File** > **New**.
2. In the New Gallery dialog, in the Categories tree, select **ODI**.
3. Select from the Items list **Create a New ODI Repository Login**.
4. Click **OK**.

   The Repository Connection Information dialog opens.
5. Specify the Oracle Data Integrator connection details as follows:
   - **Login Name**: A generic alias (for example: *Repository*)
   - **User**: The ODI supervisor user name you have defined when creating the master repository or an ODI user name you have defined in the Security Navigator after having created the master repository.
   - **Password**: The ODI supervisor password you have defined when creating the master repository or an ODI user password you have defined in the Security Navigator after having created the master repository.
6. Specify the Database Connection (Master Repository) details as follows:
   - **User**: Database user ID/login of the schema (database, library) that contains the ODI master repository
   - **Password**: This user's password
   - **Driver List**: Select the driver required to connect to the DBMS supporting the master repository you have just created from the dropdown list.
   - **Driver Name**: The complete driver name
   - **Hurl**: The URL used to establish the JDBC connection to the database hosting the repository
7. Click **Test** to check the connection is working.
8. Select **Work Repository** and specify the work repository details as follows:
   - **Work repository name:** The name you gave your work repository in the previous step (*WorkRep1* in the example). You can display the list of work repositories available in your master repository by clicking on the button to the right of this field.
9. Click **OK** to validate your entries.

Changing a Work Repository Password

To change a work repository password:

1. In the Repositories tree of the Topology Navigator, expand the Work Repositories node.

2. Double-click a work repository, or right-click and select Open. The Work Repository Editor opens.
3. On the Definition tab of the Work Repository Editor, click **Change password**.
4. Enter the current password, and enter the new password twice.
5. Click **OK**.

Advanced Actions for Administering Repositories

Advanced actions for administering repositories do not concern the creation process of repositories. The actions described in this section deal with advanced actions performed on already existing repositories. Once the repositories are created you may want to switch the password storage or you may need to recover the password storage after a credential store crash. Actions dealing with password handling are covered in "Setting Up External Password Storage" on page 25-14. The export and import of master and work repositories is covered in Chapter 20, "Exporting and Importing."

This section contains the following topics:

- Attaching and Detaching (Deleting) a Work Repository
- Erasing a Work Repository
- Configuring Repository Connections

Attaching and Detaching (Deleting) a Work Repository

Attaching a work repository consists of linking an existing work repository to the current master repository. This existing work repository already exists in the database and has been previously detached from this or another master repository.

Deleting a work repository detaches it, by deleting its link to the master repository. This is an opposite operation to attaching. This operation does not destroy the work repository content.

Attaching a Work Repository

To attach a work repository to a master repository:

1. In the Topology Navigator, go to the **Repositories** panel.
2. Right-click the Work Repositories node and select **New Work Repository**.

   The **Work Repository Creation Wizard** opens.
3. Specify the Oracle Data Integrator work repository connection details as follows:
   - **Technology**: From the list, select the technology that will host your work repository. Default is *Oracle*.
   - **JDBC Driver**: The driver used to access the technology, that will host the repository.
   - **JDBC Url**: The complete path of the data server to host the work repository.

     Note that the parameters **JDBC Driver** and **JDBC Url** are synchronized and the default values are technology dependent
   - **User**: User ID / login of the owner of the tables you are going to create and host of the work repository.
   - **Password**: This user's password.

4. Click **Test Connection** to check the connection is working.
5. Click **Next**.
6. Specify the **Password** of the Oracle Data Integrator work repository to attach.
7. Click **Next**.
8. Specify the **Name** of the Oracle Data Integrator work repository to attach.
9. Click **Finish**.

Deleting a Work Repository

To detach a repository, delete the link from the master repository using the following procedure:

1. In the Topology Navigator, expand the **Repositories** panel.
2. Expand the **Work Repositories** node and right-click the work repository you want to delete.
3. Select **Delete**.
4. In the Confirmation dialog click **Yes**.
5. The work repository is detached from the master repository and is removed from the **Repositories** panel in Topology Navigator.

Erasing a Work Repository

Deleting a work repository is equivalent to detaching a work repository from the master repository. For more information, refer to "Attaching and Detaching (Deleting) a Work Repository" on page 3-8.

Erasing a work repository consists of deleting the work repository from the database.

WARNING: Erasing your work repository is an irreversible operation. All information stored in the work repository will be definitively deleted, including the metadata of your models, projects and run-time information such as scenarios, schedules, and logs.

Erasing a Work Repository

To erase a work repository from the database:

1. In the Topology Navigator, expand the **Repositories** panel.
2. Expand the **Work Repositories** node and right-click the work repository you want to delete.
3. Select **Erase from Database**.
4. In the Confirmation dialog click **Yes**, if you want to definitively erase the work repository from the database.
5. The work repository is erased from the database and is deleted from the **Repositories** panel in Topology Navigator.

Configuring Repository Connections

Concurrent connections to the repository database may be controlled and limited by the database engine where the repository is stored. On Oracle the initialization parameter limiting the number of connections is *processes*. When running a large number of parallel executions, you may need to tune the database to increase the maximum number of connections allowed to the repository database.

The number of connections required depends on the number of ODI Studio instances and Load Plan steps running:

- Each ODI Studio session requires two database connections (one to the master, one to the work repository) for the duration of the session, and also a third database connection is required for a security check for a very short period when the session begins.
- For non-Oracle databases, each Load Plan step consumes an additional connection as a lock while the Load Plan is being executed.

4 Setting Up a Topology

This chapter describes how to set up the topology in Oracle Data Integrator. An overview of Oracle Data Integrator topology concepts and components is provided.

This chapter includes the following sections:

- Introduction to the Oracle Data Integrator Topology
- Setting Up the Topology
- Managing Agents

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.

Before you can perform the procedures described in this chapter, you must have installed and configured Oracle Data Integrator, database schemas, domains, and agents, as described in the *Installation Guide for Oracle Data Integrator*.

Note: The Installation Guide 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

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.), 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.

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.

Logical Architecture

The logical architecture allows a user to identify as a single Logical Schema a group of similar physical schemas - that is containing datastores that are structurally identical - but located in different physical locations. Logical Schemas, like their physical counterpart, are attached to a technology.

Context allow to resolve logical schemas into physical schemas. In a given context, one logical schema resolves in 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, the model's logical schema resolves to a single physical schema, and the data contained in this schema in the data server can be accessed by the integration processes.

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 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" on page 21-20 for more information.

Typical projects will require a single Agent in production; however, "Load Balancing Agents" on page 4-18 describes how to set up several 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 arrives on the agent, the agent acknowledges this request and starts the session.
   - The agent launches the sessions start 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 Chapter 21, "Running Integration Processes" 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 or scripting engines.

Agents are multi-threaded lightweight components. An agent can run multiple sessions in parallel. When declaring a physical agent, it is recommended 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" on page 4-18.

Standalone and Java EE Agents

The Oracle Data Integrator agents exists in two flavors: standalone agent and Java EE agent.

A **standalone agent** runs in a separate Java Virtual Machine (JVM) process. It connects to the work repository and to the source and target data servers via JDBC. Standalone agents can be installed on any server with a Java Virtual Machine installed. This type of agent is more appropriate when you need to use a resource that is local to one of your data servers (for example, the file system or a loader utility installed with the database instance), and you do not want to install a Java EE application server on this machine.

A **Java EE agent** is deployed as a web application in a Java EE application server (for example Oracle WebLogic Server or IBM WebSphere). The Java EE agent can benefit from all the features of the application server (for example, JDBC data sources or clustering for Oracle WebLogic Server). This type of agent is more appropriate when there is a need for centralizing the deployment and management of all applications in an enterprise application server, or when you have requirements for high availability.

It is possible to mix in a single environment standalone and Java EE agents in a distributed environment.

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 via 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.`

Languages

Languages defines the languages and language elements available when editing expressions at design-time. Languages provided by default in Oracle Data Integrator do not require any user change.

Repositories

The topology contains information about the Oracle Data Integrator repositories. Repository definition, configuration and installation is covered in the *Installation and Upgrade Guide for Oracle Data Integrator*.

Setting Up the Topology

The following steps are a guideline to create the topology. You can always modify the topology after an initial setting:

1. Create the contexts corresponding to your different environments. See "Creating a Context" on page 4-5.
2. Create the data servers corresponding to the servers used by Oracle Data Integrator. See "Creating a Data Server" on page 4-6.
3. For each data server, create the physical schemas corresponding to the schemas containing data to be integrated with Oracle Data Integrator. See "Creating a Physical Schema" on page 4-12.
4. Create logical schemas and associate them with physical schemas in the contexts. See "Creating a Logical Schema" on page 4-13.
5. Create the physical agents corresponding to the standalone or Java EE agents that are installed in your information systems. See "Creating a Physical Agent" on page 4-13.
6. Create logical agents and associate them with physical agents in the contexts. See "Creating a Logical Agent" on page 4-14.

Creating a Context

To create a context:

1. In Topology Navigator expand the Contexts accordion.
2. Click **New context** in the accordion header.
3. Fill in the following fields:
   - **Name**: Name of the context, as it appears in the Oracle Data Integrator graphical interface.
   - **Code**: Code of the context, allowing a context to be referenced and identified among the different repositories.
   - **Password**: Password requested when the user requests switches to this context in a graphical interface. It is recommended to use a password for critical contexts (for example, contexts pointing to Production data)

- Check **Default** if you want this context to be displayed by default in the different lists in Designer Navigator or Operator Navigator.

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

Creating a Data Server

A Data Server corresponds for example to a Database, JMS server instance, a scripting engine or a file system accessed with Oracle Data Integrator in the integration flows. Under a data server, subdivisions are created in the form of Physical Schemas.

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

Pre-requisites and Guidelines

It is recommended to follow the guidelines below when creating a data server.

Review the Technology Specific Requirements

Some technologies require the installation and the configuration of elements such as:

- Installation of a JDBC Driver. See *Installing and Configuring Oracle Data Integrator* for more information.
- Installation of a Client Connector,
- Data source configuration.

Refer to the documentation of the technology you are connecting to through the data server and to the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator*. The connection information may also change depending on the technology. Refer to the server documentation provided, and contact the server administrator to define the connection methods.

Create an Oracle Data Integrator User

For each database engine used by Oracle Data Integrator, it is recommended to create a user dedicated for ODI on this data server (typically named `ODI_TEMP`).

Grant this user privileges to

- Create/drop objects and perform data manipulation in his own schema.
- Manipulate data into objects of the other schemas of this data server according to the operations required for the integration processes.

This user should be used as follows:

- Use this user name/password in the data server user/password definition.
- Use this user's schema as your Work Schema for all data schemas on this server.

Creating a Data Server

To create a Data Server:

1. In Topology Navigator expand the **Technologies** node in the Physical Architecture accordion.

Tip: The list of technologies that are displayed in the Physical Architecture accordion may be very long. To narrow the list of displayed technologies, you can hide unused technologies by selecting **Hide Unused Technologies** from the Topology Navigator toolbar menu.

2. Select the technology you want to create a data server for.
3. Right-click and select **New Data Server**
4. Fill in the following fields in the **Definition** tab:
   - **Name**: Name of the Data Server that will appear in Oracle Data Integrator.

     For naming data servers, it is recommended to use the following naming standard: `<TECHNOLOGY_NAME>_<SERVER_NAME>`.
   - **... (Data Server)**: This is the physical name of the data server used by other data servers to identify it. Enter this name if your data servers can be inter-connected in a native way. This parameter is not mandatory for all technologies.

     For example, for Oracle, this name corresponds to the name of the instance, used for accessing this data server from another Oracle data server through DBLinks.
   - **User/Password**: User name and password for connecting to the data server. This parameter is not mandatory for all technologies, as for example for the File technology.

     Depending on the technology, this could be a "Login", a "User", or an "account". For some connections using the JNDI protocol, the user name and its associated password can be optional (if they have been given in the LDAP directory).
5. Define the connection parameters for the data server:

   A technology can be accessed directly through JDBC or the JDBC connection to this data server can be served from a JNDI directory.

   If the technology is accessed through a JNDI directory:

   1. Check the **JNDI Connection** on the Definition tab.
   2. Go to the **JNDI** tab, and fill in the following fields:

Field	Description
JNDI authentication	■ **None**: Anonymous access to the naming or directory service ■ **Simple**: Authenticated access, non-encrypted ■ **CRAM-MD5**: Authenticated access, encrypted MD5 ■ **<other value>**: authenticated access, encrypted according to <other value>
JNDI User/Password	User/password connecting to the JNDI directory

Field	Description
JNDI Protocol	Protocol used for the connection Note that only the most common protocols are listed here. This is not an exhaustive list. ■ **LDAP**: Access to an LDAP directory ■ **SMQP**: Access to a SwiftMQ MOM directory ■ **<other value>**: access following the sub-protocol <other value>
JNDI Driver	The driver allowing the JNDI connection Example Sun LDAP directory: `com.sun.jndi.ldap.LdapCtxFactory`
JNDI URL	The URL allowing the JNDI connection For example: `ldap://suse70:389/o=linuxfocus.org`
JNDI Resource	The directory element containing the connection parameters For example: `cn=sampledb`

If the technology is connected through JDBC:

1. Un-check the **JNDI Connection** box.
2. Go to the **JDBC** tab, and fill in the following fields:

Field	Description
JDBC Driver	Name of the JDBC driver used for connecting to the data server
JDBC URL	URL allowing you to connect to the data server.

You can get a list of pre-defined JDBC drivers and URLs by clicking **Display available drivers** or Display URL sample.

6. From the **File** menu, click **Save** to validate the creation of the data server.

Creating a Data Server (Advanced Settings)

The following actions are optional:

- Adding Connection Properties
- Defining Data Sources
- Setting Up On Connect/Disconnect Commands

Adding Connection Properties

These properties are passed when creating the connection, in order to provide optional configuration parameters. Each property is a (key, value) pair.

- For JDBC: These properties depend on the driver used. Please see the driver documentation for a list of available properties. It is possible in JDBC to specify here the user and password for the connection, instead of specifying there in the **Definition** tab.
- For JNDI: These properties depend on the resource used.

To add a connection property to a data server:

1. On the **Properties** tab click **Add a Property**.

2. Specify a Key identifying this property. This key is case-sensitive.
3. Specify a value for the property.
4. From the **File** menu, click **Save**.

Defining Data Sources

On the Data Sources tab you can define JDBC data sources that will be used by Oracle Data Integrator Java EE Agents deployed on application servers to connect to this data server. Note that data sources are not applicable for standalone agents.

Defining data sources is not mandatory, but allows the Java EE agent to benefit from the data sources and connection pooling features available on the application server. Connection pooling allows reusing connections across several sessions. If a data source is not declared for a given data server in a Java EE agent, this Java EE agent always connects the data server using direct JDBC connection, that is without using any of the application server data sources.

Before defining the data sources in Oracle Data Integrator, please note the following:

- Datasources for WebLogic Server should be created with the **Statement Cache Size** parameter set to *0* in the **Connection Pool** configuration. Statement caching has a minor impact on data integration performances, and may lead to unexpected results such as data truncation with some JDBC drivers. Note that this concerns only data connections to the source and target data servers, not the repository connections.
- If using Connection Pooling with datasources, it is recommended to avoid **ALTER SESSION** statements in procedures and Knowledge Modules. If a connection requires **ALTER SESSION** statements, it is recommended to disable connection pooling in the related datasources.

To define JDBC data sources for a data server:

1. On the **DataSources** tab of the Data Server editor click **Add a DataSource**
2. Select a physical Agent in the **Agent** field.
3. Enter the data source name in the **JNDI Name** field.

   Note that this name must match the name of the data source in your application server.
4. Check **JNDI Standard** if you want to use the environment naming context (ENC).

   When **JNDI Standard** is checked, Oracle Data Integrator automatically prefixes the data source name with the string `java:comp/env/` to identify it in the application server's JNDI directory.

   Note that the JNDI Standard is not supported by Oracle WebLogic Server and for global data sources.
5. From the **File** menu, click **Save**.

After having defined a data source for a Java EE agent, you must create it in the application server into which the Java EE agent is deployed. There are several ways to create data sources in the application server, including:

- Configure the data sources from the application server console. For more information, refer to your application server documentation.
- "Deploying Datasources from Oracle Data Integrator in an application server for an Agent" on page 4-17

- "Deploying an Agent in a Java EE Application Server" on page 4-15

Setting Up On Connect/Disconnect Commands

On the On Connect/Disconnect tab you can define SQL commands that will be executed when a connection to a data server defined in the physical architecture is created or closed.

The On Connect command is executed every time an ODI component, including ODI client components, connects to this data server.

The On Disconnect command is executed every time an ODI component, including ODI client components, disconnects from this data server.

These SQL commands are stored in the master repository along with the data server definition.

Before setting up commands On Connect/Disconnect, please note the following:

- The On Connect/Disconnect commands are only supported by data servers with a technology type Database (JDBC).
- The On Connect and Disconnect commands are executed even when using data sources. In this case, the commands are executed when taking and releasing the connection from the connection pool.
- Substitution APIs are supported. Note that the design time tags `<%` are not supported. Only the execution time tags `<?` and `<@` are supported.
- Only global variables in substitution mode (`#GLOBAL.<VAR_NAME>` or `#<VAR_NAME>`) are supported. See "Variable Scope" for more information. Note that the Expression Editor only displays variables that are valid for the current data server.
- The variables that are used in On Connect and Disconnect commands are only replaced at runtime, when the session starts. A command using variables will fail when testing the data server connection or performing a View Data operation on this data server. Make sure that these variables are declared in the scenarios.
- Oracle Data Integrator Sequences are not supported in the On Connect and Disconnect commands.

The commands On Connect/Disconnect have the following usage:

- When a session runs, it opens connections to data servers. every time a connection is opened on a data server that has a command On Connect defined, a task is created under a specific step called *Command on Connect*. This task is named after the data server to which the connection is established, the step and task that create the connection to this data server. It contains the code of the On Connect command.
- When the session completes, it closes all connections to the data servers. Every time a connection is closed on a data server that has a command On Disunite defined, a task is created under a specific step called *Command on Disconnect*. This task is named after the data server that is disconnected, the step and task that dropped the connection to this data server. It contains the code of the On Disconnect command.
- When an operation is made in ODI Studio or ODI Console that requires a connection to the data server (such as View Data or Test Connection), the commands On Connect/Disconnect are also executed if the Client Transaction is selected for this command.

Note: You can specify whether or not to show On Connect and Disconnect steps in Operator Navigator. If the user parameter Hide On Connect and Disconnect Steps is set to `Yes`, On Connect and Disconnect steps are *not* shown.

To set up On Connect/Disconnect commands:

1. On the **On Connect/Disconnect** tab of the Data Server editor, click **Launch the Expression Editor** in the On Connect section or in the On Disconnect section.
2. In the Expression Editor, enter the SQL command.

   Note: The Expression Editor displays only the substitution methods and keywords that are available for the technology of the data server. Note that global variables are only displayed if the connection to the work repository is available.

3. Click **OK**. The SQL command is displayed in the Command field.
4. Optionally, select **Commit**, if you want to commit the connection after executing the command. Note that if **AutoCommit** or **Client Transaction** is selected in the Execute On list, this value will be ignored.
5. Optionally, select **Ignore Errors**, if you want to ignore the exceptions encountered during the command execution. Note that if **Ignore Errors** is not selected, the calling operation will end in error status. A command with **Ignore Error** selected that fails during a session will appear as a task in a Warning state.
6. From the Log Level list, select the logging level (from 1 to 6) of the connect or disconnect command. At execution time, commands can be kept in the session log based on their log level. Default is `3`.
7. From the Execute On list, select the transaction(s) on which you want to execute the command.

   Note: Transactions from 0 to 9 and the **Autocommit** transaction correspond to connection created by sessions (by procedures or knowledge modules). The **Client Transaction** corresponds to the client components (ODI Console and Studio).

   You can select **Select All** or **Unselect All** to select or unselect all transactions.
8. From the **File** menu, click **Save**.

You can now test the connection, see "Testing a Data Server Connection" on page 4-11 for more information.

Testing a Data Server Connection

It is recommended to test the data server connection before proceeding in the topology definition.

To test a connection to a data server:

1. In Topology Navigator expand the **Technologies** node in the **Physical Architecture** accordion and then expand the technology containing your data server.

2. Double-click the data server you want to test. The Data Server Editor opens.
3. Click **Test Connection**.

   The Test Connection dialog is displayed.
4. Select the agent that will carry out the test. **Local (No Agent)** indicates that the local station will attempt to connect.
5. Click **Detail** to obtain the characteristics and capacities of the database and JDBC driver.
6. Click **Test** to launch the test.

A window showing "connection successful!" is displayed if the test has worked; if not, an error window appears. Use the detail button in this error window to obtain more information about the cause of the connection failure.

Creating a Physical Schema

An Oracle Data Integrator Physical Schema corresponds to a pair of Schemas:

- A **(Data) Schema**, into which Oracle Data Integrator will look for the source and target data structures for the mappings.
- A **Work Schema**, into which Oracle Data Integrator can create and manipulate temporary work data structures associated to the sources and targets contained in the Data Schema.

Frequently used technologies have their physical schema creation methods detailed in the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator*.

Before creating a Physical Schema, note the following:

- Not all technologies support multiple schemas. In some technologies, you do not specify the work and data schemas since one data server has only one schema.
- Some technologies do not support the creation of temporary structures. The work schema is useless for these technologies.
- The user specified in the data server to which the Physical Schema is attached must have appropriate privileges on the schemas attached to this data server.

To create a Physical Schema:

1. Select the data server, Right-click and select **New Physical Schema**. The Physical Schema Editor appears.
2. If the technology supports multiple schemas:
   a. Select or type the Data Schema for this Data Integrator physical schema in **... (Schema)**. A list of the schemas appears if the technologies supports schema listing.
   b. Select or type the Work Schema for this Data Integrator physical schema in **... (Work Schema)**. A list of the schemas appears if the technologies supports schema listing.
3. Check the **Default** box if you want this schema to be the default one for this data server (The first physical schema is always the default one).
4. Go to the **Context** tab.
5. Click **Add**.
6. Select a Context and an existing Logical Schema for this new Physical Schema.

If no Logical Schema for this technology exists yet, you can create it from this Editor.

To create a Logical Schema:

1. Select an existing Context in the left column.
2. Type the name of a Logical Schema in the right column.

   This Logical Schema is automatically created and associated to this physical schema in this context when saving this Editor.

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

Creating a Logical Schema

To create a logical schema:

1. In Topology Navigator expand the **Technologies** node in the Logical Architecture accordion.
2. Select the technology you want to attach your logical schema to.
3. Right-click and select **New Logical Schema**.
4. Fill in the **schema name**.
5. For each Context in the left column, select an existing Physical Schema in the right column. This Physical Schema is automatically associated to the logical schema in this context. Repeat this operation for all necessary contexts.
6. From the **File** menu, click **Save**.

Creating a Physical Agent

To create a Physical Agent:

1. In Topology Navigator right-click the Agents node in the Physical Architecture accordion.
2. Select **New Agent**.
3. Fill in the following fields:
   - **Name**: Name of the agent used in the Java graphical interface. Note: Avoid using *Internal* as agent name. Oracle Data Integrator uses the *Internal* agent when running sessions using the internal agent and reserves the *Internal* agent name.
   - **Host**: Network name or IP address of the machine the agent will been launched on.
   - **Port**: Listening port used by the agent. By default, this port is the 20910.
   - **Web Application Context**: Name of the web application corresponding to the Java EE agent deployed on an application server. For standalone agents, this field should be set to **oraclediagent**.
   - **Protocol**: Protocol to use for the agent connection. Possible values are `http` or `https`. Default is `http`.
   - **Maximum number of sessions supported** by this agent.
   - **Maximum number of threads**: Controls the number of maximum threads an ODI Agent can use at any given time. Tune this as per your system resources and CPU capacity.

- **Maximum threads per session**: ODI supports executing sessions with multiple threads. This limits maximum parallelism for a single session execution.
- **Session Blueprint cache Management**:
  - **Maximum cache entries**: For performance, session blueprints are cached. Tune this parameter to control the JVM memory consumption due to the Blueprint cache.

    Unused Blueprint Lifetime (sec): Idle time interval for flushing a blueprint from the cache.

4. If you want to setup load balancing, go to the Load balancing tab and select a set of linked physical agent to which the current agent can delegate executions. See "Setting Up Load Balancing" on page 4-18 for more information.
5. If the agent is launched, click **Test**. The successful connection dialog is displayed.
6. Click **Yes**.

Creating a Logical Agent

To create a logical agent:

1. In Topology Navigator right-click the Agents node in the Logical Architecture accordion.
2. Select **New Logical Agent.**
3. Fill in the **Agent Name**.
4. For each Context in the left column, select an existing Physical Agent in the right column. This Physical Agent is automatically associated to the logical agent in this context. Repeat this operation for all necessary contexts.
5. From the **File** menu, click **Save**.

Managing Agents

This section describes how to work with a standalone agent, a Java EE agent and how to handle load balancing.

Standalone Agent

Managing the standalone agent involves the actions discussed in these sections:

- Configuring the Standalone Agent
- Launching a Standalone Agent
- Stopping an Agent

Note: The agent command line scripts, which are required for performing the tasks described in this section, are only available if you have installed the Oracle Data Integrator Standalone Agent. See *Installing and Configuring Oracle Data Integrator* for information about how to install the Standalone Agent.

Configuring the Standalone Agent

Configuring a standalone agent is described in *Installing and Configuring Oracle Data Integrator*. See:

- "Installing Oracle Data Integrator"
- "Creating the Oracle Data Integrator Master and Work Repository Schema"
- "Configuring the Standalone Installation Topology for the Standalone Agent"

Launching a Standalone Agent

The standalone agent is able to execute scenarios on predefined schedules or on demand. The instructions for launching the standalone agent are provided in "Starting the Node Manager and Standalone Agent," in the *Installation Guide for Oracle Data Integrator*.

Stopping an Agent

The procedure for stopping a standalone agent is described in "Stopping Your Oracle Data Integrator Agents," in the *Installation Guide for Oracle Data Integrator*

Java EE Agent

Managing a Java EE agent involves the actions discussed in the sections:

- Deploying an Agent in a Java EE Application Server
- Creating a Server Template for the Java EE Agent
- Deploying Datasources from Oracle Data Integrator in an application server for an Agent

Deploying an Agent in a Java EE Application Server

Configuring a Java EE agent is described in *Installing and Configuring Oracle Data Integrator*. See:

- "Installing Oracle Data Integrator"
- "Creating the Oracle Data Integrator Master and Work Repository Schema"
- "Configuring the Standalone Installation Topology for the Java EE Agent"

Creating a Server Template for the Java EE Agent

Oracle Data Integrator provides a Server Template Generation wizard to help you create a server template for a run-time agent.

To open the Server Template Generation wizard:

1. From the Physical Agent Editor toolbar menu, select **Generate Server Template**. This starts the Template Generation wizard.
2. In the **Agent Information** step, review the agent information and modify the default configuration if needed.

   The Agent Information includes the following parameters:

   - **General**

     Agent Name: Displays the name of the Agent that you want to deploy.

     From the list, select the server on which you want to deploy the agent. Possible values are `Oracle WebLogic` and `IBM WebSphere`.

- **Master Repository Connection**

  Datasource JNDI Name: The name of the datasource used by the Java EE agent to connect to the master repository. The template can contain a definition of this datasource. Default is `jdbc/odiMasterRepository`.

- **Connection Retry Settings**

  Connection Retry Count: Number of retry attempts done if the agent loses the connection to the repository. Note that setting this parameter to a non-zero value, enables a high availability connection retry feature if the ODI repository resides on an Oracle RAC database. If this feature is enabled, the agent can continue to execute sessions without interruptions even if one or more Oracle RAC nodes become unavailable.

  Retry Delay (milliseconds): Interval (in milliseconds) between each connection retry attempt.

- **Supervisor Authentication**

  Supervisor Key: Name of the key in the application server credential store that contains the login and the password of an ODI user with Supervisor privileges. This agent will use this user credentials to connect to the repository.

3. Click **Next**.
4. In the **Libraries and Drivers** step, select from the list the external libraries and drivers to deploy with this agent. Only libraries added by the user appear here.

   Note that the libraries can be any JAR or ZIP file that is required for this agent. Additional JDBC drivers or libraries for accessing the source and target data servers must be selected here.

   You can use the corresponding buttons in the toolbar to select or deselect all libraries and/or drivers in the list.
5. Click **Next**.
6. In the **Datasources** step, select the datasources definitions that you want to include in this agent template. You can only select datasources from the wizard. Naming and adding these datasources is done in the **Data Sources** tab of the **Physical Agent** editor.
7. Click **Next**.
8. In **Template Target and Summary** step, enter the **Target Template Path** where the server template will be generated.
9. Click **Finish** to close the wizard and generate the server template.

   The Template generation information dialog appears.
10. Click **OK** to close the dialog.

The generated template can be used to deploy the agent in WLS or WAS using the respective configuration wizard. Refer to *Installing and Configuring Oracle Data Integrator* for more information.

Declare the Supervisor in the Credential Store

After deploying the template, it is necessary to declare the Supervisor into the WLS or WAS Credential Store. Refer to *Installing and Configuring Oracle Data Integrator* for more information.

Deploying Datasources from Oracle Data Integrator in an application server for an Agent

You can create datasources from the Topology Navigator into an application server (either Oracle WebLogic Server or IBM WebSphere) for which a Java EE agent is configured.

To deploy datasources in an application server:

1. Open the Physical Agent Editor configured for the application server into which you want to deploy the datasources.
2. Go to the **Datasources** tab.
3. Drag and Drop the source/target data servers from the Physical Architecture tree in the Topology Navigator into the **DataSources** tab.
4. Provide a **JNDI Name** for these datasources.
5. Right-click any of the datasource, then select **Deploy Datasource on Server**.
6. On the Datasources Deployment dialog, select the server on which you want to deploy the data sources. Possible values are WLS or WAS server.
7. In the **Deployment Server Details** section, fill in the following fields:
   - **Host**: Host name or IP address of the application server.
   - **Port**: Bootstrap port of the deployment manager
   - **User**: Server user name.
   - **Password**: This user's password
8. In the **Datasource Deployment** section, provide the name of the server on which the datasource should be deployed, for example `odi_server1`.
9. Click **OK**.

Note: This operation only creates the Datasources definition in WebLogic Server or WebSphere Application Server. It does not install drivers or library files needed for these datasources to work. Additional drivers added to the Studio classpath can be included into the Agent Template. See "Creating a Server Template for the Java EE Agent" on page 4-15 for more information.

WLS Datasource Configuration and Usage

When setting up datasources in WebLogic Server for Oracle Data Integrator, please note the following:

- Datasources should be created with the **Statement Cache Size** parameter set to *0* in the **Connection Pool** configuration. Statement caching has a minor impact on data integration performances, and may lead to unexpected results such as data truncation with some JDBC drivers.
- If using Connection Pooling with datasources, it is recommended to avoid **ALTER SESSION** statements in procedures and Knowledge Modules. If a connection requires **ALTER SESSION** statements, it is recommended to disable connection pooling in the related datasources, as an altered connection returns to the connection pool after usage.

Load Balancing Agents

Oracle Data Integrator allows you to load balance parallel session execution between physical agents.

Each physical agent is defined with:

- A maximum number of sessions it can execute simultaneously from a work repository

  The maximum number of sessions is a value that must be set depending on the capabilities of the machine running the agent. It can be also set depending on the amount of processing power you want to give to the Oracle Data Integrator agent.
- Optionally, a number of linked physical agents to which it can delegate sessions' executions.

An agent's load is determined at a given time by the ratio `(Number of running sessions / Maximum number of sessions)` for this agent.

Delegating Sessions

When a session is started on an agent with linked agents, Oracle Data Integrator determines which one of the linked agents is less loaded, and the session is delegated to this linked agent.

An agent can be linked to itself, in order to execute some of the incoming sessions, instead of delegating them all to other agents. Note that an agent not linked to itself is only able to delegate sessions to its linked agents, and will never execute a session.

Delegation cascades in the hierarchy of linked agents. If agent A has agent B1 and B2 linked to it, and agent B1 has agent C1 linked to it, then sessions started on agent A will be executed by agent B2 or agent C1. Note that it is not recommended to make loops in agents links.

If the user parameter "Use new Load Balancing" is set to `Yes`, sessions are also re-balanced each time a session finishes. This means that if an agent runs out of sessions, it will possibly be reallocated sessions already allocated to another agent.

Agent Unavailable

When for a given agent the number of running sessions reaches its maximum number of sessions, the agent will put incoming sessions in a "queued" status until the number of running sessions falls below the maximum of sessions.

If an agent is unavailable (because it crashed for example), all its sessions in queue will be re-assigned to another load balanced agent that is neither running any session nor having sessions in queue if the user parameter *Use the new load balancing* is set to Yes.

Setting Up Load Balancing

To setup load balancing:

1. Define a set of physical agents, and link them in a hierarchy of agents (see "Creating a Physical Agent" on page 4-13 for more information).
2. Start all the physical agents corresponding to the agents defined in the topology.
3. Run the executions on the root agent of your hierarchy. Oracle Data Integrator will balance the load of the executions between its linked agents.

Part III

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:

5 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

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.

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.

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.

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 5-6.

Oracle Data Integrator is able to reverse-engineer models containing datastore shortcuts. For more information, see Chapter 17, "Using Shortcuts".

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 6, "Using Journalizing".

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

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

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.

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 9, "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.

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.

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.
   - **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 ore 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.

Reverse-Engineering File Datastores

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

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.

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.

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.

Adding and Deleting Datastore Attributes

To add attributes to a datastore:

1. In the **Attributes** tab of the datastore, click **Add Attribute** in the toolbar 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.

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.

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.

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.

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.

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.

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.

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.

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 11, "Creating and Using Mappings," for information.

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

Defining Manually 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 revers-engineering of partitions. To reverse-engineer partitions and sub-partitions, you have to use customized reverse-engineering.

To define manually 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**.

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.

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 11, "Creating and Using Mappings."

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.

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 Table 21–1, " Execution Parameters" 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.

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.

6 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

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**.

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.

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.

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.

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 Al**l.

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 Al**l.

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 10, "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.

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.

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.

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.

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.

Using Changed Data: Simple Journalizing

Using changed data from simple journalizing consists of designing mappings using journalized datastores as sources. See Chapter 11, "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.

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 where 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 Al**l.

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

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 Appendix A, "Oracle Data Integrator Tools Reference," for more information on these functions.

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 10, "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.

7 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

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.

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.

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.

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.

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.

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.

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 5, "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.

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.

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 5, "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 5, "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.

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)**.

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 *Reference*s: 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.

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, not propagation mapping from this datastore can be generated.

8 Creating and Using Data Services

This chapter describes how to configure and generate data services with Oracle Data Integrator. Data services enable access to your data via a web service interface. It also allows access to the changes captured using Oracle Data Integrator's Changed Data Capture feature.

This chapter includes the following sections:

- Introduction to Data Services
- Setting Up Data Services
- Generating and Deploying Data Services

Introduction to Data Services

Data Services are specialized Web Services that provide access to data in datastores, and to changes captured for these datastores using Changed Data Capture. These Web Services are automatically generated by Oracle Data Integrator and deployed to a Web Services container in an application server.

Data Services can be generated and deployed into a web service stack implementing the Java API for XML Web Services (JAX-WS), such as Oracle WebLogic Server or IBM WebSphere.

WARNING: Axis2 is removed in this version of Oracle Data Integrator. Customers previously using Axis2 should migrate their data services implementation by regenerating and re-deploying them in a JAX-WS container.

Setting Up Data Services

Data services are deployed in a web service container (an application server into which the web service stack is installed). This web service container must be declared in the topology in the form of a data server, attached to the **Axis2** or **JAX-WS** technology.

As data services are deployed in an application server, data sources must also be defined in the topology for accessing the data from this application server, and deployed or created in the application server.

Setting up data services involves steps covered in the following sections:

- Configuring the Web Services Container
- Setting up the Data Sources

- Configuring the Model

Configuring the Web Services Container

You must declare the web service container as a data server in the topology, in order to let Oracle Data Integrator deploy Data Services into it.

> **Note:** Be careful not to mistake the web service containers and the servers containing the data. While both are declared as data servers in Oracle Data Integrator, the former do not contain any data. They are only used to publish Data Services.

Web service containers declared in Oracle Data Integrator have one of three modes of deploying Web Services:

- Copying files directly onto the server, if you have file access to the server.
- Uploading onto the server by FTP.
- Uploading with the *Web Service Upload* method with Axis2.

The next steps in the configuration of the Web Services container depend on type of web service container and the deployment mode you choose to use.

To configure a web service container:

1. In Topology Navigator expand the **Technologies** node in the Physical Architecture panel.
2. Select the technology corresponding to the web server container: **Axis2** or **JAX-WS**. If you are using Oracle WebLogic Server or another JEE 5 compatible application server, use JAX-WS.
3. Right-click and select **New Data Server**
4. Fill in the following fields in the **Definition** tab:
   - **Name**: Name of the Data Server that will appear in Oracle Data Integrator.

     For naming data servers, it is recommended to use the following naming standard: `<TECHNOLOGY_NAME>_<SERVER_NAME>`.
   - **Base URL for published services**: Enter the base URL from which the web services will be available. For Axis2, it is `http://<host>:<HTTP Port>/axis2/services/`, and for Oracle WebLogic Server it is `http://<host>:<HTTP Port>/`
5. Select one of the following Deployment options:
   - **Save web services into directory**: directory into which the web service will be created. It can be a network directory on the application server or a local directory if you plan to deploy the web services separately into the container.
   - **Upload web services by FTP**: select this option to upload the generated web service to the container. You must provide a **FTP URL** as well as a **User name** and **Password** for performing the upload operation.
   - **Upload web services with Axis2**: select this option to upload the generated web service to the container using Axis2 web service upload mechanism. This option appears only for Axis2 containers. You must provide the **Base URL for Axis2 web application** - typically `http://<host>:<HTTP`

`Port>/axis2/axis2admin/` - as well as an Axis2 **User name** and **Password** for performing the upload operation.

6. From the **File** menu, click **Save**. The data server appears in the physical architecture.
7. Select this data server, right-click and select **New Physical Schema**. A new physical schema Editor appears. In the **Context** tab, and create a logical schema for this new physical schema, or associate it to an existing logical schema. The process for creating logical schemas is detailed in Chapter 4, "Setting Up a Topology."
8. From the **File** menu, click **Save**.

You only need to configure one physical schema for the web container. Note that the logical schema/context/physical schema association is important here as the context will condition the container into which deployment will take place.

Setting up the Data Sources

The Data Services generated by Oracle Data Integrator do not contain connection information for sources and targets. Instead, they make use of data sources defined within the Web Services container or on the application server. These data sources contain connection properties required to access data, and must correspond to data servers already defined within the Oracle Data Integrator topology.

To set up a data source, you can either:

- Configure the data sources from the application server console. For more information, refer to your application server documentation.
- Deploy the data source from Oracle Data Integrator if the container is an Oracle WebLogic Server. See Chapter 4, "Setting Up a Topology," for more information on data source deployment.

Configuring the Model

To configure Data Services, you must first create and populate a model. See Chapter 5, "Creating and Using Data Models and Datastores," for more information.

You should also have imported the appropriate Service Knowledge Module (SKM) into one of your projects. The SKM contains the code template from which the Data Services will be generated. For more information on importing KMs, see Chapter 9, "Creating an Integration Project."

To configure a model for data services:

1. In the **Models** tree in the Designer Navigator, select the model.
2. Double-click this model to edit it.
3. Fill in the following fields in the **Services** tab:
    - **Application server:** Select the logical schema corresponding to the container you have previously defined.
    - **Namespace**: Type in the namespace that will be used in the web services WSDL.
    - **Package name**: Name of the generated Java package that contains your Web Service. Generally, this is of the form `com.<company name>.<project name>`.
    - **Name of the data source**, as defined in your container. Depending on the Application server you are using, the data source might be local or global:

  - If your data source is global, you only need to enter the data source name in the Datasource name field.
  - If your data source is local, the data source name should be prefixed by `java:/comp/env/`.

  Note that OC4J uses per default a global data source, Tomcat a local data source. Refer to the documentation of your application server for more information.

- **Name of data service**: This name is used for the data services operating at the model level. You can also define a data service name for each datastore later.

4. Select a **Service Knowledge Module** (SKM) from the list, and set its options. See the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator* for more information about this KM and its options. Only SKMs imported into projects appear in this list.
5. Go to the **Deployed Datastores** tab.
6. Select every datastore that you wish to expose with a data service. For each of those, specify a **Data Service Name** and the name of the **Published Entity**.
7. From the **File** menu, click **Save**.

Although not required, you can also fine-tune the configuration of the generated data services at the datastore and attribute level.

For example, you can specify the operations that will be permitted for each attribute. One important use of this is to lock an attribute against being written to via data services.

To configure data services options at the datastore level:

1. In the **Models** tree in the Designer Navigator, select the datastore.
2. Double-click this datastore to edit it.
3. Select the **Services** tab.
4. Check **Deploy as Data Service** if you want the datastore to be deployed.
5. Enter the **Data Service Name** and the name of the **Published Entity** for the datastore.
6. From the **File** menu, click **Save**.

To configure data service options at the attribute level:

1. In the **Models** tree in the Designer Navigator, select the attribute.
2. Double-click this attribute to edit it.
3. Select the **Services** tab.
4. Check the operation that you want to allow: **SELECT, INSERT, DELETE**. The INSERT action includes the UPDATE action.
5. From the **File** menu, click **Save**.

Generating and Deploying Data Services

Once the model, data sources and container have been configured, it is possible to generate and deploy the data services.

Generating and Deploying Data Services

Generating data services for a model generates model-level data services as well as the data services for the selected datastores in this model.

To generate Data Services for a model:

1. In the **Models** tree in the Designer Navigator, select the model.
2. Right-click, and select **Generate Service**. The **Generating Data Service** window opens.
3. In the **Generating Data Service** window, fill in the following fields:
   - **Store generated Data Services in:** Oracle Data Integrator places the generated source code and the compiled Web Service here. This directory is a temporary location that can be deleted after generation. You can review the generated source code for the data services here.
   - **Context**: Context into which the data services are generated and deployed. This context choice has three effects:
     – Determining the JDBC/Java datatype bindings at generation time.
     – Determining which physical schemas are used to serve the data.
     – Determining which physical Web Services container is deployed to
   - **Generation Phases**: Choose one or more generation phases. For normal deployment, all three phases should be selected. However, it may be useful to only perform the generation phase when testing new SKMs, for instance. See below for the meaning of these phases.
4. Click **OK** to start data service generation and deployment.

Phase	Description
Generate code	This phase performs the following operation. ■ Deletes the content of the generation directory. ■ Generates the Java source code for the data services using the code template from the SKM.
Compilation	This phase performs the following operations: ■ Extracts web service framework. ■ Compiles the Java source code.
Deployment	This phase performs the following operations: ■ Packages the compiled code. ■ Deploys the package to the deployment target, using the deployment method selected for the container.
deprecated	**deprecated**
Generate 10.x style WSDL	This is not an generation phase. This is an option available when generating Axis2 web services. Select this option to generate web services compatible with a 10*g* ODI WSDL.

Overview of Generated Services

The data services generated by Oracle Data Integrator include model-level services and datastore level services. These services are described below.

Model-level services

Data services are generated at model-level when the model is enabled for consistent set CDC.

The following services are available at model-level:

- extend Window (no parameters): Carries out an extend window operation.
- lock (Subscriber Name): Locks the consistent set for the named subscriber. To lock the consistent set for several subscribers, call the service several times, using several OdiInvokeWebService steps for example.
- unlock (Subscriber Name): Unlocks the consistent set for the named subscriber.
- purge (no parameters): Purges consumed changes.

See Chapter 6, "Using Journalizing," for more information on these operations.

Datastore-level services

The range of operations offered by each generated data service depends on the SKM used to generate it. There are several common properties shared by the SKMs available with Oracle Data Integrator. In almost every case the name of the published entity forms part of the name of each operation. In the following examples, the published entity "Customer" is used.

The following operations are available at datastore-level:

- Operations on a **single** entity. These operations allow a single record to be manipulated, by specifying a value for its primary key. Other fields may have to be supplied to describe the new row, if any. Examples: addcustomer, getcustomer, deletecustomer, updatecustomer.
- Operations on a group of entities specified by **filter**. These operations involve specifying values for one or several fields to define a filter, then optionally supplying other values for the changes to made to those rows. In general, a maximum number of rows to return can also be specified. Examples: getcustomerfilter, deletecustomerfilter, updatecustomerfilter.
- Operations on a **list** of entities. This list is constructed by supplying several individual entities, as described in the "single entity" case above. Examples: addcustomerlist, deletecustomerlist, getcustomerlist, updatecustomerlist.

Testing Data Services

The easiest way to test generated data services is to use the graphical interface for the OdiInvokeWebService Oracle Data Integrator tool. See Chapter 16, "Using Web Services," for more information on this subject.

Part IV

Developing Integration Projects

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

This part contains the following chapters:

9 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

An integration project is composed of several components. These components include organizational objects, such as folder, and development objects such as mappings or 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 9-3 suggests a typical project lifestyle.

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.

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 10, "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 11, "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 13, "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 14, "Using Scenarios," for more information on scenarios.

Global Components

Global components are similar to the project objects. The main different 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.

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 practises 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 15, "Using Load Plans."

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. As this code is used to identify objects within this project, it is recommended to make this code for 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.

Managing Knowledge Modules

Knowledge Modules (KMs) are components of Oracle Data Integrator' Open Connector technology. KMs contain the knowledge required by Oracle Data Integrator 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 5, "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 11, "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 6, "Using Journalizing."

- IKM (Integration Knowledge Modules) are used to integrate (load) data to the target tables. These KMs are used in mappings. See Chapter 11, "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 5, "Creating and Using Data Models and Datastores," and Chapter 11, "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 Chapter 8, "Creating and Using Data Services."

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 20-9 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.

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* 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 Incremental Update (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 6, "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 11g Consistent (Streams)* enables CDC for Oracle 11*g* in consistent set mode using Oracle Streams features.

- *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.

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 must import the Knowledge Module appropriate 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, IKM SQL to SQL Append, IKM SQL Control Append, 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*.

Importing and Replacing Knowledge Modules

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

- When you create a new project or want to use new KMs in an existing project, you must import the KMs. You can either import a project KM or a global KM. See "Project and Global Knowledge Modules" on page 9-4 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 Module**s....
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.

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 Types" on page 20-3 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.

Encrypting and Decrypting a KM

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 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 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.

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.".

10 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

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.

Introduction to Steps

Table 10–1 lists the different types of steps. References are made to sections that provide additional details

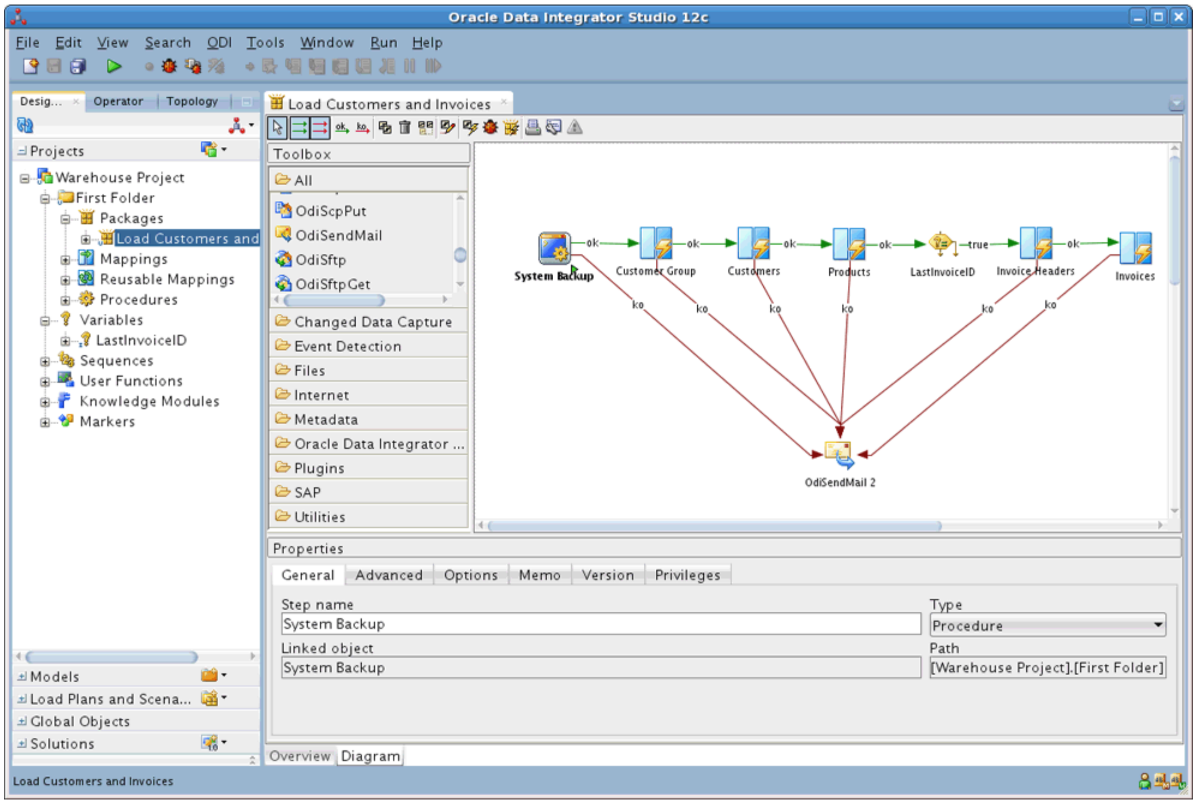
Table 10–1 Step Types

Type	Description	See Section
Flow (Mapping)	Executes an Mapping.	"Adding a Mapping step" on page 10-4
Procedure	Executes a Procedure.	"Adding a Procedure step" on page 10-5
Variable	Declares, sets, refreshes or evaluates the value of a variable.	"Variable Steps" on page 10-5
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 10-6

Table 10–1 (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 10-7

Figure 10–1 Sample Package



Step name: System Backup
Type: Procedure
Linked object: System Backup
Path: [Warehouse Project].[First Folder]

For example, the "Load Customers and Invoice" Package example shown in Figure 10–1 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.

Introduction to Creating Packages

Packages are created in the Package Diagram Editor. See "Introduction to the Package editor" on page 10-3 for more information.

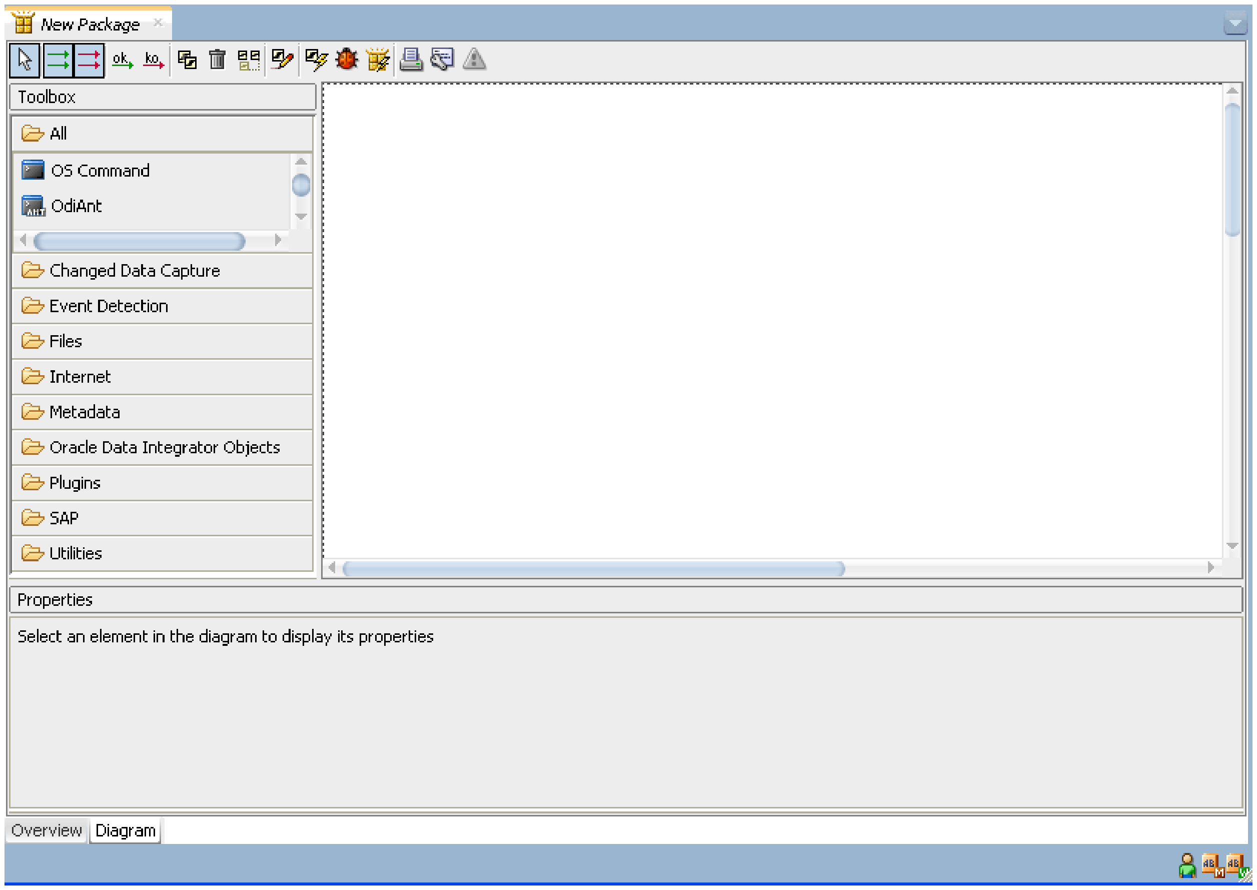
Creating a Package consists of the following main steps:

1. Creating a New Package. See "Creating a new Package" on page 10-4 for more information.
2. Working with Steps in the Package (add, duplicate, delete, and so on). See "Working with Steps" on page 10-4 for more information.
3. Defining Step Sequences. See "Defining the Sequence of Steps" on page 10-9 for more information.
4. Running the Package. See "Running a Package" on page 10-10 for more information.

Introduction to the Package editor

The Package editor provides a single environment for designing Packages. Figure 10–2 gives an overview of the Package editor.

Figure 10–2 Package editor



Select an element in the diagram to display its properties

Table 10–2 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.

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 10-4.
6. From the **File** menu, click **Save**.

Working with Steps

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

Adding a Step

Adding a step depends on the nature of the steps being inserted. See Table 10–1, " Step Types" for more information on the different types of steps. The procedures for adding the different type of steps are given below.

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**.

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**.

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 10-9 for more information on branching steps.

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

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 Appendix A, "Oracle Data Integrator Tools 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.

- **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 Appendix A, "Oracle Data Integrator Tools 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.

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 6, "Using Journalizing," for more information on these options.
5. From the **File** menu, click **Save**.

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.

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.

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.

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.

Arranging the Steps Layout

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

To 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.

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

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.

Step Type	Failure conditions
Flow	■ 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.
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.

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.

11 Creating and Using Mappings

This chapter describes how to create and use mappings.

This chapter includes the following sections:

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 12*c*.

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

Parts of a Mapping

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

- **Datastores**

  Source datastores are read 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 be populated into the data models before you can use them in a mapping. See Chapter 5, "Creating and Using Data Models and Datastores" for more information.

- **Datasets**

Optionally, you can use datasets within mappings as sources. Datasets provide a logical container in which you can organize sources, and define joins and filters on them through an entity-relationship mechanism, rather than the flow mechanism used elsewhere in mappings. Datasets operate similarly to ODI 11*g* interfaces, and if you import 11*g* interfaces into ODI 12*c*, 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 available component in the component palette:

  – Expression

  – Aggregate

  – Distinct

  – Set

  – Filter

  – Join

  – Lookup

  – Sort

  – Split

- **Connections**

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

- **Staging Schemas**

  Optionally, you can specify a staging area for a mapping or for a specific deployment specification 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 4, "Setting Up a 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 allows 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.

EKMs define how data will be extracted from your data sources.

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 target table after the data is loaded by the main mapping logic.

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

See Chapter 9, "Creating an Integration Project" for more information. See

- **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 13, "Creating and Using Procedures, Variables, Sequences, and User Functions" for more information.

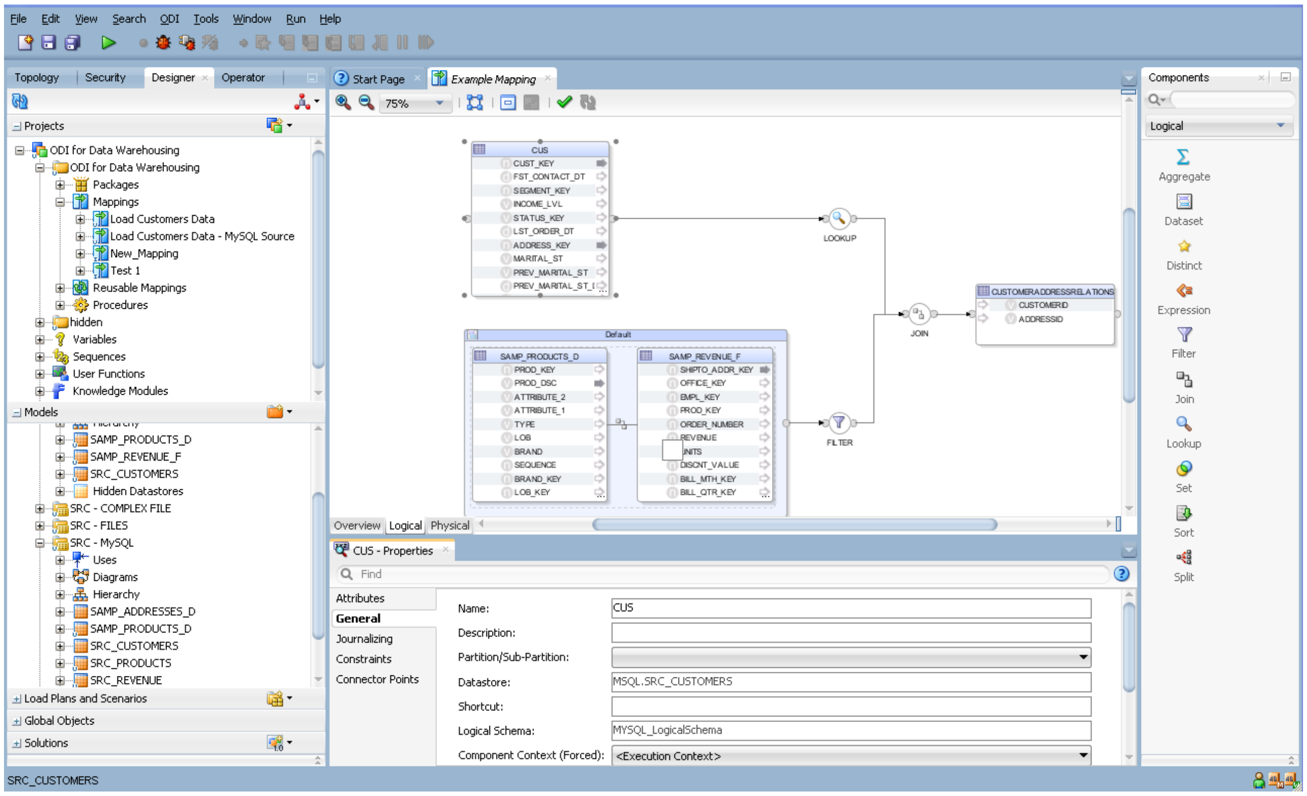
Navigating the Mapping Editor

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 11–1 Mapping Editor



Name:	CUS
Description:	
Partition/Sub-Partition:	
Datastore:	MSQL.SRC_CUSTOMERS
Shortcut:	
Logical Schema:	MYSQL_LogicalSchema
Component Context (Forced):	<Execution Context>

The mapping editor consists of the sections described in Table 11–1:

Table 11–1 Mapping Editor Sections

Section	Location in Figure 11–1	Description
Mapping Diagram	Middle	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. 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.
Mapping Editor tabs	Middle, at the bottom of the mapping diagram	The Mapping Editor tabs are ordered according to the mapping creation process. These tabs are: ■ **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	Displays properties for the selected object. If the Property Inspector does not display, select Properties from the Window menu.

Table 11–1 (Cont.) Mapping Editor Sections

Section	Location in Figure 11–1	Description
Component Palette	Right	Displays the mapping components you can use for creating mappings. You can drag and drop components into the logical or physical mapping diagram from the components palette. If the Component Palette does not display, select Components from the Window menu.
Structure Panel	Not shown	Displays a text-based hierarchical tree view of a mapping, which is navigable using the tab and arrow keys. The Structure Panel does not display by default. To open it, select Structure from the Window menu.
Thumbnail Panel	Not shown	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. The Thumbnail Panel does not display by default. To open it, select Thumbnail from the Window menu.

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 11-29 for more information.

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 mimic behavior from older versions of ODI, but they are entirely optional and all behavior of a dataset can be created using other components in the mapping editor.

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**.

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 hitting 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 read 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 or Upstream Expressions

Where applicable, when you delete a component, a check box in the confirmation dialog allows you to preserve, or remove, downstream or upstream 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, and then connect a filter in the middle. None of the mapping expressions are lost.

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 can be shown to automatically map attributes based on name or position.

Attribute Matching

The Attribute Matching Dialog is displayed when a connector is drawn to a projector component (see: "Projector Components" on page 11-10) 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 inherited 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.

Connector Points and Connector Ports

Connector points define the connections between components inside a mapping. A connection 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.

Most components have both input and output connector points (the component type may place limitations on how many connector points are allowed, and some components can have only input or only output connections). Some components allow the addition or deletion of connector points in the property inspector.

You can click a connector port on one component and drag a line to another component's connector port to define a connection. ODI will either use an unused existing connector point on each component, or create an additional connector point as needed.

For example, a Join component by default has two input connector points and one output connector point. 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 an additional Input Connector Point.

Note: You cannot drag a connection to or from a port that already has the maximum number of connections. For example, a Join component can have only one output connector point; if you try to drag another connection from the output 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.

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 validation or run-time.

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.

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

Expressions have a result type, such as VARCHAR or NUMERIC. 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 an expression on its incoming values. 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. Errors, if any, will be displayed in an error dialog.

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 11-22.

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: Chapter 21, "Running Integration Processes."

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 4-2.
   - Select the **Deployment Specification** you want to run. See: "Creating and Managing Deployment Specifications" on page 11-27.
   - 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 4-2.

- 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" on page 23-9.
- 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" on page 21-20.

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: Chapter 23, "Monitoring Integration Processes."

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 data stores 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 11-12
- "Creating Sets" on page 11-16
- "Creating Aggregates" on page 11-17
- "Creating Distincts" on page 11-20
- "Calling a Reusable Mapping" on page 11-21

- "Creating a Dataset in a Mapping" on page 11-22

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 Filters" on page 11-13
- "Creating Joins and Lookups" on page 11-14
- "Creating Sorts" on page 11-19
- "Creating Splits" on page 11-19
- "Creating Expressions" on page 11-20

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.

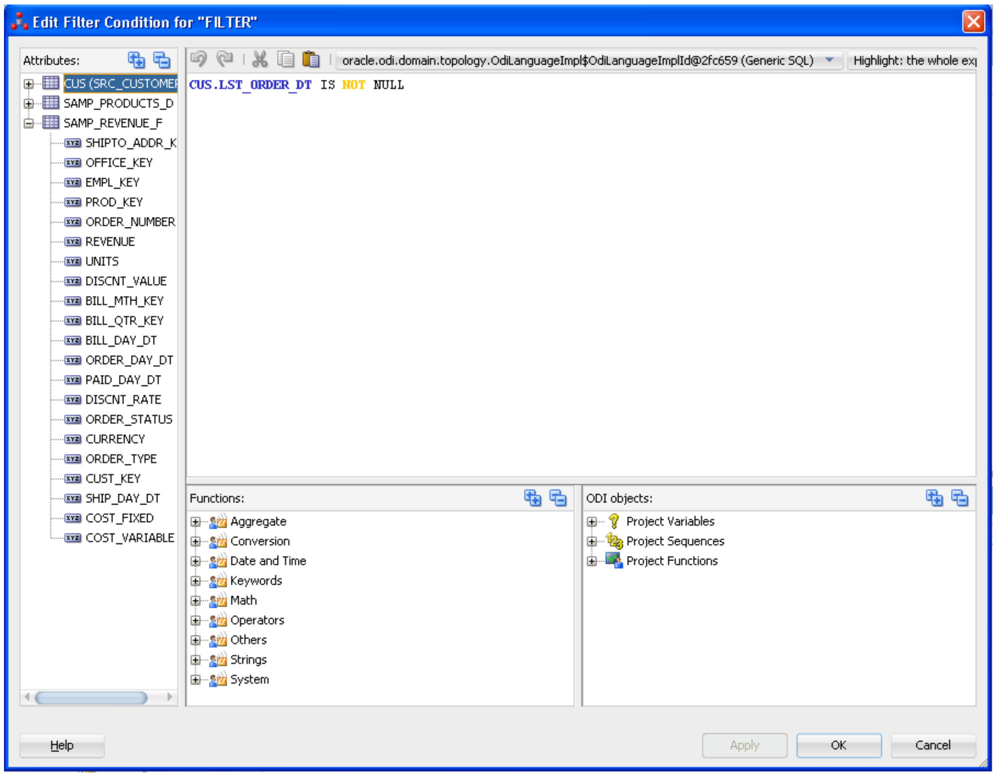
To assist you with more complex expressions, you can also open an advanced editor called the Expression Editor. (In some cases, the editor is labeled according to the type of component; for example, from a Filter component, the editor is called the Filter Condition Advanced Editor. However, the functionality 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 11–2

Figure 11–2 Example Expression Editor



```
CUS.LST_ORDER_DT IS NOT NULL
```

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 toolbar buttons.

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 one or more components to the output (right) connector of the datastore. A datastore is not a source until it has at least one outgoing connection.

   To make the datastore a target, drag a link from one or more components to the input (left) connector of the datastore. A datastore is not a target until it has at least one 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.

Creating Filters

A filter is a selector component (see: "Selector Components" on page 11-11) 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 11-11.

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 11-25.

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 11-11) 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 components, datastores, datasets, or reusable mappings.

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 11-11) 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. A lookup can be implemented in generated code either through a Left Outer Join or a nested Select statement.

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 or more 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.

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 11-11.

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 Lookup Type 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.
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*.

Creating Sets

A set component is a projector component (see: "Projector Components" on page 11-10) that combines multiple input flows into one using set operation such as `UNION`, `INTERSECT`, `EXCEPT`, `MINUS` and others. The behavior reflects the SQL operators.

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 11-7).
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 11-8).
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 group, but some of its expressions can be done on the source execution group. For more information about execution groups, see "Configuring Execution Locations" on page 11-25.

Creating Aggregates

The aggregate component is a projector component (see: "Projector Components" on page 11-10) 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 11-7).
5. If necessary, map all attributes from source to target that were not mapped though attribute matching, and create transformation expressions as necessary (see: "Defining Expressions and Conditions" on page 11-8).
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 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`.

Creating Multiple Targets

In Oracle Data Integrator 12*c*, 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, a split component should be used to define the split conditions.

See Also: "Creating Splits" on page 11-19

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, enter a target load order in the **Target Load Order** field:

   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`.

Creating Sorts

A Sort is a projector component (see: "Projector Components" on page 11-10) 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.

Creating Splits

A Split is a selector component (see: "Selector Components" on page 11-11) 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 proceeding 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.

Creating Distincts

A distinct is a projector component (see: "Projector Components" on page 11-10) 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.

Creating Expressions

An expression is a selector component (see: "Selector Components" on page 11-11) 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 11-28, and "Working with User Functions" on page 13-26.

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 11-8).

Calling 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. The reusable mapping appears in the diagram.

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_INPUTn`, the second connection `LOOKUP_INPUTn`.

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

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 11*g* way of organizing data sources, as opposed to the flow metaphor used in an ODI 12*c* mapping. If you import projects from ODI 11*g*, 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.

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 filter and join 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.

Physical Design

The physical tab shows the distribution of execution among different execution units that represent physical servers. ODI computes a default deployment specification 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 deployment specifications; they are listed in tabs under the diagram. By having multiple deployment specifications you can create different execution strategies for the same mapping. In order to create or delete deployment specifications, right-click on the deployment specification tabs.

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
- Configuring In-Session Parallelism
- Configuring Parallel Target Table Load
- Configuring Temporary Indexes
- Configuring Journalizing
- Configuring Extraction Options
- Creating and Managing Deployment Specifications

About the Physical Mapping Diagram

In the physical diagram, the following items appear:

- **Deployment Specification**: The entire physical diagram represents one deployment specification. Click the background or select the white tab representing with the deployment specification label to display the physical mapping properties. By default, the staging location is collocated 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 deployment specifications by clicking the small tab at the bottom of the physical diagram, next to the current deployment spec tab. A new deployment spec 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 among each other 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*.

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, by changing KMs several times you might lose custom KM option values.

To change the IKM 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 **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, by changing KMs several times you might lose custom KM option values.

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, by changing KMs several times you might lose custom KM option values.

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

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.

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.

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 is created. Select the execution unit to modify its properties using the property inspector.

Configuring In-Session Parallelism

ODI agent is the scheduler that runs an entire ODI mapping job on a given host. If your 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.

Configuring Parallel Target Table Load

You can enable parallel target table loading in a deployment specification. Select the deployment spec (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 11*g*, `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 12*c*, 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.

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.

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 6, "Using Journalizing."

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 SQL generation is not further configurable.

Extraction options are driven by the Extract Knowledge Module (XKM) that can be 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.

Creating and Managing Deployment Specifications

The entire physical diagram represents one deployment specification. Click the background or select the white tab with the deployment specification label to display the physical mapping properties for the displayed deployment specification.

You can define additional deployment specifications by clicking the small tab at the bottom of the physical diagram, next to the current deployment specification tab(s). A new deployment specification is created automatically, generated from the logical design of the mapping. You can modify this deployment specification, and save it as part of the mapping.

For example, you could use one deployment specification for your initial load, and another deployment specification for incremental load using changed data capture (CDC). The two deployment specifications would have different journalizing and knowledge module settings.

As another example, you could use different optimization contexts for each deployment specification. 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.

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.

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.
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.

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

Adding and Removing Components

With the Property Inspector, the Component Palette, and the Structure Panel, you can add or remove components of a mapping.

Adding Components

To add a component to a mapping with the Property Inspector 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.

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.

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.

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**.

Using Keyboard Navigation for Common Tasks

This section describes the keyboard navigation in the Property Inspector.

Table 11–2 shows the common tasks and the keyboard navigation used in the Property Inspector.

Table 11–2 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

Table 11–2 (Cont.) Keyboard Navigation for Common Tasks

Navigation	Task
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

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.

Note: In ODI 11*g* the CKM to be used when flow or static control is invoked was defined on the interface. ODI 12*c* 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

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 11-24.

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.

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.

Designing E-LT and ETL-Style Mappings

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 11-5.

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:

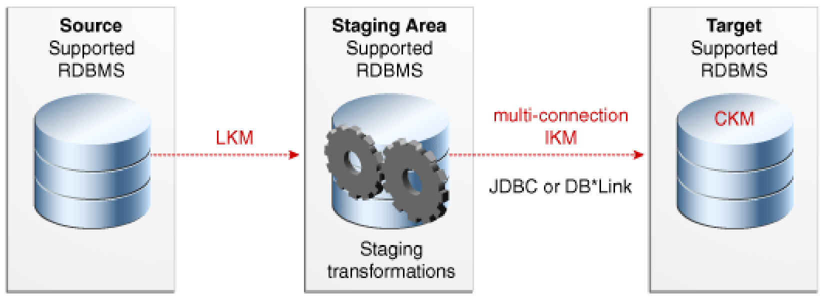
- Using a Multi-connection IKM
- Using an LKM and a mono-connection IKM

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 11–3 shows the configuration of an integration mapping using a multi-connection IKM to update the target data.

Figure 11–3 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 11-5. This section describes only the ETL-style specific steps.
2. In the Physical tab of the Mapping Editor, select a deployment spec by clicking the desired deployment spec 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.

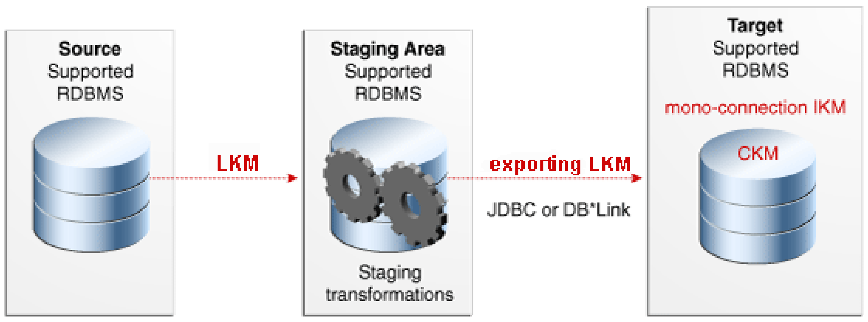
3. Select an Access Point component in the physical schema and go to the property inspector. [Link to access point definition]
4. Select an LKM from the LKM Selector list to load from the source(s) to the staging area. See the chapter in the *Connectivity and Knowledge Modules Guide for Oracle Data Integrator* 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 11–4 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 11–4 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 11-5. This section describes only the ETL-style specific steps.

2. In the Physical tab of the Mapping Editor, select a deployment spec by clicking the desired deployment spec 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 11-23.
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.

12 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

Oracle Data Integrator 12c provides a backward-compatibility feature that allows you to import and run Interfaces created in ODI 11*g*. You can create these 11*g*-compatible mappings in the following two ways:

- When upgrading to ODI 12*c* using the Upgrade Assistant, disable the **Upgrade interfaces to 12c mappings - losing 11g SDK compatibility** option
- Create mappings using the ODI 11*g* SDK

De-selecting the 11*g* SDK compatibility flag in the Upgrade Assistant is only required if you want to *modify* your 11*g*-compatible mappings later, using the 11*g* SDK. 11*g* SDK-compatible mappings are read-only when viewed in ODI Studio 12*c*. The 11*g* SDK is the only way to modify an upgraded 11*g*-compatible mapping. If you do not want to modify your 11*g*-compatible mappings, you do not need to enable 11*g* SDK compatibility using the Upgrade Assistant.

11*g*-compatible mappings use a special mapping component, the "11*g* Compatible Dataset," to emulate 11*g* Interface behavior.

You can run an 11*g* SDK-compatible mapping in ODI 12*c* in exactly the same way as running an ODI 12*c* mapping.

You can convert an 11*g* SDK-compatible mapping to a true ODI 12*c* mapping, using a context menu command.

ODI 12*c* mappings cannot be converted to 11*g* compatibility.

Creating Compatible Mappings

You have two options for creating 11*g* SDK-compatible mappings:

- Creating Mappings using Upgrade Assistant
- Creating Mappings with the 11g SDK in ODI 12c

Creating Mappings using Upgrade Assistant

You can upgrade an earlier version of ODI to ODI 12*c*, using the Oracle Upgrade Assistant. While upgrading ODI, 11*g* Interfaces are converted to 12*c* mappings by default.

The Upgrade Assistant gives you the option of enabling 11*g* SDK Compatibility Mode, by de-selecting an option. 11*g* SDK Compatibility Mode allows you to modify upgraded 11*g*-compatible mappings using the ODI 11*g* SDK.

To convert ODI 11*g* Interfaces to 11*g*-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*.

Creating Mappings with the 11*g* SDK in ODI 12*c*

In any upgraded or standard ODI 12*c* environment, you can create 11*g*-compatible mappings using the ODI 11*g* SDK. In ODI 12*c*, these mappings are read-only.

If you upgraded to ODI 12*c* using the Oracle Upgrade Assistant, and you enabled 11*g* compatibility mode, you can modify 11*g*-compatible mappings using the 11*g* SDK.

To create an 11*g*-compatible mapping using the 11*g* SDK, review the *Java API Reference for Oracle Data Integrator*.

About Internal Identifiers (IDs)

To ensure object uniqueness across several work repositories, ODI 11*g* 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 11*g* is identified by an internal ID. The internal ID appears on the Version tab of each object.

ODI 11*g* Master and Work Repositories are identified by their unique 3-digit internal IDs. The internal ID of an 11*g* 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 11*g* object IDs, these guidelines should be followed:

- ODI 11*g* Work repositories must always have different internal IDs. Work repositories with the same ID are considered to contain the same objects.
- When importing ODI 11*g* objects from an 11*g* repository, you must define an Upgrade Key. The Upgrade Key should uniquely identify the set of repositories that are working together.

Renumbering Master Repositories

Renumbering a master 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 11*g* Compatibility Mode. In ODI 12*c*, repositories have unique global identifiers, which cannot be (and do not need to be) renumbered.

Renumbering a master 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 master repository is an administrative operation that requires you to perform a backup of the repository that will be renumbered on the database.

Renumbering an 11*g* Compatible Master Repository

1. In the Topology Navigator, expand the **Repositories** panel.
2. Expand the **Master Repositories** node and right-click the master 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 master repository and click **OK**.
6. The master repository and all the details stored in it such as topology, security, and version management details are renumbered.

13

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

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

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 command lines, possibly mixing different languages. Every command line may contain two commands that can be executed on a source and on a target. The command lines are executed sequentially. Some command lines may be skipped if they are controlled by an option. These options parameterize whether or not a command line should be executed as well as the code of the commands.

The code within a procedure can be made generic by using string 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 *Knowledge Module Developer's Guide for 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.

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

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 though 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 13-4).

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.

Define the Procedure's Options

Procedure options act like parameters for your steps and improve the code reusability.

There are two 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**: 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. This is where you select which options will trigger the execution of the command.
- **Help**: Descriptive help on the option. For KMs, this text is displayed in the properties pane when the KM 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. From the **File** menu, click **Save**.
6. Repeat these operations for each option that is required for the procedure.

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 13-8 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 13-8 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 13-8 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 13-7 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 13-7.
   - **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 Table 21–1, " Execution Parameters", 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.
- **Always Execute:** Enable if you want this command to be executed all the time regardless of the option values. Otherwise, check the options of type boolean that control the command execution. At run-time, if any of the selected options is set to Yes, the command is executed.

  Note: The Always Execute option is 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

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 you 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 *Knowledge Module Developer's Guide for 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 Appendix A, "Oracle Data Integrator Tools 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 13–1 Loading Data from a Remote SQL Database

Suppose you want to insert data into the Teradata PARTS table from an Oracle PRODUCT table. Table 13–1 gives details on how to implement this in a procedure step.

Table 13–1 Procedure Details for Loading Data from a Remote SQL Database

Source Technology	Oracle
Source Logical Schema	ORACLE_INVENTORY

Source Command	`select PRD_ID       MY_PRODUCT_ID,`
	`       PRD_NAME     PRODUCT_NAME,`
	`from   <%=odiRef.getObjectName("L","PRODUCT","D")%>`

Target Technology	Teradata
Target Logical Schema	TERADATA_DWH

Target Command	`insert into PARTS`
	`(PART_ID, PART_ORIGIN, PART_NAME)`
	`values`
	`(:MY_PRODUCT_ID, 'Oracle Inventory',`
	`:PRODUCT_NAME)`

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 13–2 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 step as described in Table 13–2.

Table 13–2 Procedure Details for Sending Multiple Emails

Source Technology	Oracle
Source Logical Schema	ORACLE_DWH_ADMIN

Source Command	`Select FirstName FNAME, EMailaddress EMAIL`
	`From <%=odiRef.getObjectName("L","Operators","D")%>`
	`Where RequireWarning = 'Yes'`

Target Technology	ODITools
Target Logical Schema	None

Table 13–2 (Cont.) Procedure Details for Sending Multiple Emails

Target Command	`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`

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.

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.

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 Table 21–1, " Execution Parameters" 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.

Using a Procedure in a Package

Procedures can be used as package steps. Refer to "Adding a Procedure step" on page 10-5 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.

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 14-2.

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 KM" on page 9-9

Working with Variables

This section provides an introduction to variables and describes how to create and use variables in Oracle Data Integrator.

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 13-21 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 13-14.

The following section describes how to create and use variables.

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: ■ **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: `YYYY-MM-DDThh:mm:ssZ` where the capital letter T is used to separate the date and time components. For example: `2011-12-30T13:49:02` 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: ■ **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: ■ 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 13-14 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 15-14 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.

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.

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 10-5. 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 10-5.
- **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" on page A-74 and "Executing a Scenario from a Command Line" on page 21-4.

  How to create a Assign Variable step is covered in "Adding a Variable step" on page 10-5.

- **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 10-5.
- **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 10-5.

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 13–3 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.

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

Using Variables in Procedures

You can use variables anywhere within your procedures' code as illustrated in the Table 13–4.

Table 13–4 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

Table 13–4 (Cont.) Example of how to use Variables in a Procedure

Step ID:	Step Type	Step Code	Description
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 13–4, 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.

Using Variables within Variables

It is sometimes useful to have variables depend on other variable values as illustrated in Table 13–5.

Table 13–5 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 13–5, 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`

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.

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 13–3 illustrates how you would implement this in Oracle Data Integrator:

Example 13–3 Referring to a Variable in the URL of a Server's Definition

1. Create a StoresLocation table as follows:

Table 13–6 Stores Location table

StoreID	Store Name	Store URL	IsActive
1	Denver	10.21.32.198:1521:ORA1	YES
2	San Francisco	10.21.34.119:1525:SANF	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:

Table 13–7 SELECT/action command

Source Technology	Oracle (technology of the data server containing the StoresLocation table).
Source Logical Schema	Logical schema containing the StoresLocation table.

Source Command	`Select StoreId`
	`From StoresLocation`

Target Technology	ODITools
Target Logical Schema	None
Target Command	`SnpsStartScen "-SCEN_NAME=LOAD_STORE" "-SCEN_VERSION=1" "-SYNC_MODE=2" "-EDW.STORE_ID=#StoreId"`

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 13-8 and "Managing Agents" on page 4-14 for further details.

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)" on page 4-8 for more information.

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" on page 21-4.

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 14-2.

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 13-12 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" on page 23-6 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.

Working with Sequences

This section provides an introduction to sequences and describes how to create and use sequences in Oracle Data Integrator.

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 13-21 for more information.

The following sections describe how to create and use sequences.

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

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.

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.

Schema	**Logical schema containing the sequences table**
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_TAB = '3'` 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.

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.

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 valued 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.

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.

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.

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.

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.

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 13-28.

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.

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.

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 13–8 shows how to implement this as a user function.

Table 13–8 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	`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`

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 13–9 User Function Translated into Code for Different Technologies (Example 2)

Implementation for Oracle	`substring(Initcap(to_char(CUSTOMER.LAST_ORDER_DATE 'MONTH')) , 1, 3)`
Implementation for Teradata	`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)`
Implementation for Microsoft SQL	`substring(datename(month, CUSTOMER.LAST_ORDER_DATE) , 1, 3)`

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.

14 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

When a component is finished and tested, you can generate the **scenario** corresponding its actual state. This operation takes place in 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 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 a development environment under the source component in the Projects tree of Designer Navigator, and appear - for development and production environments - in the Scenarios tree of Operator Navigator.

Scenarios can also be versioned. See Chapter 19, "Using Version Control," 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 Chapter 21, "Running Integration Processes."

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.
2. On the **Scenarios** tab, click **Generate Scenario**. The New Scenario dialog appears.
3. 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 (if the latest version is an integer) or set to the current date (if the latest version is not an integer)

   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.

4. Click **OK**.
5. 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 unselect all variables
6. Click **OK**.

The scenario appears on the Scenarios tab and under the Scenarios node of the source object under the project.

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 the scenario in the Projects accordion.
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.

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 Generation dialog, select the scenario **Generation Mode**:
   - **Replace**: Overwrites for each object the last scenario version with a new one with the same 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 id, name and version. It preserves the schedule, sessions and scenario reports. 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.
   >
   > If the version of the last scenario is an integer, it will be automatically incremented by 1 when selecting the **Creation** generation mode. If not, the version will be automatically set to the current date.

4. In the **Objects to Generate** section, select the types of objects for which you want to generate scenarios.
5. In the **Marker Filter** section, you can filter the components to generate according to a marker from a marker group.
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.

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 20-8.

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 20-7 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 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. `<?`**`xml version="1.0"`**` encoding="ISO-8859-1"?>`
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.

6. Click **OK**.

The XML-formatted export files are created at the specified location.

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 via Designer or Operator Navigator. With a execution repository, only 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," for more information.

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 20, "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.

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**.

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

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. In this case, please type in a longer personal key. A personal key of 10 or more characters is required.

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.

15 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

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 15-6.

Load Plans allow setting and using variables at multiple levels. See "Working with Variables in Load Plans" on page 15-14 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 15-15 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 15-17 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 15-11 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 15-18 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.

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 15-17 for more information.

Differences between Packages, Scenarios, and Load Plans

A *Load Plan* is the largest executable object in Oracle Data Integrator. It uses *Scenario*s 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.

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 15–1 lists the different types of Load Plan steps and the possible child steps.

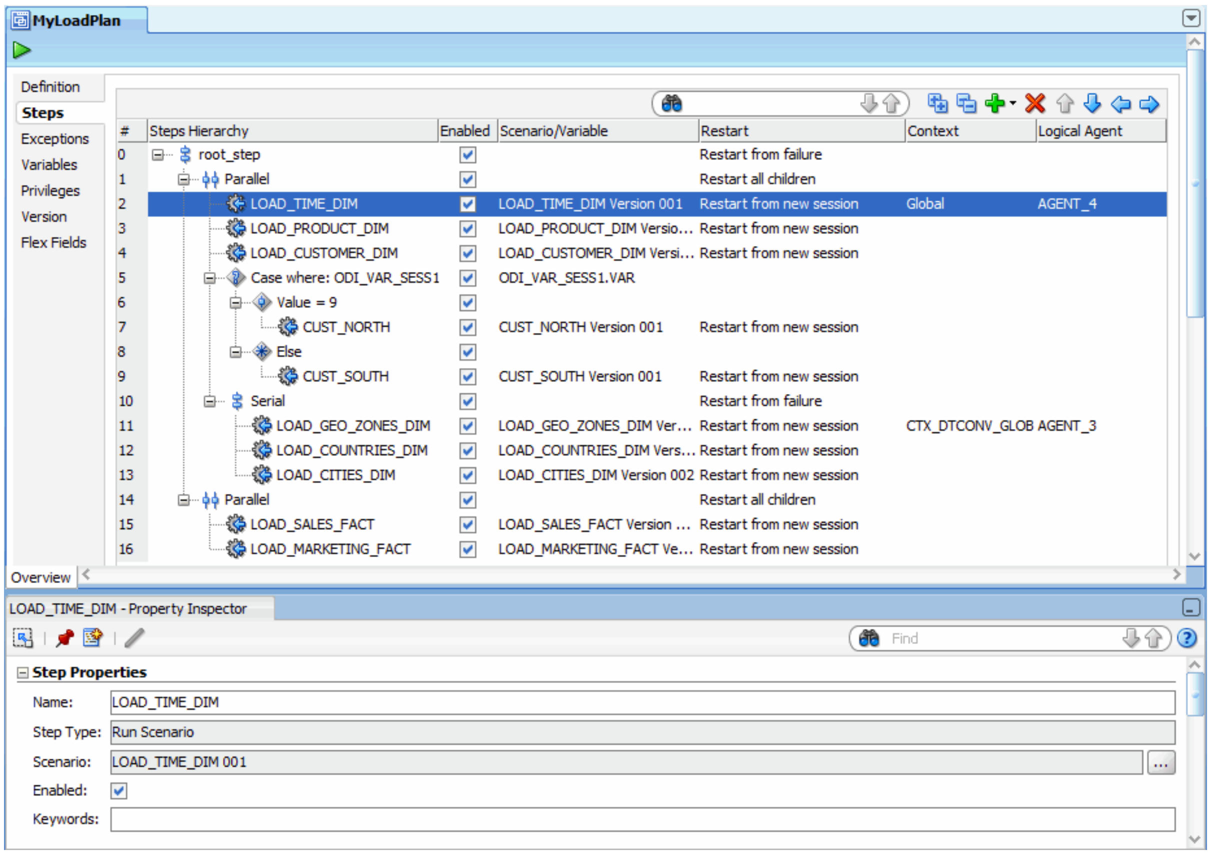
Table 15–1 Load Plan Steps

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.	■ 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.	■ 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.	Of a Case Step: ■ When step ■ Else step Of a When step: ■ Serial step ■ Parallel step ■ Run Scenario step ■ Case step Of an Else step: ■ 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.	■ Serial step ■ Parallel step ■ Run Scenario step ■ Case step

Figure 15–1 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 15–1 Sample Load Plan



#	Steps Hierarchy	Enabled	Scenario/Variable	Restart	Context	Logical Agent
0	root_step	✓		Restart from failure		
1	Parallel	✓		Restart all children		
2	LOAD_TIME_DIM	✓	LOAD_TIME_DIM Version 001	Restart from new session	Global	AGENT_4
3	LOAD_PRODUCT_DIM	✓	LOAD_PRODUCT_DIM Versio…	Restart from new session		
4	LOAD_CUSTOMER_DIM	✓	LOAD_CUSTOMER_DIM Versi…	Restart from new session		
5	Case where: ODI_VAR_SESS1	✓	ODI_VAR_SESS1.VAR			
6	Value = 9	✓				
7	CUST_NORTH	✓	CUST_NORTH Version 001	Restart from new session		
8	Else	✓				
9	CUST_SOUTH	✓	CUST_SOUTH Version 001	Restart from new session		
10	Serial	✓		Restart from failure		
11	LOAD_GEO_ZONES_DIM	✓	LOAD_GEO_ZONES_DIM Ver…	Restart from new session	CTX_DTCONV_GLOB	AGENT_3
12	LOAD_COUNTRIES_DIM	✓	LOAD_COUNTRIES_DIM Vers…	Restart from new session		
13	LOAD_CITIES_DIM	✓	LOAD_CITIES_DIM Version 002	Restart from new session		
14	Parallel	✓		Restart all children		
15	LOAD_SALES_FACT	✓	LOAD_SALES_FACT Version …	Restart from new session		
16	LOAD_MARKETING_FACT	✓	LOAD_MARKETING_FACT Ve…	Restart from new session		

LOAD_TIME_DIM - Property Inspector

Find

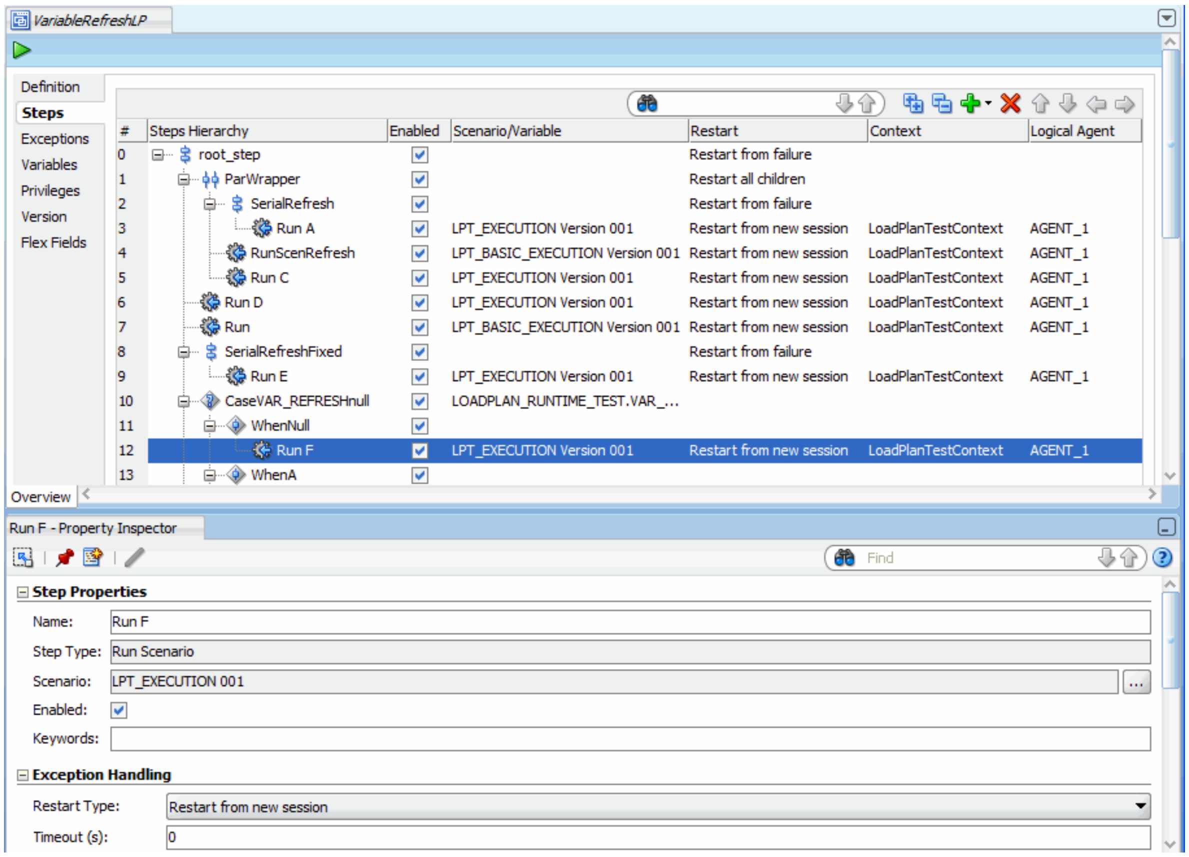
Step Properties

Name:	LOAD_TIME_DIM
Step Type:	Run Scenario
Scenario:	LOAD_TIME_DIM 001
Enabled:	✓
Keywords:	

Introduction to the Load Plan Editor

The Load Plan Editor provides a single environment for designing Load Plans. Figure 15–2 gives an overview of the Load Plan Editor.

Figure 15–2 Steps Tab of the Load Pan Editor



VariableRefreshLP

Definition
Steps
Exceptions
Variables
Privileges
Version
Flex Fields

#	Steps Hierarchy	Enabled	Scenario/Variable	Restart	Context	Logical Agent
0	root_step	✓		Restart from failure		
1	ParWrapper	✓		Restart all children		
2	SerialRefresh	✓		Restart from failure		
3	Run A	✓	LPT_EXECUTION Version 001	Restart from new session	LoadPlanTestContext	AGENT_1
4	RunScenRefresh	✓	LPT_BASIC_EXECUTION Version 001	Restart from new session	LoadPlanTestContext	AGENT_1
5	Run C	✓	LPT_EXECUTION Version 001	Restart from new session	LoadPlanTestContext	AGENT_1
6	Run D	✓	LPT_EXECUTION Version 001	Restart from new session	LoadPlanTestContext	AGENT_1
7	Run	✓	LPT_BASIC_EXECUTION Version 001	Restart from new session	LoadPlanTestContext	AGENT_1
8	SerialRefreshFixed	✓		Restart from failure		
9	Run E	✓	LPT_EXECUTION Version 001	Restart from new session	LoadPlanTestContext	AGENT_1
10	CaseVAR_REFRESHnull	✓	LOADPLAN_RUNTIME_TEST.VAR_...			
11	WhenNull	✓				
12	Run F	✓	LPT_EXECUTION Version 001	Restart from new session	LoadPlanTestContext	AGENT_1
13	WhenA	✓				

Overview

Run F - Property Inspector

Find

Step Properties

Name:	Run F
Step Type:	Run Scenario
Scenario:	LPT_EXECUTION 001
Enabled:	✓
Keywords:	

Exception Handling

Restart Type:	Restart from new session
Timeout (s):	0

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 15-8 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 15–2 details the different toolbar components.

Table 15–2 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 15–2 (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 15-8 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.

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 15-6 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 15-7 for more information.
3. Define how the exceptions should be handled. See "Handling Load Plan Exceptions and Restartability" on page 15-15 for more information.

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** 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 Task Log Level**: Select the log level for the 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" on page 23-9 for more information.

4. Go to the **Steps** tab and add steps as described in "Defining the Load Plan Step Sequence" on page 15-7.
5. If your Load Plan requires conditional branching, or if your scenarios use variables, go to the **Variables** tab and declare variables as described in "Declaring Load Plan Variables" on page 15-14.
6. To add exception steps that are used in the event of a load plan step failing, go to the **Exceptions** tab and define exception steps as described in "Defining Exceptions Flows" on page 15-15.
7. 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.

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 15–1, " 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 15-8 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 15–2, " 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

Adding Load Plan Steps

A Load Plan step can be added either by using the Add Step Wizard or by selecting the Quick Step tool for a specific step type. See Table 15–1, " 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 15-11 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 15–1, " 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 15–3 for the step type you are adding.

Table 15–3 Add Step Wizard Actions

Step Type	Description and Action Required
Serial or Parallel step	Enter a **Step Name** for the new Load Plan step.
Run Scenario step	1. Click the **Lookup Scenario** button. 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 15-11 for more information. Note that when you use the version number `-1`, the latest version of the scenario will be used. 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. 4. Click **Next**. 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 15-14 for more information.
Case	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 15-14 for more information. 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 15-11 for more information.

Table 15–3 (Cont.) Add Step Wizard Actions

Step Type	Description and Action Required
When	1. Select the **Operator** to use in the WHEN clause evaluation. Possible values are: ■ Less Than (<) ■ Less Than or Equal (<=) ■ Different (<>) ■ Equals (=) ■ Greater Than (>) ■ Greater Than or Equal (>=) ■ Is not Null ■ Is Null 2. Enter the **Value** to use in the WHEN clause evaluation. 3. The **Step Name** is automatically populated with the operator that is used. Optionally, change the Step Name. See "Editing Load Plan Steps" on page 15-11 for more information.
Else	The **Step Name** is automatically populated with the step type. Optionally, change the Step Name. See "Editing Load Plan Steps" on page 15-11 for more information.

7. Click **Finish**.
8. 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:

1. Open the Load Plan editor and go to the **Steps** tab.
2. In the Steps Hierarchy, select the Load Plan step under which you want to create a child step.
3. In the Steps toolbar, select **Add Step** and the Quick Step option corresponding to the Step type you want to add. Table 15–4 lists the options of the Quick Step tool.

Table 15–4 Quick Step Tool

Quick Step tool option	Description and Action Required
↓	Adds a serial step after the selection. 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 15-11 for more information.

Table 15–4 (Cont.) Quick Step Tool

Quick Step tool option	Description and Action Required
⧫⧫	Adds a parallel step after the selection. 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 15-11 for more information.
⚙	Adds a run scenario step after the selection. Follow the instructions for Run Scenario steps in Table 15–3.
◆	Adds a Case step after the selection. Follow the instructions for Case steps in Table 15–3.
◆	Adds a When step after the selection. Follow the instructions for When steps in Table 15–3.
◆	Adds an Else step after the selection. Follow the instructions for Else steps in Table 15–3.

Note: Only step types that are valid for the current selection are enabled in the Quick Step tool.

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 15–3 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 15–3, " 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 15-15 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**.

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.

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.

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: 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.

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

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.

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.

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.

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

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.

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 15-15 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.

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 15–5.

Table 15–5 Restart Type Values

Step Type	Values and Description
Serial	■ **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.
Parallel	■ **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	■ **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. The same limitation as those described in "Restarting a Session" on page 21-6 apply to the sessions restarted from a failed step or failed task.

Running Load Plans

You can run a Load Plan from Designer Navigator or Operator Navigator in ODI Studio.

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. See Chapter 23, "Monitoring Integration Processes," for more information. See also Chapter 21, "Running Integration Processes," for more information on the other run-time operations on Load Plans.

Using Load Plans in Production

Using Load Plans in production involves the following tasks:

- Starting, monitoring, stopping and restarting Load Plans. See "Running Load Plans in Production" on page 15-18 for information.
- Scheduling Load Plans. See "Scheduling Load Plans" on page 15-18 for more information.
- Moving Load Plans across environments. See "Exporting, Importing and Versioning Load Plans" on page 15-18

Running Load Plans in Production

In Production, the following tasks can be performed to execute Load Plans interactively:

- Executing a Load Plan
- Restarting a Load Plan Run
- Stopping a Load Plan Run

Scheduling Load Plans

You can schedule the executions of your scenarios and Load Plans using the Oracle Data Integrator built-in scheduler or an external scheduler. See "Scheduling Scenarios and Load Plans" on page 21-16 for more information.

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.

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 20-8.

For more information on exporting several Load Plans at once, see "Export Multiple ODI Objects" on page 20-8.

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 14-4.

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 20-9 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.

Versioning Load Plans

Load Plans can also be deployed and promoted to production using versions and solutions. See Chapter 19, "Using Version Control," for more information.

16 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

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 16–1 shows an overview of how the different types of Web services can interact.

Figure 16–1 Web Services in Action

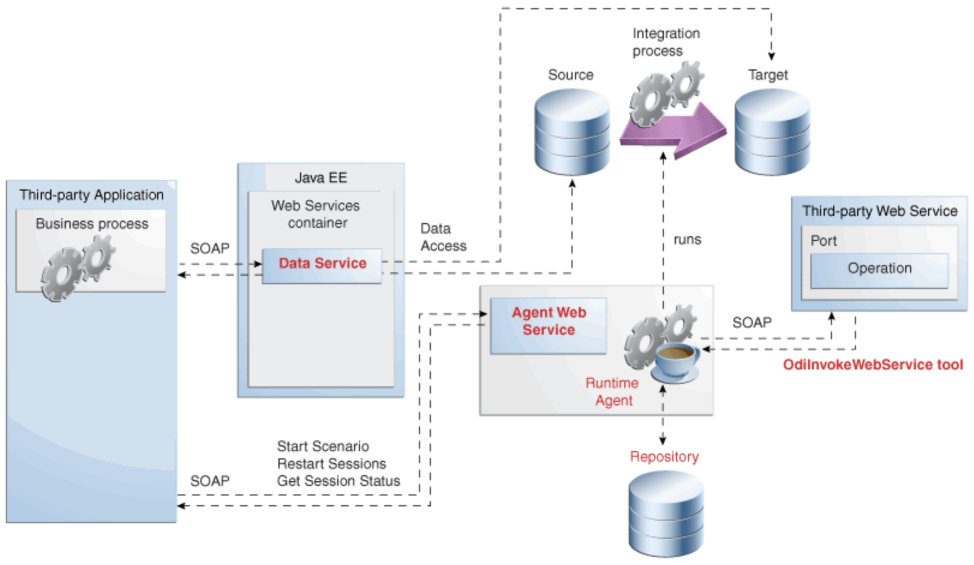


Figure 16–1 shows a simple example with the Data Services, Run-Time Web services (Public Web service and Agent Web service) and the OdiInvokeWebService tool.

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.

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.

  How to perform the different ODI execution tasks with the ODI Run-Time Services, such as executing a scenario and restarting a session, is detailed in "Managing Executions Using Web Services" on page 21-20. 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 Chapter 8, "Creating and Using Data Services."

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

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.

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

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.

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.

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:

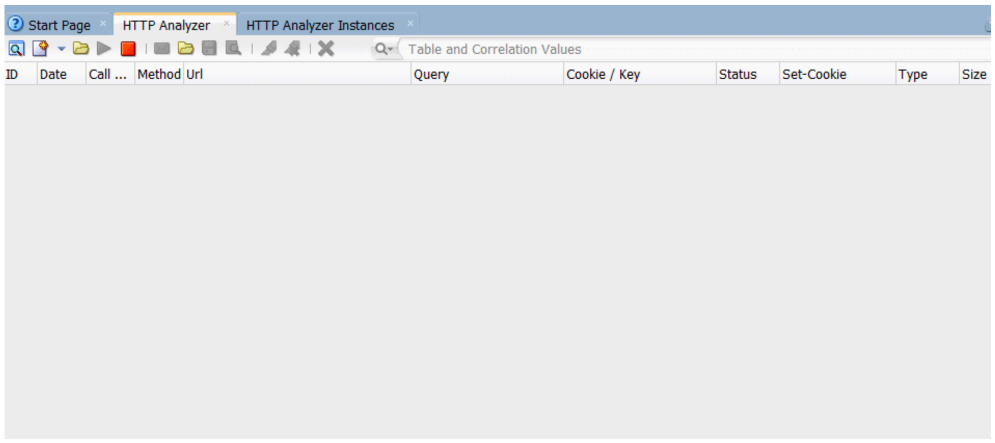
1. Open the HTTP Analyzer preferences dialog by doing one of the following:
   - Click the Start HTTP Analyzer button in the HTTP Analyzer Instances window or Log window.
   - Choose **Tools > Preferences** to open the Preferences dialog, and navigating to the HTTP Analyzer page.

   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**.

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 16–2.

Figure 16–2 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 16–1 HTTP Analyzer Log Window Toolbar Icons

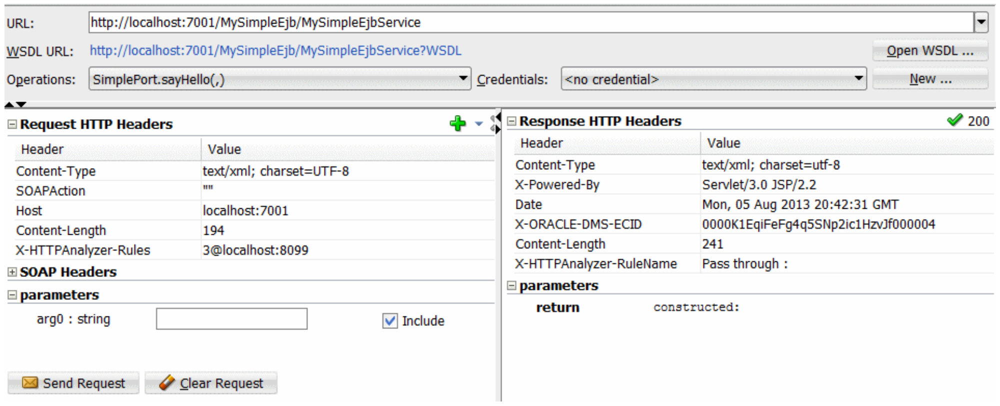
Icon	Name	Function
	Analyzer Preferences	Click to open the HTTP Analyzer Preferences dialog where you can specify a new listener port, or change the default proxy. An alternative way to open this dialog is to choose **Tools > Preferences**, and then navigate to the HTTP Analyzer page. For more information, 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.

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 16–3.

Figure 16–3 HTTP Analyzer Test Window



URL: http://localhost:7001/MySimpleEjb/MySimpleEjbService

WSDL URL: http://localhost:7001/MySimpleEjb/MySimpleEjbService?WSDL

Operations: SimplePort.sayHello(,)

Credentials: <no credential>

Request HTTP Headers

Header	Value
Content-Type	text/xml; charset=UTF-8
SOAPAction	""
Host	localhost:7001
Content-Length	194
X-HTTPAnalyzer-Rules	3@localhost:8099

SOAP Headers

parameters

arg0 : string ✓ Include

Response HTTP Headers ✓ 200

Header	Value
Content-Type	text/xml; charset=utf-8
X-Powered-By	Servlet/3.0 JSP/2.2
Date	Mon, 05 Aug 2013 20:42:31 GMT
X-ORACLE-DMS-ECID	0000K1EqiFeFg4q5SNp2ic1HzvJf000004
Content-Length	241
X-HTTPAnalyzer-RuleName	Pass through :

parameters

return constructed:

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 16–3.
- 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>
```

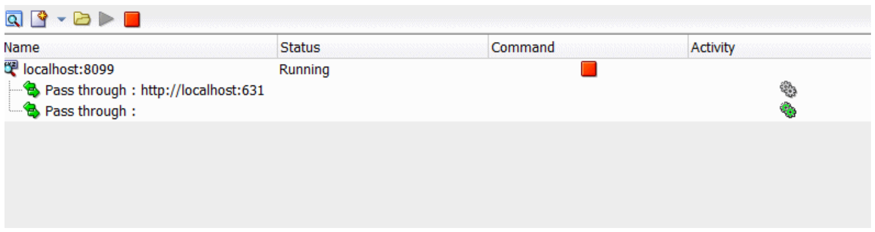
How to Use the Instances Window

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 16–4.

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 16–4 HTTP Analyzer Instances Window



Name	Status	Command	Activity
localhost:8099	Running	🟥	
Pass through : http://localhost:631			
Pass through :			

You create a new instance in the HTTP Analyzer dialog, which opens when you click the **Create New Soap Request** button.

Table 16–2 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 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.

Table 16–2 (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.

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.

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.

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.

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.

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

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 which just passes the request to the original service. This rule should normally be moved to end of the list if new rules are added.

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.

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.

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.

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.

Reference: Troubleshooting the HTTP Analyzer

This section contains information to help resolve problems that you may have when running the HTTP Analyzer.

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.

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.

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 OdiInvokeWebServiceAdvance, which you can use to configure command parameters.

See "OdiInvokeWebService" on page A-39 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 10-6 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 the XML section of "Oracle Data Integrator Tools by Category" on page A-8 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 13-8 for more information.

17 Using Shortcuts

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

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.

Shortcutting Concepts

A *shortcut* is a link to an Oracle Data Integrator object. You can create a shortcut for datastores, 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 model folders.

See "Working with Shortcuts in your Projects" on page 17-4 for more information.

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. The shortcut icons are listed in Table 17–1.

Table 17–1 Shortcut Icons

Shortcut Icon	Shortcut Object
	Datastore
	Mapping
	Package
	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*, 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 package, or a Load Plan step

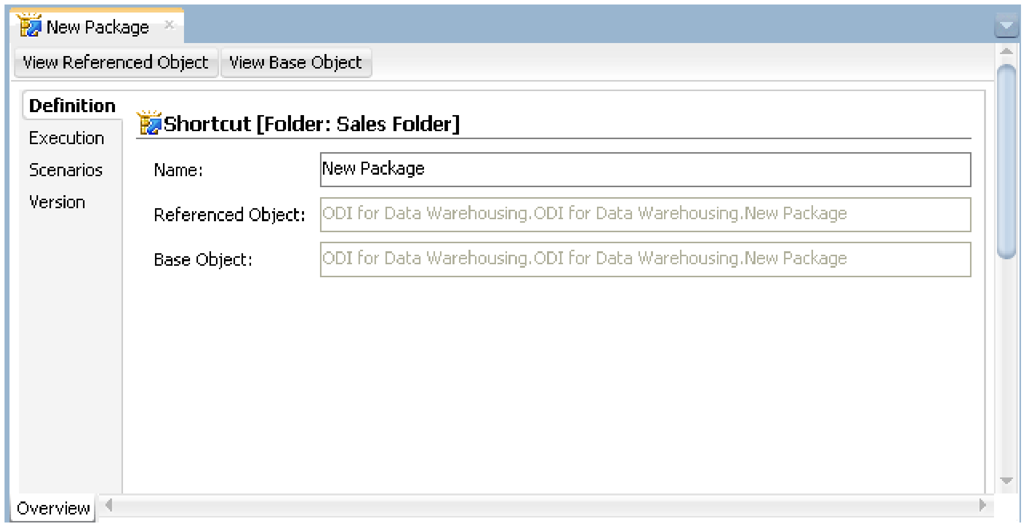
  – Procedure shortcuts can be used in a package or a Load Plan step

  When adding a shortcut to a Load Plan step, Oracle Data Integrator converts the shortcut object into a Run Scenario step.

Introduction to the Shortcut Editor

The Shortcut editor provides a single environment for editing and managing shortcuts in Oracle Data Integrator. Figure 17–1 gives an overview of the Shortcut editor.

Figure 17–1 Shortcut Editor of a Package Shortcut



Shortcut [Folder: Sales Folder]

Name: New Package

Referenced Object: ODI for Data Warehousing.ODI for Data Warehousing.New Package

Base Object: ODI for Data Warehousing.ODI for Data Warehousing.New Package

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

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 17-2.

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 17-2 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 17-4 for more information.

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

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.**

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.

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.

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 Chapter 21, "Running Integration Processes," for more information.

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 this is a temporary mapping or 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. Please review the materialized mapping.

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 20-11 for more information.

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 20-12 for more information.

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 Shortcut Model

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 5-9.

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 5-12.

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 5-12.

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 14-2.

Reverse-Engineer a Shortcut Model

You can reverse-engineer a shortcut model 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 5, "Creating and Using Data Models and Datastores."

Part V

Managing Integration Projects

This part describes how to organize and maintain your Oracle Data Integrator projects.

This part contains the following chapters:

18 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

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.

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.

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.

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.

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.

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.

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 wan 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.

Datastores 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.

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.

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 an 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.

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 12*c* 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," and Chapter 20, "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.

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.

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 14-3 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.

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.

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. This two methods can be used simultaneously or separately.

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.

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.

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**.

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**.

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.

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**.

Generating a Diagram Report

You can generate a complete PDF report of your diagram. Refer to "Printing a Diagram" on page 7-4 for more information.

19 Using Version Control

This chapter describes how to work with version management in Oracle Data Integrator.

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 **solutions**.

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 Solutions

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.

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
- Solutions
- 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).
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 19-3.

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. 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.

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 Solution.

- **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.

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).

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.

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 the Printer icon.
2. In the Report Generation dialog, set the **object** and **field filters** 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 file after its generation.

   Select **Open the file after the generation** 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.

Working with Solutions

A **solution** 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. Solutions are saved into the master repository. A solution assembles a group of versions called the solution's **elements**.

A solution is automatically assembled using cross-references. By scanning cross-references, a solution automatically includes all dependent objects required for a particular object. For example, when adding a project to a solution, versions for all the models used in this project's interfaces are automatically checked in and added to the solution. You can also manually add or remove elements into and from the solution.

Solutions are displayed in the Solutions accordion in Designer Navigator and in Operator Navigator.

The following objects may be added into solutions:

- Projects
- Models, Model Folders
- Scenarios
- Load Plans
- Global Variables, Knowledge Modules, User Functions and Sequences.

To create a solution:

1. In Designer Navigator or Operator Navigator, from the Solutions toolbar menu select **New Solution**.
2. In the Solutions editor, enter the **Name** of your solution and a **Description**.
3. From the File menu select **Save**.

The resulting solution is an empty shell into which elements may then be added.

Working with Elements in a Solution

This section details the different actions that can be performed when working with elements of a solution.

Adding Elements

To add an element, drag the object from the tree into the Elements list in the solution 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 solution, 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 solution, select the elements you want to restore and then click the Restore button. The elements selected are all restored from the solution's versions.

Synchronizing Solutions

Synchronizing a solution automatically adds required elements that have not yet been included in the solution, creates new versions of modified elements and automatically removes unnecessary elements. The synchronization process brings the content of the solution up to date with the elements (projects, models, etc) stored in the repository.

To synchronize a solution:

1. Open the solution 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 solution is up to date, then a message appears. Otherwise, a list of elements to add or remove from the solution 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 solution. Version creation windows may appear for elements requiring a new version to be created.

You should synchronize your solutions regularly to keep the solution contents up-to-date. You should also do it before checking in a solution version.

Restoring and Checking in a Solution

The procedure for checking in and restoring a solution 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 solution to import scenarios into production in Operator Navigator or Designer Navigator.

To restore a scenario from a solution:

1. Double-click a solution to open the Solution 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 19-3.

Note: When restoring a solution, elements in the solution are not automatically restored. They must be restored manually from the Solution editor.

Importing and Exporting Solutions

Solutions can be exported and imported similarly to other objects in Oracle Data Integrator. Export/Import is used to transfer solutions from one master repository to another. Refer to Chapter 20, "Exporting and Importing," for more information.

20 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

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)

Oracle Data Integrator 12*c* introduces the use of globally-unique object identifiers. Unlike previous versions of ODI, in ODI 12*c*, 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 11*g* compatibility mode.

For more information about 11*g* compatibility mode, see: Chapter 12, "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 12*c*. In order to ensure that ODI 11*g* 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 11*g* 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-12*c* 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*.

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."
- 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 20–1 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 20–1 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
■ (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	■ Technology Codes ■ Context Codes ■ Logical Schema Names ■ Data Type Codes ■ Physical Server Names of the Optimization Contexts of Mappings

Import Types

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.

Import Type	Description
Duplication	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. 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 12*c* Repositories. 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. Note that this mode is designed to insert only 'new' elements. 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. 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.

Import Type	Description
Synonym Mode INSERT	Tries to insert the same object (with the same GUID) into the target repository. The original object GUID is preserved. If an object of the same type with the same internal ID already exists then nothing is inserted. 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. If any of the incoming attributes violates any referential constraints, the import operation is aborted and an error message is thrown. Note that sessions can only be imported in this mode.
Synonym Mode UPDATE	Tries to modify the same object (with the same GUID) in the repository. This import type updates the objects already existing in the target Repository with the content of the export file. If the object does not exist, the object is not imported. 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.
Synonym Mode INSERT_ UPDATE	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. 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. 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. 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.

Import Type	Description
Import Replace	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. 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. Note the following when using the Import Replace mode: 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. **Warnings**: ■ 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.

Tips for Import/Export

This section provides tips for the import and export operations.

Repository IDs

When importing ODI 11*g* 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 12*c*. 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 12, "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 20-11 and "Working with Solutions" on page 19-7.

Import Type

Choose the import type carefully. See "Import Types" on page 20-3 for more information.

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 20-2. 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

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 20-2. 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.

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.

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 Types" on page 20-3 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.

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 20–2.

Table 20–2 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	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 20-7 for more details. Note that when you are exporting a Load Plan, scenarios will not be exported even if you check this option.
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.
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	XML Version specified in the export file. Parameter .xml version in the XML file header. `<?xml version="1.0" encoding="ISO-8859-1"?>`
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

   You must at least specify the **Export Name**.

4. Click **OK**.

The object is exported as an XML file in the specified location.

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 Solutions" on page 19-7 or "Smart Export and Import" on page 20-11.

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 20–2.

   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 Solution or the Smart Import. See "Working with Solutions" on page 19-7 and "Smart Export and Import" on page 20-11 for more information.

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 9-4 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 Types" on page 20-3 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 11*g* Objects into a 12*c* Environment

Any versionable 11*g*object can be imported into a 12*c* 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
- Solution
- Load Plan

When importing objects, you must define an Upgrade Key. ODI uses this key to generate a unique GUID for the objects.

Note: 11*g* interfaces can only be imported into a 12*c* 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 Types" on page 20-3 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.

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.

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 Types" on page 20-3 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 Types" on page 20-3 for more information.
   2. Enter the **File Import Directory**.
   3. Select the file(s) to import from the list.
4. Click **OK**.

The global KM is now available in all your projects.

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 solution for every broken link or conflict detected during the Smart Import

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:
   1. In the Export Name field, enter the name given to the export (mandatory). Default is `SmartExport.xml`.
   2. 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.
   3. 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

   4. 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.

Note:

- 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.

5. 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 20-14 for more information.
6. 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.

5. 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 20-15 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 20–3 details the different toolbar components.

Table 20–3 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.

Table 20–3 (Cont.) Smart Export Toolbar

Icon	Name	Description
🔖	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.

Performing a Smart Import

Note: When performing a Smart Import of ODI 11*g* 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-12*c* export file. For more information, see: Chapter 12, "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:
   1. In the **File Selection** field, enter the location of the Smart Export file to import.
   2. Optionally, select a response file to replay a previous Smart Import wizard execution by presetting all fields from the **Response File** field.
   3. Click **Next** to move to the next step of the Smart Import Wizard.

      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.
   1. 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.
   2. In the **Action** column, select the action to perform on the object during the import operation. Possible values are listed in Table 20–4.

Table 20–4 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.

Table 20–4 (Cont.) Actions during Import

Action	Description
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.

3. In the **Repository Object** column, select the required repository object. This is the repository object that matches the best the import object.
4. 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.

5. 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 20–5 describes the possible actions.

Table 20–5 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.
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 solution for every issue. However, missing references may still result during the actual import process depending on the choices you made for the import actions.

6. 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.
7. 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.
   1. In the **File Selection** field, verify the import file name.

2. 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**.
3. 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.
4. 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.

Repository-Level Export/Import

At repository level you can export and import the master repository and the work repositories.

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 20–2, " 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:
    1. Select the **Import Type**. Refer to "Import Types" on page 20-3 for more information.
    2. Select whether you want to import the files **From a Folder** or **From a ZIP file.**
    3. Enter the file import folder or zip file.
5. Click **OK**.

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 11*g*-compatible repositories, because 12*c* repositories have a unique Global ID.

If the target 11*g*-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 12-2 for more information on the risks, and how to renumber a repository is described in "Renumbering Master Repositories" on page 12-3.

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 11*g* 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 Table 25–2, " MBean Server Parameters".

   Refer to the "Setting Up External Password Storage" on page 25-14 for more information on password storage details.
10. In the Master Repository Import Wizard click **Finish** to validate your entries.

A new repository is created and the exported components are imported in this master repository.

Export/Import the Topology and Security Settings

Exporting then importing the topology or security 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 20–2, " 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 Types" on page 20-3 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**.

The specified files are imported into the master repository.

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 inTable 20–2, " 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 Types" on page 20-3 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**.

The specified files are imported into the work repository.

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 20–6:

Table 20–6 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. ■ **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**.

Exporting and Importing the Log

You can export and import log data for archiving purposes. See "Exporting and Importing Log Data" on page 23-12 for more information.

Part VI

Running and Monitoring Integration Processes

This part describes how to run and monitor integration processes.

This part contains the following chapters:

21

Running Integration Processes

This chapter describes how to run and schedule integration processes.

This chapter includes the following sections:

Understanding ODI Executions

An *execution* takes place when an integration task needs to be performed by Oracle Data Integrator. This integration task may be one of the following:

- An operation on a model, sub-model or a datastore, such as a customized reverse-engineering, a journalizing operation or a static check started from the Oracle Data Integrator Studio
- The execution of a design-time object, such as a mapping, a package or a procedure, typically started from the Oracle Data Integrator Studio
- The execution of a run-time scenario or a Load Plan that was launched from the Oracle Data Integrator Studio, from a command line, via a schedule or a web service interface

Oracle Data Integrator generates the code for an execution in the form of a *session* or in the form of a *Load Plan run* if a Load Plan is executed.

A run-time *Agent* processes this code and connects to the sources and targets to perform the data integration. These sources and targets are located by the Agent using a given execution *context*.

When an execution is started from Oracle Data Integrator Studio, the Execution Dialog is displayed. This dialog contains the execution parameters listed in Table 21–1.

Table 21–1 Execution Parameters

Properties	Description
Context	The context into which the session is started.
Agent	The agent which will execute the mapping. The object can also be executed using the agent that is built into Oracle Data Integrator Studio, by selecting **Local (No Agent)**.
Log Level	Level of logging information to retain. All session tasks with a defined log level lower than or equal to this value will be kept in the Session log when the session completes. However, if the object execution ends abnormally, all tasks will be kept, regardless of this setting. Note that log level 6 has the same behavior as log level 5, but with in addition of variable tracking. See "Tracking Variables and Sequences" on page 13-21 for more information.
Simulation	Check Simulation if you want to simulate the execution and create an execution report. Refer to "Simulating an Execution" on page 21-20 for more information.

Session Lifecycle

This section describes the session lifecycle. See "Introduction to Load Plans" on page 15-1 for more information on Load Plan runs and the Load Plan life cycle.

The lifecycle of a session is as follows:

1. An execution request is sent to the agent, or the agent triggers an execution from a schedule.

   Note that if the execution is triggered from Oracle Data Integrator Studio on a design-time object (mapping, package, etc.), Studio pre-generates in the work repository the code for the session before sending the request. If the execution is started from a scenario, this phase is not necessary as the scenario already contains pre-generated code.

2. The agent completes code generation for the session: It uses the context provided to resolve the physical information such as data server connections and fully qualified tables names. This resulting code is written into the work repository as a session in *Waiting* status.

3. The agent initializes the connections to the source and target data servers that are required for the execution of the session.

4. The agent acknowledges the execution request. If the execution was started from the Studio, the Session Started Dialog is displayed.

5. The agent executes each of the tasks contained in this session, using the capabilities of the database servers, operating systems, or scripting engines to run the code contained in the session's tasks.

6. While processing the session, the agent updates the execution log in the repository, reports execution statistics and error messages.

   Once the session is started, you can monitor it in the log, using for example Operator Navigator. Refer to Chapter 23, "Monitoring Integration Processes," for more information on session monitoring.

7. When the session completes, tasks are preserved or removed from the log according to the *log level* value provided when starting for this session.

Note: A Session is always identified by a unique *Session Number* (or *Session ID*). This number can be viewed when monitoring the session, and is also returned by the command line or web service interfaces when starting a session.

When starting an execution from other locations such as a command line or a web service, you provide similar execution parameters, and receive a similar *Session Started* feedback. If the session is started synchronously from a command line or web service interface, the command line or web service will wait until the session completes, and provide the session return code and an error message, if any.

Executing Mappings, Procedures, Packages and Model Operations

Mappings, procedures, and packages are design-time objects that can be executed from the Designer Navigator of Oracle Data Integrator Studio:

- For more information on mappings execution, refer to "Running Mappings" on page 11-9.
- For more information on procedures execution, refer to "Using Procedures" on page 13-10.
- For more information on packages execution, refer to "Running a Package" on page 10-10.
- For more information on model operations, refer to "Creating and Reverse-Engineering a Model" on page 5-3, "Checking Data Quality in a Model" on page 5-11 and "Setting up Journalizing" on page 6-2.

Executing a Scenario

Scenarios can be executed in several ways:

- Executing a Scenario from ODI Studio
- Executing a Scenario from a Command Line.
- From a Web Service. See "Executing a Scenario Using a Web Service" on page 21-21 for more information.
- From ODI Console. See "Managing Scenarios and Sessions" on page 24-5.

Note: Before running a scenario, you need to have the scenario generated from Designer Navigator or imported from a file. Refer to Chapter 14, "Using Scenarios," for more information.

Executing a Scenario from ODI Studio

You can start a scenario from Oracle Data Integrator Studio from Designer or Operator Navigator.

To start a scenario from Oracle Data Integrator Studio:

1. Select the scenario in the **Projects** accordion (in Designer Navigator) or the **Scenarios** accordion (in Operator Navigator).
2. Right-click, then select **Run**.
3. In the **Run** dialog, set the execution parameters. Refer to Table 21–1, " Execution Parameters" for more information. To execute the scenario with the agent that is built into Oracle Data Integrator Studio, select **Local (No Agent)**.
4. Click **OK**.
5. If the scenario uses variables as parameters, the Variable values dialog is displayed. Select the values for the session variables. Selecting **Latest value** for a variable uses its current value, or default value if none is available.

When the agent has started to process the session, the **Session Started** dialog appears.

Executing a Scenario from a Command Line

You can start a scenario from a command line.

Before executing a scenario from a command line, read carefully the following requirements:

- The command line scripts, which are required for performing the tasks described in this section, are only available if you have installed the Oracle Data Integrator Standalone Agent. See *Installing and Configuring Oracle Data Integrator* for information about how to install the Standalone Agent.
- To use this command the connection to your repository must be configured in the odiparams file. See "Managing Agents" on page 4-14 for more information.
- When starting a scenario from a command line, the session is not started by default against a remote run-time agent, but is executed by a local Java process started from the command line. This process can be aborted locally, but cannot receive a session stop signal as it is not a real run-time agent. As a consequence, sessions started this way cannot be stopped remotely.

  This process will be identified in the Data Integrator log after the `Local Agent` name. You can change this name using the `NAME` parameter.

  If you want to start the session against a run-time agent, you must use the `AGENT_URL` parameter.

To start a scenario from a command line:

1. Change directory to the `/agent/bin` directory of the Oracle Data Integrator installation.
2. Enter the following command to start a scenario.

   On UNIX systems:

   ```
   ./startscen.sh <scenario_name> <scenario_version> <context_code> [<log_
   level>] [-AGENT_URL=<remote_agent_url>] [-ASYNC=yes|no] [-NAME=<local_
   agent_name>] [-SESSION_NAME=<session_name>] [-KEYWORDS=<keywords>]
   [<variable>=<value>]*
   ```

   On Windows systems:

   ```
   startscen.bat <scenario_name> <scenario_version> <context_code> [<log_
   level>] [-AGENT_URL=<remote_agent_url>][-ASYNC=yes|no] ["-NAME=<local_
   agent_name>"] ["-SESSION_NAME=<session_name>"] ["-KEYWORDS=<keywords>"]
   ["<variable>=<value>"]*
   ```

Note: On Windows platforms, it is necessary to "delimit" the command arguments containing "=" signs or spaces, by using double quotes. The command call may differ from the Unix command call. For example:

On Unix

```
./startscen.sh DWH 001 GLOBAL SESSION_NAME=MICHIGAN
```

On Windows

```
startscen.bat DWH 001 GLOBAL "SESSION_NAME=MICHIGAN"
```

Table 21–2 lists the different parameters, both mandatory and optional. The parameters are preceded by the "-" character and the possible values are preceded by the "=" character. You must follow the character protection syntax specific to the operating system on which you enter the command.

Table 21–2 Startscen command Parameters

Parameters	Description
`<scenario_name>`	Name of the scenario (mandatory).
`<scenario_version>`	Version of the scenario (mandatory). If the version specified is -1, the latest version of the scenario is executed.
`<context_code>`	Code of the execution context (mandatory).
`[<log_level>]`	Level of logging information to retain. This parameter is in the format `<n>` where `<n>` is the expected logging level, between 0 and 6. The default log level is 5. Note that log level 6 has the same behavior as log level 5, but with in addition of variable tracking. See "Tracking Variables and Sequences" on page 13-21 for more information. Example: `startscen.bat SCENAR 1 GLOBAL 5`
`[-AGENT_URL=<remote_agent_url>`	URL of the run-time agent that will run this session. If this parameter is set, then `NAME` parameter is ignored.
[-ASYNC=yes|no]	Set to yes, for an asynchronous execution on the remote agent. If `ASYNC` is used, `AGENT_URL` is manadatory. Note that when the asynchronous execution is used, the session ID of the scenario is returned.

Table 21–2 (Cont.) Startscen command Parameters

Parameters	Description
`[-NAME=<local_agent_name>]`	Agent name that will appear in the execution log for this session, instead of `Local Agent`. This parameter is ignored if `AGENT_URL` is used. Note that using an existing physical agent name in the `NAME` parameter is not recommended. The run-time agent whose name is used does not have all the information about this session and will not be able to manage it correctly. The following features will not work correctly for this session: ■ Clean stale session: This session will be considered as stale by this agent if this agent is started. The session will be pushed to error when the agent will detect this session ■ Kill Sessions: This agent cannot kill the session when requested. ■ Agent Session Count: This session is counted in this agent's sessions, even if it is not executed by it. It is recommended to use a `NAME` that does not match any existing physical agent name. If you want to start a session on a given physical agent, you must use the `AGENT_URL` parameter instead.
`[-SESSION_NAME=<session_name>]`	Name of the session that will appear in the execution log. If not specified, the scenario name is used as the session name.
`[-KEYWORDS=<keywords>]`	List of keywords attached to this session. These keywords make session identification easier. The list is a comma-separated list of keywords.
`[<variable>=<value>]`	Allows to assign a `<value>` to a `<variable>` for the execution of the scenario. `<variable>` is either a project or global variable. Project variables should be named `<Project Code>.<Variable Name>`. Global variables should be called `GLOBAL.<variable Name>`. This parameter can be repeated to assign several variables. Do not use a hash sign (`#`) to prefix the variable name on the startscen command line.

Restarting a Session

Any session that has encountered an error, or has been stopped by the user can be restarted.

Oracle Data Integrator uses JDBC transactions when interacting with source and target data servers, and any open transaction state is not persisted when a session finishes in error state. The appropriate restart point is the task that started the unfinished transaction(s). If such a restart point is not identifiable, it is recommended that you start a fresh session by executing the scenario instead of restarting existing sessions that are in error state.

Only sessions in status **Error** or **Waiting** can be restarted. By default, a session restarts from the last task that failed to execute (typically a task in error or in waiting state). A session may need to be restarted in order to proceed with existing staging tables and avoid re-running long loading phases. In that case the user should take into consideration transaction management, which is KM specific. A general guideline is: If a crash occurs during a loading task, you can restart from the loading task that failed. If a crash occurs during an integration phase, restart from the first integration task, because integration into the target is within a transaction. This guideline applies only to one mapping at a time. If several mappings are chained and only the last one performs the commit, then they should all be restarted because the transaction runs over several mappings.

To restart from a specific task or step:

1. In Operator Navigator, navigate to this task or step, edit it and switch it to Waiting state.
2. Set all tasks and steps after this one in the Operator tree view to Waiting state.
3. Restart the session using one of the following methods:
   - Restarting a Session from ODI Studio
   - Restarting a Session from a Command Line.
   - From a Web Service. See "Restarting a Session Using a Web Service" on page 21-23 for more information.
   - From ODI Console. See "Managing Scenarios and Sessions" on page 24-5.

   WARNING: When restarting a session, all connections and transactions to the source and target systems are re-created, and not recovered from the previous session run. As a consequence, uncommitted operations on transactions from the previous run are not applied, and data required for successfully continuing the session may not be present.

Restarting a Session from ODI Studio

To restart a session from Oracle Data Integrator Studio:

1. In Operator Navigator, select the session that you want to restart.
2. Right-click and select **Restart**.
3. In the **Restart Session** dialog, specify the agent you want to use for running the new session.

   To select the agent to execute the session, do one of the following:
   - Select **Use the previous agent: <agent name>** to use the agent that was used for the previous session execution.
   - Select **Choose another agent** to select from the list the agent that you want to use for the session execution.

   Note: Select **Internal** to use the ODI Studio built-in Agent.

4. Select the Log Level. Note that log level 6 has the same behavior as log level 5, but with in addition of variable tracking. See "Tracking Variables and Sequences" on page 13-21 for more information.
5. Click **OK** to restart the indicated session and to close the dialog. Click **Cancel** if you do not want to restart session.

When Oracle Data Integrator has restarted the session, the **Session Started** dialog appears.

Restarting a Session from a Command Line

Before restarting a session from a command line, read carefully the following requirements:

- The command line scripts, which are required for performing the tasks described in this section, are only available if you have installed the Oracle Data Integrator Standalone Agent. See *Installing and Configuring Oracle Data Integrator* for information about how to install the Standalone Agent.
- To use this command the connection to your repository must be configured in the odiparams file. See "Managing Agents" on page 4-14 for more information.
- When restarting a session from a command line, the session is not started by default against a remote run-time agent, but is executed by a local Java process started from the command line. This process can be aborted locally, but cannot receive a session stop signal as it is not a real run-time agent. As a consequence, sessions started this way cannot be stopped remotely.

  If you want to start the session against a run-time agent, you must use the `AGENT_URL` parameter.

To restart a session from a command line:

1. Change directory to the `/agent/bin` directory of the Oracle Data Integrator installation.
2. Enter the following command to start a scenario.

   On UNIX systems:

   ```
   ./restartsession.sh <session_number> [-log_level][-AGENT_URL=<remote_
   agent_url>]
   ```

   On Windows systems:

   ```
   restartsession.bat <session_number> [-log_level]["-AGENT_URL=<remote_
   agent_url>"]
   ```

Table 21–3 lists the different parameters of this command, both mandatory and optional. The parameters are preceded by the "-" character and the possible values are preceded by the "=" character. You must follow the character protection syntax specific to the operating system on which you enter the command.

Table 21–3 restartsess command Parameters

Parameters	Description
`<session_number>`	Number (ID) of the session to be restarted.
`[-log_level]`	Level of logging information to retain. Note that log level 6 has the same behavior as log level 5, but with in addition of variable tracking. Note that if this log_level parameter is not provided when restarting a session, the previous log level used for executing the session will be reused. See "Tracking Variables and Sequences" on page 13-21 for more information.
`[-AGENT_URL=<remote_agent_url>]`	URL of the run-time agent that will restart this session. By default the session is executed by a local Java process started from the command line.

Note: To use this command the connection to your repository must be configured in the odiparams file. See "Managing Agents" on page 4-14 for more information.

Note: When restarting a session from a command line, the session is not started by default against a remote run-time agent, but is executed by a local Java process started from the command line.

If you want to start the session against a run-time agent, you must use the `AGENT_URL` parameter.

Stopping a Session

Any running or waiting session can be stopped. You may want to stop a session when you realize that for example your mapping contains errors or when the execution takes a long time.

Note that there are two ways to stop a session:

- **Normal**: The session is stopped once the current task is finished.
- **Immediate**: The current task is immediately interrupted and the session is stopped. This mode allows to stop long-running tasks, as for example long SQL statements before they complete.

Note: The immediate stop works only with technologies and drivers that support task interruption. It is supported if the `statement.cancel` method is implemented in the JDBC driver.

Note: Only sessions that are running within a Java EE or standalone Agent can be stopped. Sessions running in the Studio built-in Agent or started with the `startscen.sh` or `startscen.bat` script without the `AGENT_URL` parameter, cannot be stopped. See "Executing a Scenario" on page 21-3 for more information.

Session can be stopped in several ways:

- Stopping a Session From ODI Studio
- Stopping a Session From a Command Line.
- From ODI Console. See "Managing Scenarios and Sessions" on page 24-5.

Stopping a Session From ODI Studio

To stop a session from Oracle Data Integrator Studio:

1. In Operator Navigator, select the running or waiting session to stop from the tree.
2. Right-click then select **Stop Normal** or **Stop Immediate**.
3. In the Stop Session Dialog, click **OK**.

The session is stopped and changed to *Error* status.

Stopping a Session From a Command Line

Before stopping a session from a command line, read carefully the following requirements:

- The command line scripts, which are required for performing the tasks described in this section, are only available if you have installed the Oracle Data Integrator Standalone Agent. See *Installing and Configuring Oracle Data Integrator* for information about how to install the Standalone Agent.
- To use this command the connection to your repository must be configured in the odiparams file. See "Managing Agents" on page 4-14 for more information.

To stop a session from a command line:

1. Change directory to the `/agent/bin` directory of the Oracle Data Integrator installation.
2. Enter the following command to start a scenario.

   On UNIX systems:

   ```
   ./stopsession.sh <session_id> [-AGENT_URL=<remote_agent_url>] [-STOP_
   LEVEL=<normal (default) | immediate>]
   ```

   On Windows systems:

   ```
   stopsession.bat <session_id> ["-AGENT_URL=<remote_agent_url>"] ["-STOP_
   LEVEL=<normal (default) | immediate>"]
   ```

Table 21–4 lists the different parameters of this command, both mandatory and optional. The parameters are preceded by the "-" character and the possible values are preceded by the "=" character. You must follow the character protection syntax specific to the operating system on which you enter the command.

Table 21–4 StopSession command Parameters

Parameters	Description
`<session_id>`	Number (ID) of the session to be restarted.
`[-AGENT_URL=<remote_agent_url>`	URL of the run-time agent that stops this session. By default the session is executed by a local Java process started from the command line.
`[-STOP_LEVEL=<normal (default) | immediate>]`	The level used to stop a running session. If it is omitted, `normal` will be used as the default stop level.

Note: To use this command the connection to your repository must be configured in the odiparams file. See "Managing Agents" on page 4-14 for more information.

Executing a Load Plan

Load Plans can be executed in several ways:

- Executing a Load Plan from ODI Studio
- Executing a Load Plan from a Command Line
- From a Web Service. See "Executing a Load Plan Using a Web Service" on page 21-23 for more information.
- From ODI Console. See "Managing Load Plans" on page 24-8.

Note: A Load Plan cannot be executed using the ODI Studio built-in agent called *Local (No Agent)*.

Executing a Load Plan from ODI Studio

In ODI Studio, you can run a Load Plan in Designer Navigator or in Operator Navigator.

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 **Run**.
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 step.
   - Select the **Log Level.** All sessions with a defined log level lower than or equal to this value will be kept in the Session log when the session completes. However, if the object execution ends abnormally, all tasks will be kept, regardless of this setting.

     Note that log level 6 has the same behavior as log level 5, but with in addition of variable tracking. See "Tracking Variables and Sequences" on page 13-21 for more information.

     Select **Use Session Task Log Level** (default) to use the Session Tasks Log Level value defined in 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 Window** appears.
6. Click **OK**.

A new execution of the Load Plan is started: a Load Plan instance is created and also the first Load Plan run. You can review the Load Plan execution in the Operator Navigator.

Executing a Load Plan from a Command Line

You can start a Load Plan from a command line.

Before executing a Load Plan from a command line, read carefully the following requirements:

- The command line scripts, which are required for performing the tasks described in this section, are only available if you have installed the Oracle Data Integrator Standalone Agent. See *Installing and Configuring Oracle Data Integrator* for information about how to install the Standalone Agent.
- To use this command, the connection to your repository must be configured in the odiparams file. See "Managing Agents" on page 4-14 for more information.
- A Load Plan Run is started against a run-time agent identified by the `AGENT_URL` parameter.

To start a Load Plan from a command line:

1. Change directory to `/agent/bin` directory of the Oracle Data Integrator installation.
2. Enter the following command to start a Load Plan.

   On UNIX systems:

   ```
   ./startloadplan.sh <load_plan_name> <context_code> [log_level] -AGENT_
   URL=<agent_url> [-KEYWORDS=<keywords>] [<variable>=<value>]*
   ```

   On WINDOWS systems:

   ```
   startloadplan.bat <load_plan_name> <context_code> [log_level]"-AGENT_
   URL=<agent_url>" ["-KEYWORDS=<keywords>"] ["<variable>=<value>"]*
   ```

   Note: On Windows platforms, it is necessary to "delimit" the command arguments containing "=" signs or spaces, by using double quotes. The command call may differ from the Unix command call. For example:

   On UNIX systems:

   ```
   ./startloadplan.sh DWLoadPlan DEV -AGENT_
   URL=http://localhost:20910/oraclediagent
   ```

   On WINDOWS systems:

   ```
   startloadplan.bat DWLoadPlan DEV "-AGENT_
   URL=http://localhost:20910/oraclediagent"
   ```

Table 21–5 lists the different parameters, both mandatory and optional. The parameters are preceded by the "-" character and the possible values are preceded by the "=" character. You must follow the character protection syntax specific to the operating system on which you enter the command.

Table 21–5 Startloadplan Command Parameters

Parameters	Description
`<load_plan_name>`	Name of the Load Plan to be started (mandatory)
`<context_code>`	Code of the context used for starting the Load Plan. Note that if this value is not provided, the Load Plan uses the context of the session that calls it (mandatory)
`[log_level]`	Level of logging information to retain. All sessions with a defined log level lower than or equal to this value will be kept in the Session log when the session completes. However, if the object execution ends abnormally, all tasks will be kept, regardless of this setting. Note that log level 6 has the same behavior as log level 5, but with in addition of variable tracking. Default is the Load Plan's Session Tasks Log Level that has been used for starting the Load Plan. See "Tracking Variables and Sequences" on page 13-21 for more information.
`["-AGENT_URL=<agent_url>"]`	URL of the Physical Agent that starts the Load Plan (mandatory)
`["-KEYWORDS=<Keywords>"]`	Keywords to improve the organization of ODI logs by session folders and automatic classification. Enter a comma separated list of keywords that will be attached to this Load Plan.

Table 21–5 (Cont.) Startloadplan Command Parameters

Parameters	Description
`["variable>=<value> "]`	Startup values for the Load Plan variables (optional). Note that project variables should be named `<project_code>.<variable_name>` and global variables should be named `GLOBAL.<variable_name>`. This list is of the form `<variable>=<value>`. The format for Date and Number variables is as follows: ▪ **Date**: `yyyy-MM-dd'T'HH:mm:ssZ` For example: `2009-12-06T15:59:34+0100` ▪ **Number**: Integer value For example: `29833` For example: `"A_PROJ.A_REFRESH_VAR=bb" "A_PROJ.A_CROSS_PROJ_VAR=aa" "A_PROJ.A_VAR=cc"`

Restarting a Load Plan Run

Restarting a Load Plan, starts a new run for the selected Load Plan instance. Note that when a Load Plan restarts the Restart Type parameter for the steps in error defines how the Load Plan and child sessions will be restarted. See "Defining the Restart Behavior" on page 15-17 and "Restarting a Session" on page 21-6 for more information.

Note: Restarting a Load Plan instance depends on the status of its most-recent (highest-numbered) run. Restart is only enabled for the most-recent run, if its status is Error.

Load Plans can be restarted in several ways:

- Restarting a Load Plan from ODI Studio
- Restarting a Load Plan from a Command Line
- From a Web Service. See "Restarting a Load Plan Instance Using a Web Service" on page 21-25 for more information.
- From ODI Console. See "Managing Load Plans" on page 24-8.

Restarting a Load Plan from ODI Studio

To restart a Load Plan from ODI Studio:

1. In Operator Navigator, select the Load Plan Run to restart from the Load Plan Executions accordion.
2. Right-click then select **Restart**.
3. In the Restart Load Plan Dialog, select the Agent that restarts the Load Plan. Optionally, select a different log level.
4. Click **OK**.

The Load Plan is restarted and a new Load Plan run is created.

Restarting a Load Plan from a Command Line

Before restarting a Load Plan from a command line, read carefully the following requirements:

- The command line scripts, which are required for performing the tasks described in this section, are only available if you have installed the Oracle Data Integrator Standalone Agent. See *Installing and Configuring Oracle Data Integrator* for information about how to install the Standalone Agent.
- To use this command the connection to your repository must be configured in the odiparams file. See "Managing Agents" on page 4-14 for more information.
- A Load Plan Run is restarted against a remote run-time agent identified by the `AGENT_URL` parameter.

To restart a Load Plan from a command line:

1. Change directory to `/agent/bin` directory of the Oracle Data Integrator installation.
2. Enter the following command to restart a Load Plan.

   On UNIX systems:

   ```
   ./restartloadplan.sh <load_plan_instance_id> [log_level] -AGENT_
   URL=<agent_url>
   ```

   On WINDOWS systems:

   ```
   restartloadplan.bat <load_plan_instance_id> [log_level] "-AGENT_
   URL=<agent_url>"
   ```

> **Note:** On Windows platforms, it is necessary to "delimit" the command arguments containing "=" signs or spaces, by using double quotes. The command call may differ from the Unix command call.

Table 21–6 lists the different parameters, both mandatory and optional. The parameters are preceded by the "-" character and the possible values are preceded by the "=" character. You must follow the character protection syntax specific to the operating system on which you enter the command.

Table 21–6 Restartloadplan Command Parameters

Parameters	**Description**
<load_plan_instance_id>	ID of the stopped or failed Load Plan instance that is to be restarted (mandatory)
[log_level]	Level of logging information to retain. All sessions with a defined log level lower than or equal to this value will be kept in the Session log when the session completes. However, if the object execution ends abnormally, all tasks will be kept, regardless of this setting. Note that log level 6 has the same behavior as log level 5, but with in addition of variable tracking. Default is the log level value used for the Load Plan's previous run. See "Tracking Variables and Sequences" on page 13-21 for more information.
["-AGENT_URL=<agent_url>"]	URL of the Physical Agent that starts the Load Plan (optional)

Stopping a Load Plan Run

Any running or waiting Load Plan Run can be stopped. You may want to stop a Load Plan Run when you realize that for example your Load Plan contains errors or when the execution takes a long time.

Note that there are two ways to stop a Load Plan Run:

- **Stop Normal**: In normal stop mode, the agent in charge of stopping the Load Plan sends a Stop Normal signal to each agent running a session for this Load Plan. Each agent will wait for the completion of the current task of the session and then end the session in error. Exception steps will not be executed by the Load Plan and once all exceptions are finished the load plan is moved to an error state.
- **Stop Immediate**: In immediate stop mode, the agent in charge of stopping the Load Plan sends a Stop immediate signal to each agent running a session for this Load Plan. Each agent will immediately end the session in error and *not* wait for the completion of the current task of the session. Exception steps will not be executed by the Load Plan and once all exceptions are finished the load plan is moved to an error state.

Load Plans can be stopped in several ways:

- Stopping a Load Plan from ODI Studio
- Stopping a Load Plan Run from a Command Line
- From a Web Service. See "Stopping a Load Plan Run Using a Web Service" on page 21-24 for more information.
- From ODI Console. See "Managing Load Plans" on page 24-8.

Stopping a Load Plan from ODI Studio

To stop a Load Plan Run from ODI Studio:

1. In Operator Navigator, select the running or waiting Load Plan Run to stop from the Load Plan Executions accordion.
2. Right-click then select **Stop Normal** or **Stop Immediate**.
3. In the Stop Load Plan Dialog, select the Agent that stops the Load Plan.
4. Click **OK**.

The Load Plan run is stopped and changed to *Error* status.

Stopping a Load Plan Run from a Command Line

Before stopping a Load Plan from a command line, read carefully the following requirements:

- The command line scripts, which are required for performing the tasks described in this section, are only available if you have installed the Oracle Data Integrator Standalone Agent. See *Installing and Configuring Oracle Data Integrator* for information about how to install the Standalone Agent.
- To use this command the connection to your repository must be configured in the odiparams file. See "Managing Agents" on page 4-14 for more information.
- A Load Plan Run signal is sent by a remote run-time agent identified by the `AGENT_URL` parameter.

To stop a Load Plan run from a command line:

1. Change directory to `/agent/bin` directory of the Oracle Data Integrator installation.
2. Enter the following command to start a Load Plan.

   On UNIX systems:

   ```
   ./stoploadplan.sh <load_plan_instance_id> [<load_plan_run_count>]
   -AGENT_URL=<agent_url> [-STOP_LEVEL=<normal (default) | immediate>]
   ```

   On WINDOWS systems:

   ```
   stoploadplan.bat <load_plan_instance_id> [<load_plan_run_count>]
   "-AGENT_URL=<agent_url>" ["-STOP_LEVEL=<normal (default) | immediate>"]
   ```

Table 21–7 lists the different parameters, both mandatory and optional. The parameters are preceded by the "-" character and the possible values are preceded by the "=" character. You must follow the character protection syntax specific to the operating system on which you enter the command.

Table 21–7 Stoploadplan Command Parameters

Parameters	Description
<load_plan_instance_id>	ID of the running Load Plan run that is to be stopped (mandatory)
[<load_plan_run_count>]	Load Plan run count of the load plan instance. It prevents unintentional stopping of a load plan run that happens to be the latest one. If it is omitted, the last Load Plan run count will be used (optional)
["-AGENT_URL=<agent_url>"]	URL of the Physical Agent that starts the Load Plan (optional)
[-STOP_LEVEL=<normal (default) | immediate>]	Level used to stop the Load Plan run. Default is `normal`.

Note: On Windows platforms, it is necessary to "delimit" the command arguments containing "=" signs or spaces, by using double quotes. The command call may differ from the Unix command call.

Scheduling Scenarios and Load Plans

You can schedule the executions of your scenarios and Load Plans using the Oracle Data Integrator built-in scheduler or an external scheduler. Both methods are detailed in this section:

- Scheduling a Scenario or a Load Plan with the Built-in Scheduler
- Scheduling a Scenario or a Load Plan with an External Scheduler

Scheduling a Scenario or a Load Plan with the Built-in Scheduler

You can attach schedules to scenarios and also to Load Plans. Such schedules are managed by the scheduler built-in run-time agent.

It is important to understand that a schedule concerns only one scenario or one Load Plan, while a scenario or a Load Plan can have several schedules and can be scheduled in several ways. The different schedules appear under the **Scheduling** node of the scenario or Load Plan. Each schedule allows a start date and a repetition cycle to be specified.

For example:

- **Schedule 1**: Every Thursday at 9 PM, once only.
- **Schedule 2**: Every day from 8 am to 12 noon, repeated every 5 seconds.
- **Schedule 3**: Every day from 2 PM to 6 PM, repeated every 5 seconds, with a maximum cycle duration of 5 hours.

Scheduling a Scenario or a Load Plan

To schedule a scenario or a Load Plan from Oracle Data Integrator Studio.

1. Right-click the **Scheduling** node under a scenario or a Load Plan in the Designer or Operator Navigator.
2. Select **New Scheduling**. The Scheduling editor is displayed.
3. On the **Definition** tab of the Scheduling editor specify the parameters as follows:

Properties	Description
Context	Context into which the scenario or Load Plan is started.
Agent	Agent executing the scenario or Load Plan.
Log Level	Level of logging information to retain.

The **Status** parameters define the activation of the schedule.

Properties	Description
Active	The scheduling will be active when the agent is restarted or when the scheduling of the physical agent is updated.
Inactive	The schedule is not active and will not run.
Active for the period	Activity range of the schedule. A schedule active for a period of time will only run within this given period.

The **Execution** parameters define the frequency of execution for each execution cycle.

Properties	Description
Execution	Frequency of execution option (annual, monthly,... simple). This option is completed by a set of options that depend on this main option.

4. On the **Execution Cycle** tab, specify the parameters for the repeat mode of the scenario as follows:

Properties	Description
None (Execute once)	The scenario or Load Plan is executed only one time.

Properties	Description
Many times	The scenario or Load Plan is repeated several times. ▪ **Maximum number of repetitions**: The maximum number of times the scenario is repeated during the cycle. ▪ **Maximum Cycle Duration:** As soon as the maximum time is reached, the scenario is no longer restarted, and the cycle stops. ▪ **Interval between repetitions:** The downtime between each scenario execution.
Constraints	Allows limitations to be placed on one cycle iteration, in the event of a problem during execution. ▪ **Number of Attempts on Failure**: Maximum number of consecutive execution attempts for one iteration. ▪ **Stop Execution After**: Maximum execution time for one iteration. If this time is reached, the scenario or Load Plan is automatically stopped.

5. On the **Variables** tab, unselect **Latest Value** for variables for which you want to provide a **Value**. Only variables used in the scenario or Load Plan and flagged as parameters for this scenario or Load Plan appear in this tab.
6. From the **File** menu, click **Save**.

The new schedule appears under the **Scheduling** node of the scenario or Load Plan.

The schedule changes are taken into account by the run-time agent when it starts or when it receives a schedule update request.

Updating an Agent's Schedule

An agent reads schedules when starting on all the repositories attached to the master repository it connects to. It is possible, if a schedule was added for this agent in a given repository, to refresh the agent schedule.

To update an agent's schedule:

1. In Topology Navigator expand the **Agents** node in the **Physical Architecture** accordion.
2. Select the Physical Agent you want to update the schedule.
3. Right-click and select **Update Scheduling...**
4. In the **Select Repositories** dialog, select the repositories from which you want to read scheduling information. Check **Select All Work Repositories** to read scheduling information from all these repositories.
5. Click **OK**.

The agent refreshes and re-computes its in-memory schedule from the schedules defined in these repositories.

You can also use the OdiUpdateAgentSchedule tool (see: "OdiUpdateAgentSchedule" on page A-80) to update an agent's schedule.

Displaying the Schedule

You can view the scheduled tasks of all your agents or you can view the scheduled tasks of one particular agent.

Note: The Scheduling Information is retrieved from the Agent's in-memory schedule. The Agent must be started and its schedule refreshed in order to display accurate schedule information.

Displaying the Schedule for All Agent

To display the schedule for all agents:

1. Select **Connect Navigator >Scheduling...** from the Operator Navigator toolbar menu.

The **View Schedule** dialog appears, displaying the schedule for all agents.

Displaying the Schedule for One Agent

To display the schedule for one agent:

1. In Topology Navigator expand the **Agents** node in the **Physical Architecture** accordion.
2. Select the Physical Agent you want to update the schedule.
3. Right-click and select **View Schedule**.

The **Schedule** Editor appears, displaying the schedule for this agent.

Note: The Scheduling Information is retrieved from the Agent's schedule. The Agent must be started and its schedule refreshed in order to display accurate schedule information.

Using the View Schedule Dialog

The schedule is displayed in form of a Gantt diagram. Table 21–8 lists the details of the **Schedule** dialog.

Table 21–8 Scheduling Details

Parameters	Description
Selected Agent	Agent for which the Schedule is displayed. You can display also the schedule of all agents by selecting **All Agents**.
Selected Work Repository	Only the scenarios executed in the selected Work Repository are displayed in the schedule. Default is **All Work Repositories**.
Scheduling from... to...	Time range for which the scheduling is displayed. Click **Refresh** to refresh this schedule.
Update	Click **Update** to update the schedule for the selected agent(s)
Time Range	The time range specified (1 hour, 2 hours, and so forth) allows you to center the diagram on the current time plus this duration. This feature provides a vision of the sessions in progress plus the incoming sessions. You can use the arrows to move the range forward or backward.
Scenarios details	This panel displays the details and execution statistics for each scheduled scenario.

If you select a zone in the diagram (keep the mouse button pressed), you automatically zoom on the select zone.

By right-clicking in the diagram, you open a context menu for zooming, saving the diagram as an image file, printing or editing the display properties.

Scheduling a Scenario or a Load Plan with an External Scheduler

To start a scenario or a Load Plan with an external scheduler, do one of the following:

- Use the *startscen* or *startloadplan* command from the external scheduler
- Use the web service interface for triggering the scenario or Load Plan execution

For more information, see:

- "Executing a Scenario from a Command Line" on page 21-4
- "Executing a Load Plan from a Command Line" on page 21-11
- "Executing a Scenario Using a Web Service" on page 21-21
- "Executing a Load Plan Using a Web Service" on page 21-23

If a scenario or a Load Plan completes successfully, the return code will be 0. If not, the return code will be different than 0. This code will be available in:

- The return code of the command line call. The error message, if any, is available on the standard error output.
- The SOAP response of the web service call. The web service response includes also the session error message, if any.

Simulating an Execution

In Oracle Data Integrator you have the possibility at design-time to simulate an execution. Simulating an execution generates and displays the code corresponding to the execution without running this code. Execution simulation provides reports suitable for code review.

Note: No session is created in the log when the execution is started in simulation mode.

To simulate an execution:

1. In the **Project** view of the Designer Navigator, select the object you want to execute.
2. Right-click and select **Run**.
3. In the **Run** dialog, set the execution parameters and select **Simulation**. See Table 21–1, " Execution Parameters" for more information.
4. Click **OK**.

The Simulation report is displayed.

You can click **Save** to save the report as `.xml` or `.html` file.

Managing Executions Using Web Services

This section explains how to use a web service to perform run-time operations. It contains the following sections.

- Introduction to Run-Time Web Services

- Executing a Scenario Using a Web Service
- Monitoring a Session Status Using a Web Service
- Restarting a Session Using a Web Service
- Executing a Load Plan Using a Web Service
- Stopping a Load Plan Run Using a Web Service
- Restarting a Load Plan Instance Using a Web Service
- Monitoring a Load Plan Run Status Using a Web Service
- Listing Contexts Using a Web Service (Deprecated)
- Listing Scenarios Using a Web Service (Deprecated)
- Accessing the Web Service from a Command Line
- Using the Run-Time Web Services with External Authentication
- Using WS-Addressing
- Using Asynchronous Web Services with Callback

Introduction to Run-Time Web Services

Oracle Data Integrator includes web services for performing run-time operations. These web services are located in two places:

- In the run-time agent, a web service allows starting a scenario or a Load Plan, monitoring a session status or a Load Plan run status, and restarting a session or a Load Plan instance, as well as stopping a Load Plan run. To use operations from this web service, you must first install and configure a standalone or a Java EE agent.
- A dedicated public web service component provides operations to list the contexts and scenarios available. To use operations from this web service, you must first install and configure this component in a Java EE container.

The following applies to the SOAP request used against the agent and public web services

- The web services operations accept password in a plain text in the SOAP request. Consequently, it is strongly recommended to use secured protocols (HTTPS) to invoke web services over a non-secured network. You can alternately use external authentication. See "Using the Run-Time Web Services with External Authentication" on page 21-29 for more information.
- Repository connection information is not necessary in the SOAP request as the agent or public web service component is configured to connect to a master repository. Only an ODI user and the name of a work repository are required to run most of the operations.

Executing a Scenario Using a Web Service

The `invokeStartScen` operation of the agent web service starts a scenario in synchronous or asynchronous mode; in a given work repository. The session is executed by the agent providing the web service.

```
<OdiStartScenRequest>
   <Credentials>
      <OdiUser>odi_user</OdiUser>
      <OdiPassword>odi_password</OdiPassword>
```

```
            <WorkRepository>work_repository</WorkRepository>
         </Credentials>
         <Request>
            <ScenarioName>scenario_name</ScenarioName>
            <ScenarioVersion>scenario_version</ScenarioVersion>
            <Context>context</Context>
            <LogLevel>log_level</LogLevel>
            <Synchronous>synchronous</Synchronous>
            <SessionName>session_name</SessionName>
            <Keywords>session_name</Keywords>
            <Variables>
            <Name>variable_name</name>
            <Value>variable_value</Value>
            </Variables>
         </Request>
      </OdiStartScenRequest>
```

The scenario execution returns the session ID in a response that depends on the value of the `synchronous` element in the request.

- In synchronous mode (`Synchronous`=1), the response is returned once the session has completed, and reflects the execution result.
- In asynchronous mode (`Synchronous`=0), the response is returned once the session is started, and only indicates the fact whether the session was correctly started or not.

This operation returns a response in the following format:

```
<?xml version = '1.0' encoding = 'ISO-8859-1'?>
<ns2:OdiStartScenResponse xmlns:ns2="xmlns.oracle.com/odi/OdiInvoke/">
   <Session>543001</Session>
</ns2:OdiStartScenResponse>
```

Monitoring a Session Status Using a Web Service

The `getSessionStatus` operation of the agent web service returns the status of one or more sessions in a given repository, identified by their Session Numbers provided in the `SessionIds` element. It manages both running and completed sessions.

```
<OdiGetSessionsStatusRequest>
   <Credentials>
      <OdiUser>odi_user</OdiUser>
      <OdiPassword>odi_password</OdiPassword>
      <WorkRepository>work_repository</WorkRepository
   </Credentials>
   <SessionIds>session_number</SessionIds>
</OdiGetSessionsStatusRequest>
```

This operation returns a response in the following format:

```
<SessionStatusResponse>
   <SessionId>session_id</SessionId>
   <SessionStatus>status_code</SessionStatus>
   <SessionReturnCode>return_code</SessionReturnCode>
</SessionStatusResponse>
```

The Return Code value is zero for successful sessions and possible status codes are:

- D: Done

- E: Error
- M: Warning
- Q: Queued
- R: Running
- W: Waiting

Restarting a Session Using a Web Service

The `invokeRestartSess` operation of the agent web service restarts a session identified by its session number (provided in the `SessionID` element) in a given work repository. The session is executed by the agent providing the web service.

Only sessions in status **Error** or **Waiting** can be restarted. The session will resume from the last non-completed task (typically, the one in error).

Note that you can change the value of the variables or use the `KeepVariables` boolean element to reuse variables values from the previous session run.

```
<invokeRestartSessRequest>
   <Credentials>
      <OdiUser>odi_user</OdiUser>
      <OdiPassword>odi_password</OdiPassword>
      <WorkRepository>work_repository</WorkRepository>
   </Credentials>
   <Request>
      <SessionID>session_number</SessionID>
      <Synchronous>synchronous</Synchronous>
      <KeepVariables>0|1</KeepVariables>
      <LogLevel>log_level</LogLevel>
      <Variables>
      <Name>variable_name</name>
      <Value>variable_value</Value>
      </Variables>
   </Request>
</invokeRestartSessRequest>
```

This operation returns a response similar to `InvokeStartScen`, depending on the `Synchronous` element's value.

Executing a Load Plan Using a Web Service

The `invokeStartLoadPlan` operation of the agent web service starts a Load Plan in a given work repository. The Load Plan is executed by the agent providing the web service. Note the following concerning the parameters of the `invokeStartLoadPlan` operation:

- `OdiPassword`: Use a password in clear text.
- `Context`: Use the context code.
- `Keywords`: If you use several keywords, enter a comma separated list of keywords.
- `Name`: Use the fully qualified name for variables: `GLOBAL.variable_name` or `PROJECT_CODE.variable_name`

The following shows the format of the OdiStartLoadPlanRequest.

```
<OdiStartLoadPlanRequest>
   <Credentials>
      <OdiUser>odi_user</OdiUser>
      <OdiPassword>odi_password</OdiPassword>
      <WorkRepository>work_repository</WorkRepository>
   </Credentials>
   <StartLoadPlanRequest>
      <LoadPlanName>load_plan_name</LoadPlanName>
      <Context>context</Context>
      <Keywords>keywords</Keywords>
      <LogLevel>log_level</LogLevel>
      <LoadPlanStartupParameters>
         <Name>variable_name</Name>
         <Value>variable_value</Value>
      </LoadPlanStartupParameters>
    </StartLoadPlanRequest>
</OdiStartLoadPlanRequest>
```

The `invokeStartLoadPlan` operation returns the following values in the response:

- Load Plan Run ID
- Load Plan Run Count
- Master Repository ID
- Master Repository timestamp

The following is an example of an `OdiStartLoadPlan` response:

```
<?xml version = '1.0' encoding = 'UTF8'?>
<ns2:OdiStartLoadPlanResponse xmlns:ns2="xmlns.oracle.com/odi/OdiInvoke/">
   <executionInfo>
      <StartedRunInformation>
         <OdiLoadPlanInstanceId>2001</OdiLoadPlanInstanceId>
         <RunCount>1</RunCount>
         <MasterRepositoryId>0</MasterRepositoryId>
         <MasterRepositoryTimestamp>1290196542926</MasterRepositoryTimestamp>
      </StartedRunInformation>
   </executionInfo>
</ns2:OdiStartLoadPlanResponse>
```

Stopping a Load Plan Run Using a Web Service

The `invokeStopLoadPlan` operation of the agent web service stops a running Load Plan run identified by the Instance ID and Run Number in a given work repository. The Load Plan instance is stopped by the agent providing the web service. Note that the `StopLevel` parameter can take the following values:

- `NORMAL`: Waits until the current task finishes and then stops the session.
- `IMMEDIATE`: Stops the session immediately, cancels all open statements and then rolls back the transactions.

See "Stopping a Load Plan Run" on page 21-15 for more information on how to stop a Load Plan run and "Executing a Load Plan Using a Web Service" on page 21-23 for more information on the other parameters used by the `invokeStopLoadPlan` operation.

```
<OdiStopLoadPlanRequest>
   <Credentials>
      <OdiUser>odi_user</OdiUser>
      <OdiPassword>odi_password</OdiPassword>
      <WorkRepository>work_repository</WorkRepository>
   </Credentials>
   <OdiStopLoadPlanRequest>
      <LoadPlanInstanceId>load_plan_instance_id</LoadPlanInstanceId>
      <LoadPlanInstanceRunCount>load_plan_run_count</LoadPlanInstanceRunCount>
      <StopLevel>stop_level</StopLevel>
   </OdiStopLoadPlanRequest>
</OdiStopLoadPlanRequest>
```

The `invokeStopLoadPlan` operation returns the following values in the response:

- Load Plan Run ID
- Load Plan Run Count
- Master Repository ID
- Master Repository timestamp

The following is an example of an `OdiStopLoadPlan` response:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:OdiStopLoadPlanResponse xmlns:ns2="xmlns.oracle.com/odi/OdiInvoke/">
         <executionInfo>
            <StoppedRunInformation>
              <OdiLoadPlanInstanceId>3001</OdiLoadPlanInstanceId>
              <RunCount>1</RunCount>
              <MasterRepositoryId>0</MasterRepositoryId>
              <MasterRepositoryTimestamp>1290196542926</MasterRepositoryTimestamp>
            </StoppedRunInformation>
         </executionInfo>
      </ns2:OdiStopLoadPlanResponse>
   </S:Body>
</S:Envelope>
```

Restarting a Load Plan Instance Using a Web Service

The `invokeRestartLoadPlan` operation of the agent web service restarts a Load Plan instance identified by the Instance ID in a given work repository. The Load Plan instance is restarted by the agent providing the web service.

```
<OdiRestartLoadPlanRequest>
   <Credentials>
      <OdiUser>odi_user</OdiUser>
      <OdiPassword>odi_password</OdiPassword>
      <WorkRepository>work_repository</WorkRepository>
   </Credentials>
   <RestartLoadPlanRequest>
      <LoadPlanInstanceId>load_plan_instance_id</LoadPlanInstanceId>
      <LogLevel>log_level</LogLevel>
   </RestartLoadPlanRequest>
</OdiRestartLoadPlanRequest>
```

Monitoring a Load Plan Run Status Using a Web Service

The `getLoadPlanStatus` operation of the agent web service returns the status of one or more Load Plans by their Instance ID and Run Number in a given repository. It manages both running and completed Load Plan instances.

```
<OdiGetLoadPlanStatusRequest>
   <Credentials>
       <OdiUser>odi_user</OdiUser>
       <OdiPassword>odi_password</OdiPassword>
      <WorkRepository>work_repository</WorkRepository>
   </Credentials>
   <LoadPlans>
      <LoadPlanInstanceId>load_plan_instance_id</LoadPlanInstanceId>
      <LoadPlanRunNumber>load_plan_run_number</LoadPlanRunNumber>
   </LoadPlans>
</OdiGetLoadPlanStatusRequest>
```

The `getStopLoadPlan`Status operation returns the following values in the response:

- Load Plan Run ID
- Load Plan Run Count
- Load Plan Run return code
- Load Plan message

The following is an example of an `OdiGetLoadPlanStatus` response:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns2:OdiGetLoadPlanStatusResponse
xmlns:ns2="xmlns.oracle.com/odi/OdiInvoke/">
         <LoadPlanStatusResponse>
            <LoadPlanInstanceId>3001</LoadPlanInstanceId>
            <LoadPlanRunNumber>1</LoadPlanRunNumber>
            <LoadPlanStatus>E</LoadPlanStatus>
            <LoadPlanReturnCode>ODI-1530</LoadPlanReturnCode>
            <LoadPlanMessage>ODI-1530: Load plan instance was stopped by user
request.</LoadPlanMessage>
         </LoadPlanStatusResponse>
      </ns2:OdiGetLoadPlanStatusResponse>
   </S:Body>
</S:Envelope>
```

Listing Contexts Using a Web Service (Deprecated)

Note: The `listContext` operation of the public web service is deprecated in 11.1.1.7 and will be removed in future release.

The `listContext` operation of the public web service lists of all the contexts present in a master repository.

```
   <listContextRequest>
      <OdiUser>odi_user</OdiUser>
      <OdiPassword>odi_password</OdiPassword>
   <listContextRequest>
```

Listing Scenarios Using a Web Service (Deprecated)

Note: The `listScenario` operation of the public web service is deprecated in 11.1.1.7 and will be removed in future release.

The `listScenario` operation of the public web service lists of all the scenarios present in a given work repository.

```
   <listScenarioRequest>
      <OdiUser>odi_user</OdiUser>
```

```
        <OdiPassword>odi_password</OdiPassword>
        <WorkRepository>work_repository</WorkRepository>
    <listScenarioRequest>
```

Accessing the Web Service from a Command Line

Oracle Data Integrator contains two shell scripts for UNIX platforms that use the web service interface for starting and monitoring scenarios from a command line via the run-time agent web service operations:

- `startscenremote.sh` starts a scenario on a remote agent on its web service. This scenario can be started synchronously or asynchronously. When started asynchronously, it is possible to have the script polling regularly for the session status until the session completes or a timeout is reached.
- `getsessionstatusremote.sh` gets the status of a session via the web service interface. This second script is used in the `startscenremote.sh` script.

Before accessing a web service from a command line, read carefully the following important notes:

- The command line scripts, which are required for performing the tasks described in this section, are only available if you have installed the Oracle Data Integrator Standalone Agent. See *Installing and Configuring Oracle Data Integrator* for information about how to install the Standalone Agent.
- Unlike the `startscen.sh` command line, these scripts rely on the lightweight WGET utility installed with the UNIX or Linux platform to perform the web service calls. It does not use any java code and uses a polling mechanism to reduce the number of running processes on the machine. These scripts are suitable when a large number of scenarios and sessions need to be managed simultaneously from a command line.

Starting a Scenario

To start a scenario from a command line via the web service:

1. Change directory to the `/agent/bin` directory of the Oracle Data Integrator installation.
2. Enter the following command to start a scenario.

   On UNIX systems:

   ```
   ./startscenremote.sh <scenario_name> <scenario_version> <context_code>
   <work_repository> <remote_agent_url> <odi_user> <odi_password> -l <log_
   level> -s <sync_mode> -n <session_name> -k <session_keyword> -a
   <assign_variable> -t <timeout> -i <interval> -h <http_timeout> -v
   ```

Table 21–9 lists the different parameters of this command, both mandatory and optional.

Table 21–9 Startscenremote command Parameters

Parameters	Description
`<scenario_name>`	Name of the scenario (mandatory).
`<scenario_version>`	Version of the scenario (mandatory). If the version specified is -1, the latest version of the scenario is executed.
`<context_code>`	Code of the execution context (mandatory).
`<work_repository>`	Name of the work repository containing the scenario.

Table 21–9 (Cont.) Startscenremote command Parameters

Parameters	Description
`<remote_agent_url>`	URL of the run-time agent that will run this session.
`<odi_user>`	Name of the user used to run this sessions.
`<odi_password>`	This user's password.
`-l <log_level>`	Level of logging information to retain. This parameter is in the format `<n>` where `<n>` is the expected logging level, between 0 and 6. The default log level is 5. Note that log level 6 has the same behavior as log level 5, but with in addition of variable tracking. See "Tracking Variables and Sequences" on page 13-21 for more information. Example: `startscen.bat SCENAR 1 GLOBAL 5`
`-s <sync_mode>`	Execution mode: ■ 0: Synchronous ■ 1:Asynchronous (Do not wait for session completion) ■ 2: Asynchronous (Wait for session completion).
`-n <session_name>`	Name of the session
`-k <session_keyword>`	List of keywords attached to this session. These keywords make session identification easier. The list is a comma-separated list of keywords.
`-a <assign_variable>`	Assign variable. Allows to assign a `<value>` to a `<variable>` for the execution of the scenario. `<variable>` is either a project or global variable. Project variables should be named `<Project Code>.<Variable Name>`. Global variables should be called `GLOBAL.<variable Name>`. This parameter can be repeated to assign several variables. Do not use a hash sign (#) to prefix the variable name on the startscen command line. For Example: `-a PROJ1.VAR1=100`
`-t <timeout>`	Timeout in seconds for waiting for session to complete if sync_ mode = 2.
`-i <interval>`	Polling interval for session status if sync_mode = 2.
`-h <http_timeout>`	HTTP timeout for the web services calls.
`-v`	Verbose mode.

Monitoring a Session Status

To monitor the status of a session from a command line via the web service:

1. Change directory to the `/agent/bin` directory of the Oracle Data Integrator installation.
2. Enter the following command to start a scenario.

   On UNIX systems:

   ```
   ./getsessionstatusremote.sh <session_number> <work_repository> <remote_
   agent_url> <odi_user> <odi_password> -w <wait_mode> -t <timeout> -i
   <interval> -h <http_timeout> -v
   ```

Table 21–10 lists the different parameters of this command, both mandatory and optional.

Table 21–10 GetSessionStatusRemote command Parameters

Parameters	Description
`<session_number>`	Number of the session to monitor.
`<work_repository>`	Name of the work repository containing the scenario.
`<remote_agent_url>`	URL of the run-time agent that will run this session.
`<odi_user>`	Name of the user used to run this sessions.
`<odi_password>`	This user's password.
`-w <wait_mode>`	Wait mode: ■ 0: Do not wait for session completion, report current status. ■ 1: Wait for session completion then report status.
`-t <timeout>`	Timeout in seconds for waiting for session to complete if sync_mode = 2.
`-i <interval>`	Polling interval for session status if sync_mode = 2.
`-h <http_timeout>`	HTTP timeout for the web services calls.
`-v`	Verbose mode.

Using the Run-Time Web Services with External Authentication

The web services examples in this chapter use an ODI authentication within the SOAP body, using the *OdiUser* and *OdiPassword* elements.

When external authentication is set up for the repository and container based authentication with Oracle Platform Security Services (OPSS) is configured (See "Configuring External Authentication" on page 25-17 for more information), the authentication can be passed to the web service using HTTP basic authentication, WS-Security headers, SAML tokens and so forth. OPSS will transparently handle the authentication on the server-side with the identity provider. In such situation, the *OdiUser* and *OdiPassword* elements can be omitted.

The run-time web services will first try to authenticate using OPSS. If no authentication parameters have been provided, OPSS uses anonymous user and the *OdiUser* and *OdiPassword* are checked. Otherwise (this is in case of invalid credentials to OPSS) OPSS throws an authentication exception and the web service is not invoked.

Note: OPSS authentication is only possible for a Public Web Service or JEE Agent deployed in a Oracle WebLogic Server.

Using WS-Addressing

The web services described in this chapter optionally support WS-Addressing. WS-Addressing allows replying on an endpoint when a run-time web service call completes. For this purpose, two endpoints, *ReplyTo* and *FaultTo*, can be optionally specified in the SOAP request header.

These endpoints are used in the following way:

- When the run-time web service call completes successfully, the result of an *Action* is sent to the *ReplyTo* endpoint.
- If an error is encountered with the SOAP request or if Oracle Data Integrator is unable to execute the request, a message is sent to the *FaultTo* address. If the

FaultTo address has not been specified, the error is sent to the *ReplyTo* address instead.

- If the Oracle Data Integrator Agent encounters errors while processing the request and needs to raise an ODI error message, this error message is sent back to the *ReplyTo* address.

Note that callback operations do not operate in callback mode unless a valid *ReplyTo* address is specified.

The following is an example of a request that is sent to retrieve the session status for session 20001:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:odi="xmlns.oracle.com/odi/OdiInvoke/">
<soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
<wsa:Action
soapenv:mustUnderstand="1">xmlns.oracle.com/odi/OdiInvoke/getSessionStatus</wsa:Ac
tion>
<wsa:ReplyTo soapenv:mustUnderstand="1">
<wsa:Address>http://host001:8080/examples/servlets/servlet/RequestPrinter</wsa:Add
ress>
</wsa:ReplyTo>
<wsa:MessageID
soapenv:mustUnderstand="1">uuid:71bd2037-fbef-4e1c-a991-4afcd8cb2b8e</wsa:MessageI
D>
</soapenv:Header>
   <soapenv:Body>
      <odi:OdiGetSessionsStatusRequest>
         <Credentials>
            <!--You may enter the following 3 items in any order-->
            <OdiUser></OdiUser>
            <OdiPassword></OdiPassword>
            <WorkRepository>WORKREP1</WorkRepository>
         </Credentials>
         <!--Zero or more repetitions:-->
         <SessionIds>20001</SessionIds>
      </odi:OdiGetSessionsStatusRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

The following call will be made to the *ReplyTo* address (`http://host001:8080/examples/servlets/servlet/RequestPrinter`).

Note that this call contains the response to the *Action* specified in the request, and includes the original *MessageID* to correlate request and response.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header>
<To xmlns="http://www.w3.org/2005/08/addressing">http://
host001:8080/examples/servlets/servlet/RequestPrinter</To>
<Action
xmlns="http://www.w3.org/2005/08/addressing">xmlns.oracle.com/odi/OdiInvoke/:reque
stPortType:getSessionStatusResponse</Action>
<MessageID
xmlns="http://www.w3.org/2005/08/addressing">uuid:eda383f4-3cb5-4dc2-988c-a4f70517
63ea</MessageID>
<RelatesTo
xmlns="http://www.w3.org/2005/08/addressing">uuid:71bd2037-fbef-4e1c-a991-4afcd8cb
2b8e</RelatesTo>
</S:Header>
```

```
<S:Body>
<ns2:OdiGetSessionsStatusResponse xmlns:ns2="xmlns.oracle.com/odi/OdiInvoke/">
<SessionStatusResponse>
               <SessionId>26001</SessionId>
               <SessionStatus>D</SessionStatus>
               <SessionReturnCode>0</SessionReturnCode>
           </SessionStatusResponse>
</ns2:OdiGetSessionsStatusResponse>
    </S:Body>
</S:Envelope>
```

For more information on WS-Adressing, visit these World Wide Web Consortium (W3C) web sites at the following URLs:

- Web Services Addressing:
  `http://www.w3.org/Submission/ws-addressing/`
- WS-Addressing SOAP Binding:
  `http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/`
- WS-Addressing WSDL Binding:
  `http://www.w3.org/TR/2006/WD-ws-addr-wsdl-20060216/`

Using Asynchronous Web Services with Callback

Long-running web service operations can be started asynchronously following the pattern of JRF asynchronous web services or asynchronous BPEL processes. These follow a "request-response port pair" pattern.

In this pattern, the web service client implements a callback operation. When the server completes the operation requested by the client, it sends the result to this callback operation.

Two specific operations in the agent web service support this pattern: *invokeStartScenWithCallback* and *invokeRestartSessWithCallback*.

These operations provide the following features:

- They do not return any response. These are one way operations.
- The client invoking these two operation must implement respectively the *invokeStartSceCallback* and *invokeRestartSessCallback* one way operations. Results from the *invokeStartScenWithCallback* and *invokeRestartSessWithCallback* actions are sent to these operations.
- The invocation should provide in the SOAP header the *ReplyTo* and possibly *FaultTo* addresses. If the methods are invoked without a *ReplyTo* address, the operation will execute synchronously (which corresponds to a *invokeStartScen* or *invokeRestartSess* operation). When a fault is generated in the operation, it will be sent to the *ReplyTo* address or *FaultTo* address.

A scenario or session started synchronously using the *invokeStartScenWithCallback* and *invokeRestartSessWithCallback* will start and will not return any SOAP response, as they are one way operations. When the session completes, the response is sent the callback address.

Note: Oracle BPEL takes care of automatically implementing these operations and sends out WS-Addressing headers that point to these endpoints.

22

Debugging Integration Processes

This chapter describes how to use the Oracle Data Integrator debugger to solve problems with your integration processes.

This chapter includes the following sections:

About Sessions and Blueprints

Oracle Data Integrator Studio provides a graphical debugger that allows you to manually step through the steps and tasks of a session. You can use the debugger to identify problems with mappings, procedures, packages, or scenarios. Debugging is performed in the blueprint of an active session.

You can only see a blueprint on the tab of the properties of a session. You will see the same blueprint in different sessions based on a given mapping.

Once you edit a mapping, new sessions run based on it will not use a previously cached blueprint: instead, a new blueprint is generated. Every time a mapping is executed, a new blueprint is generated afresh. Only in the case of scenarios do you retain the same blueprint for multiple runs as long as the scenario is not modified. Once the scenario is modified, then a new blueprint is generated for the subsequent sessions.

Blueprint Source and Target Code

You cannot edit the blueprint directly, but you can edit the Source/Target code for the task that the debugger is currently on in the Session editor. The edited code is used for all sessions run from a mapping. Even if you delete all of the sessions, the cached blueprint survives and will be reused when you next start a session from the unchanged mapping.

Introduction to Debugging in the Session Editor

In the blueprint in the session editor you can debug a running session or restart a completed session, by manually stepping through the steps and tasks of the session. The blueprint includes a Steps Hierarchy table, which identifies steps and tasks by an ID. Other capabilities include:

- Setting breakpoints for steps and tasks. Breakpoints are shown graphically in the blueprint, and listed in the Debug Breakpoints view.
- Viewing and updating/editing source and target code for any task within the session, and running queries to view or select data.
- Viewing the values of variables as well as uncommitted data.
- Viewing all active debug sessions, and disconnecting or connecting to debuggable sessions.
- Viewing all threads for the connected session.

Icons

The debug execution can be controlled by the debug commands in the debug toolbar:

Icon	Command	Description
●	Add Breakpoint	Toggles a breakpoint at the currently selected line in the blueprint, currently selected task in the procedure, or currently selected step in a package.
🐞	Start Debug Session	Starts a debugging session for the editor in focus. In a session editor, this command can be used to restart a session.
🐞	Connect To Debug Session	Connects to a debugging session running on an agent. This opens the currently running session and allow to step through the execution. It is not possible to connect to a local agent.
🐞	Disconnect Debug Session	Disconnects the current debugging session. After the debugging session is disconnected, the execution continues and any breakpoint is ignored. It is possible to reconnect to a disconnected session. It is not possible to disconnect from a session running on a local agent.
➔	Current Cursor	Highlights the current cursor in the blueprint, and opens the session editor, if not already open.
★	Get Data	Inserts the SQL code of the currently selected task into the SQL command field of the Debug Data window. Both Source Task Data and Target Task Data windows are populated.
📄	Step into	Steps to the beginning of the first child node of the currently selected node. If the currently selected node does not have a child, this is disabled.
📄	Run to Task End	Runs to the end of the currently selected task. If the task has children, execute all children until the end of the task is reached. This is disabled if it is not at the beginning of a task.

Icon	Command	Description
	Run to Next Task	Runs to the beginning of the next task. If at the last task in a list of tasks, go to the end of the parent task.
	Run to Step End	Runs to the end of the current step. Executes all tasks until the end of the step is reached.
	Run to Next Step	Runs to the beginning of the next step.
	Pause	Suspends the current execution. The execution can be continued by stepping through the steps/tasks, or by resuming execution.
	Resume	Resumes execution at the current cursor and continues until the session is finished or a breakpoint is reached.

Differences between Debugging Packages, Mappings, and Procedures

There are differences in where you can set breakpoints when debugging packages, mappings, and procedures:

- Breakpoints cannot be set on a mapping.
- In a procedure, breakpoints can be set on tasks
- In a package, you can set breakpoints on a step.

Starting a Debugging Session

You can start a debug session for an object by accessing a **Debug** command in the context menu or toolbar. Starting a debug session launches the Start Debug Session Dialog. See "Debug Session Options".

Starting a Session from the Toolbar

To start a debug session from the toolbar in Oracle Data Integrator Studio:

1. In the **Projects** view, select a mapping, package, procedure, or scenario.
2. From the toolbar, select **Start Debug Session**.
3. In the **Debug** window, configure options described in "Debug Session Options".
4. Click OK.

   An information message indicates that the session is started.

Starting a Session from the Navigator Tree

To start a debug session from the navigator tree in Oracle Data Integrator Studio:

1. In the navigator tree, select a mapping, package, procedure, or scenario.
2. Right-click and select **Debug**.
3. In the **Debug** window, configure options described in "Debug Session Options".
4. Click **OK**.

   An information message indicates that the session has started.

Starting a Session from the Main Menu

To start a debug session from the main menu of Oracle Data Integrator Studio:

1. Select **Run** > **Debug** > **Start Debug Session**.
2. In the **Debug** window, configure options described in "Debug Session Options".
3. Click **OK**.

   An information message indicates that the session has started.

Starting a Session for a Scenario

To start a debug session for a scenario:

1. Locate the scenario using one of the following methods:
   - In the Designer Navigator, expand the **Load Plans and Scenarios** tree and locate the scenario.
   - In the Designer Navigator, expand the **Projects** tree, and locate the scenario under the Scenarios node of the source.
   - In the **Scenarios** tab of the Mapping Editor, locate the scenario in the list of scenarios for the object.
2. Right-click the scenario and select **Debug**.
3. In the **Debug** window, configure options described in "Debug Session Options".
4. Click **OK**.

   An information message indicates that the session has started.

Connecting to a Running Debugging Session

You can connect the debugger to a debuggable session on an agent registered with the current repository. The session that the debugger connects to has to be a debuggable session and started prior to opening the **Connect to Debug Session** dialog box. It is not possible to connect to sessions running locally without an agent.

To connect to a session:

1. Select **Run** > **Debug** > **Connect to Debug Session**.
2. In the **Connect Debug Session** window, enter the following connection parameters:
   - **Agent**: Select the agent that the debuggable session runs on. The **Any Agent** choice shows all debuggable sessions from all configured agents in the session list.
   - **Session**: Shows all debuggable sessions available for connection from the agent chosen in the agent list.
3. Click **OK**.

Built-in Agents and External Agents

A debugging session is executed on a context and an agent. You can select the name of the agent that executed the debugging session. By selecting *Local (No Agent)*, you select to use the built-in agent in Oracle Data Integrator Studio.

Debug Session Lifecycle

A session is shown as running in the Operator navigator regardless of whether it is a normal or debugging session. There is no indication that the session is paused. You can get back to a stopped session run in a local agent through the Debug Sessions window.

For more information on the lifecycle of a session, see "Understanding ODI Executions" in "Chapter 21, "Running Integration Processes".

Debug Session Options

Before launching, you can configure options for the debugger session.

Suspend Before the First Task

Check *Suspend Before the First Task*, to cause the debugger to suspend the execution before the first task, even if no client is connected to the debuggable session.

If unchecked, the debug session proceeds until it hits a breakpoint or finishes.

The setting is checked by default.

Associate to Current Client

Check *Associate to Current Client* to set the debugger to start a session in debug mode and open the blueprint to debug the session.

If unchecked, the debugger starts a session in debug mode but not open the blueprint for debugging. The session can later be associated by this or other clients.

The setting is checked by default.

Delay Before Connect

Set *Delay Before Connect* to define the number of seconds before the debugger attempts a connection to the started session.

The default is 5 seconds.

Debug Descendant Sessions

Check *Debug Descendant Sessions* to prompt you to start a new debugging session whenever a Start command is executed. This setting only applies to packages or procedures, and not mappings.

If unchecked, any descendant session is executed without debugger.

The setting is unchecked by default.

Break On Error

Check *Break on Error* to cause the debugger to suspend execution at a task where an error happened. This enables you to review data within the transaction.

If unchecked, session execution engine reacts to any errors as usual. The debugger client does not provide you any notification.

The setting is unchecked by default.

Stepping through a Blueprint in the Session Editor

The blueprint shows all of the steps and task hierarchy of the selected session, and allows you to go through individual commands.

When you start a session, the session editor in its Blueprint view opens, if the *Associate to Current Client* option has been selected. Otherwise, in the debug toolbar, select **Current Cursor** to open the session editor.

Steps and Tasks

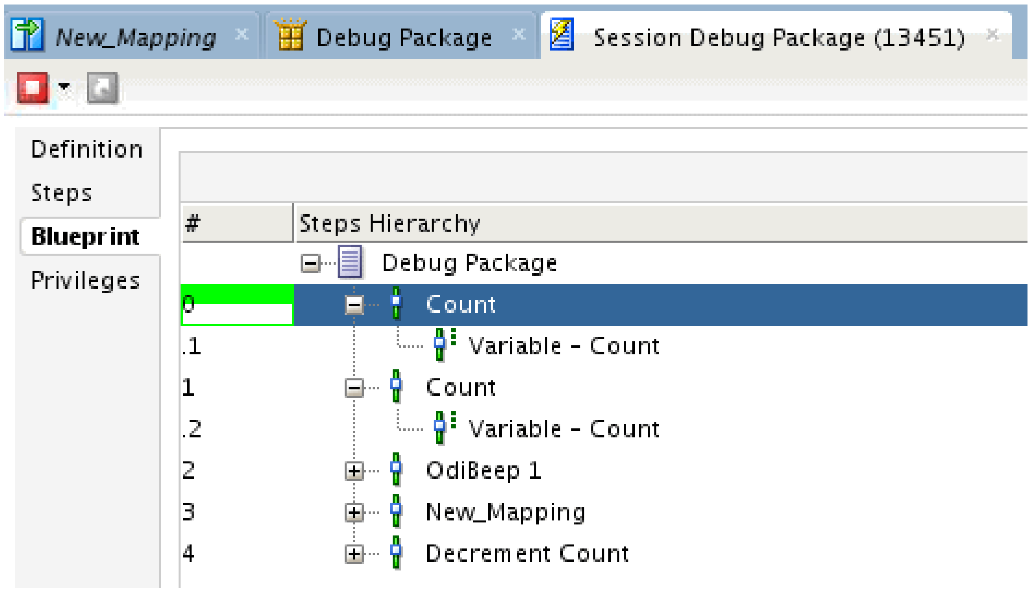
Each row in a blueprint represents a step or task. Tasks in a blueprint can be container tasks, and can execute serially or in parallel. Tasks inside a container task are leaf nodes that get executed. Sessions based on mappings have only one step, while sessions based on packages can have multiple steps.

Using the Debugging Cursor

You can place the cursor *before* or *after* the execution of a line. The debugger highlights half a cursor position before or after the line.

Figure 22–1 shows a cursor position before the execution of a line:

Figure 22–1 Cursor Positioned Before a Step



#	Steps Hierarchy
	Debug Package
0	Count
.1	Variable - Count
1	Count
.2	Variable - Count
2	OdiBeep 1
3	New_Mapping
4	Decrement Count

Debug Actions

You can perform debug actions for a selected cursor position.

Step Into

The **Step Into** action steps to the beginning of the first child node of the currently selected node. If the currently selected node does not have a child, this is disabled.

The Step Into action works for all things including steps and tasks.

Pause

The **Pause** action suspends the current execution. When the execution is paused, you can modify the code of the tasks that are yet to be executed, and that modified code is taken up when you resume the execution.

The execution can be continued by stepping through the steps/tasks, or by resuming execution.

Resume

The **Resume** action resume execution at the current cursor and continues until the session is finished or a breakpoint is reached.

Run to Task End

The **Run to Task End** action runs to the end of the currently selected task. If the task has children, execute all children until the end of the task is reached.

This action is disabled if the cursor is not at the beginning of a task.

Run to Next Task

The **Run to Next Task** action runs to the beginning of the next task. If the cursor is at the last task in a list of tasks, it goes to the end of the parent task.

Run to Step End

The **Run to Step End** action runs to the end of the current step, and executes all tasks until the end of the step is reached.

Run to Next Step

The **Run to Next Step** action runs to the beginning of the next step.

Multiple Cursors

You can use multiple cursors for in-session parallelism.

Special Behavior

A parallel container task opens a separate cursor for each child task, and a cursor that remains on the container task until the last child task has finished. Each child cursor can be used independently to step through tasks, use breakpoints, or view data or variables. The child cursor disappears when the child task is finished. Performing the **resume** command on a child cursor only progresses to the end of the child task.

Using Breakpoints

You can set breakpoints on blueprint steps and tasks to suspend execution for debugging purposes.

About the Debug Breakpoints Window

Use **Debug Breakpoints** to view the list of all breakpoints.

About Design vs. Runtime Breakpoints

There are two lists of breakpoints, *runtime breakpoints*, and *design breakpoints*.

- Runtime breakpoints can be added in the blueprint tab of a session editor. Runtime breakpoints on a blueprint are used across sessions that are started from the same blueprint. A scenario execution will reuse a blueprint as long as the scenario remains unchanged.

- Design breakpoints are defined in a design artifact such as package or procedure. When executing, a design breakpoint generates a runtime breakpoint in the blueprint it is associated with. You can add design breakpoints in the package editor.

Placing Breakpoints

To add a breakpoint:

1. Select **Run** > **Breakpoint** > **Add Breakpoint**.
2. A red dot is placed in the blueprint to represent the breakpoint.
3. Start or resume the execution.
4. When the executions suspends at the breakpoint, inspect the following:
   - debugging data. See "Debugging Variables".
   - variable values. See "Debugging Variables".
   - debugging session threads. See "Debugging Threads".

Editing Breakpoints

To edit a breakpoint:

1. Select **Run** > **Breakpoint** > **Edit Breakpoint**.
2. The Breakpoint Properties window appears.
3. Set the following properties.
   - **Enabled**: checked if the breakpoint is enabled.
     - if checked, the breakpoint is active and suspends execution.
     - if unchecked, the breakpoint is ignored during execution.
   - **Suspend after executing the task**:
     - if checked, the breakpoint suspends after executing the task or step.
     - if unchecked, the breakpoint suspends before executing the task or step.
   - **Pass count**: the number of times the breakpoint has to be passed before suspending execution; 0 suspends the execution on the initial pass, 1 passes the breakpoint once before suspending execution.
4. Click **OK**.

Removing Breakpoints

To remove a single breakpoint:

- Right-click on the step or task and select **Remove Breakpoint**.

To remove all breakpoints (available only from the breakpoint window):

- Select **Run** > **Breakpoint** > **Remove All**.

Enabling and Disabling Breakpoints

To enable or disable a single breakpoint, select:

- **Run** > **Breakpoint** > **Enable**, or,
- **Run** > **Breakpoint** > **Disable**.

To enable or disable all breakpoints (available only from the breakpoint window), select:

- **Run** > **Breakpoint** > **Enable All**, or,
- **Run** > **Breakpoint** > **Disable All**.

Pass Count

The pass count indicates the number of times the breakpoint has to be passed before suspending execution; 0 suspends the execution on every pass; 1 suspends the execution only on the first pass; 5 suspends the execution only on the fifth pass.

Debugging Data

Use the Debug Data tab to query data in the context of the current debugger task.

The data window has two tabs, Source Task Data and Target Task Data, to query the data servers associated with the source and the target for the current task. The **Get Data** command in the main toolbar inserts the SQL code of the current task into both Source Task Data and Target Task Data fields. The window provides a field to enter a SQL command and execute it by clicking the **Run SQL Code** button.

SQL commands use the transaction context of the current session, uncommitted data can be queried. If there are multiple tasks at debug execution in the session, the commands use the context of the task synced to the data window.

Get Data

Get Data inserts the SQL code of the currently selected task into the SQL command field of the Debug Data window. Both the Source Task Data and Target Task Data windows are populated.

The **Get Data** command associates with the original cursor line; if you keep stepping the data window remains attached to the original cursor line. You must press **Get Data** again to go to the new current cursor line.

Source Task Data By default the Source Task Data contains the default SQL; unparsed content of the current task. The content may be inappropriate to execute unchanged.

Target Task Data The Target Task Data typically contains a DML operation such as Insert or Update. These operations are copied unchanged from the session task, but need to be edited as the **Run SQL Code** command can only execute Select statements, not DML statements.

Run SQL Code

You are required to execute **Get Data** before doing a **Run SQL Code**. You can run queries to view or select data, but you cannot not apply changes to the data.

Editing SQL Code To edit SQL code and query data:

1. Select the **Debug Data** tab in a session.
2. Select **Get Data** to insert the SQL code of the current task into the Source Task Data and Target Task Data fields.
3. Edit the SQL command. You can use a `select` statement to query the data on the target uncommitted transaction.
4. Click **Run SQL Code**.

Debugging Variables

Use the **Debug Variables** tab to view all the variables used in the current session, and see how a variable may change as you step through the steps and tasks. You can change the value of a variable to affect the execution of the session.

You can view the following information for each variable:

Properties	Description
Name	Name of the variable.
Value	Value of the variable. Can be modified to change execution.
Type	Type of the variable.

Modifying Variables

To modify a variable:

1. Set breakpoints, if required. See "Using Breakpoints".
2. Position the cursor in the blueprint.
3. In the **Debug Variables** tab, select the variable whose value you want to modify.
4. Start or resume the execution of the session.

Debugging Threads

Use the **Debug Threads** tab to see all threads for the connected session editor in focus. The icon of the thread shows whether the thread is suspended or executing.

Go To Source

The **Go To Source** context menu command on each thread jumps to the thread location inside the blueprint.

Managing Debugging Sessions

You can see a list of all active debug sessions in the Debug Session window by selecting **Window** > **Debugger** > **Debug Sessions**.

From the Debug Sessions window, you can connect or disconnect the listed sessions.

Properties	Description
Agent	Agent executing the session.
Session Number	ID of the session.
Session Name	Name of the session.
Session Connected	Shows whether the session is connected or not connected to the debugger client.

Stop Normal and Stop Immediate

You can disconnect a session normally or immediately:

1. In the Debug Sessions window, right-click a session in the list and select **Disconnect Debug Session**.

   The session becomes active in the Studio.

2. Select the **Disconnect** button in the upper left of the session window, and choose:
   - **Stop Normal**: The session is stopped once the current task is finished.
   - **Stop Immediate**: The current task is immediately interrupted and the session is stopped. This mode allows to stop long-running tasks, as for example long SQL statements before they complete.
3. Confirm the disconnection by clicking **OK**.
4. When the session is stopped, you see the message 'Session debugging completed'. The session is removed from the Debug Sessions list.

See Stopping Sessions in "Running Integration Processes."

Restart Execution

In the session window, you can restart a stopped session.

To restart a session:

1. In the upper left of the session window, select **Restart Execution**.

   The Restart Execution window appears.
2. To select the agent to execute the session, select one of the following options:
   - **Use the Previous Agent**: **<agent name>** to use the agent that was used to execute a previous session.
   - **Choose another Agent**: to select the agent that you want to use to execute the session. Select **Internal** to use the ODI Studio built-in agent.
3. Select the Log Level. Log level 6 has the same behavior as log level 5, but with the addition of variable and sequence tracking.
4. Click **OK** to restart the indicated session and to close the dialog.

   You see an informational message, 'Session Launched', and the session is added to the Debug Sessions list.

23 Monitoring Integration Processes

This chapter describes how to manage your development executions in Operator Navigator. An overview of the Operator Navigator's user interface is provided.

This chapter includes the following sections:

- Introduction to Monitoring
- Monitoring Executions Results
- Managing your Executions

Introduction to Monitoring

Monitoring your development executions consists of viewing the execution results and managing the development executions when the executions are successful or in error. This section provides an introduction to the monitoring features in Oracle Data Integrator. How to work with your execution results is covered in "Monitoring Executions Results" on page 23-5. How to manage your development executions is covered in "Managing your Executions" on page 23-8.

Introduction to Operator Navigator

Through Operator Navigator, you can view your execution results and manage your development executions in the sessions, as well as the scenarios and Load Plans in production.

Operator Navigator stores this information in a work repository, while using the topology defined in the master repository.

Operator Navigator displays the objects available to the current user in six accordions:

- **Session List** displays all sessions organized per date, physical agent, status, keywords, and so forth
- **Hierarchical Sessions** displays the execution sessions also organized in a hierarchy with their child sessions
- **Load Plan Executions** displays the Load Plan Runs of the Load Plan instances
- **Scheduling** displays the list of physical agents and schedules
- **Load Plans and Scenarios** displays the list of scenarios and Load Plans available
- **Solutions** displays the list of solutions

The Operator Navigator Toolbar Menu

You can perform the main monitoring tasks via the Operator Navigator Toolbar menu. The Operator Navigator toolbar menu provides access to the features detailed in Table 23–1.

Table 23–1 Operator Navigator Toolbar Menu Items

Icon	Menu Item	Description
	Refresh	Click **Refresh** to refresh the trees in the Operator Navigator accordions.
	Filter	Click **Filter** to define the filters for the sessions to display in Operator Navigator.
	Filter activated	
	Auto Refresh	Click **Auto Refresh** to refresh automatically the trees in the Operator Navigator accordions.
	Connect Navigator	Click **Connect Navigator** to access the Operator Navigator toolbar menu. Through the Operator Navigator toolbar menu you can: ■ Import a scenario ■ Import and export the log ■ Perform multiple exports ■ Purge the log ■ Display the scheduling information ■ Clean stale sessions

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.

When a scenario is executed, it creates a *Session*.

Scenarios are imported into production environment and can be organized into *Load Plan and Scenario* folders. See "Managing Scenarios and Load Plans" on page 23-15 for more details.

Sessions

In Oracle Data Integrator, an execution results in a *Session*. Sessions are viewed and managed in Operator Navigator.

A *session* is an execution (of a scenario, a mapping, a package or a procedure, and so forth) undertaken by an execution agent. A session is made up of *steps* which are themselves made up of *tasks*.

A *step* is the unit of execution found between a task and a session. It corresponds to a step in a package or in a scenario. When executing a mapping or a single variable, for example, the resulting session has only one step.

Two special steps called *Command On Connect* and *Command On Disconnect* are created if you have set up On Connect and Disconnect commands on data servers used in the session. See "Setting Up On Connect/Disconnect Commands" on page 4-10 for more information.

The *task* is the smallest execution unit. It corresponds to a command in a KM, a procedure, and so forth.

Sessions can be grouped into *Session folders*. Session folders automatically group sessions that were launched with certain keywords. Refer to "Organizing the Log with Session Folders" on page 23-11 for more information.

Load Plans

A *Load Plan* is the most course-grained type of executable object in Oracle Data Integrator, used to organize and run finer-grained objects. It uses *Scenario*s in its steps. A Load Plan is an organized hierarchy of child steps. This hierarchy allows conditional processing of steps in parallel or in series.

Load Plans are imported into production environments and can be organized into *Load Plan and Scenario* folders. See "Managing Scenarios and Load Plans" on page 23-15 for more details.

Load Plan Executions

Executing a Load Plan creates a *Load Plan instance* and the first *Load Plan run* for the instance. This Load Plan instance is separated from the original Load Plan and can be modified independently. Every time a Load Plan instance is restarted, a *Load Plan run* is created for this Load Plan instance. A Load Plan run corresponds to an attempt to execute the instance. See "Load Plan Execution Lifecycle" on page 15-2 for more information.

When running, a Load Plan Run starts sessions corresponding to the scenarios sequenced in the Load Plan.

Note that in the list of Load Plan executions, only the Load Plan runs appear. Each run is identified by a Load Plan Instance ID and an Attempt (or Run) Number.

Schedules

You can *schedule* the executions of your scenarios and Load Plans using Oracle Data Integrator's built-in scheduler or an external scheduler. Both methods are detailed in "Scheduling Scenarios and Load Plans" on page 21-16.

The schedules appear in Designer and Operator Navigator under the **Scheduling** node of the scenario or Load Plan. Each schedule allows a start date and a repetition cycle to be specified.

Log

The Oracle Data Integrator log corresponds to all the Sessions and Load Plan instances/runs stored in a repository. This log can be exported, purged or filtered for monitoring. See "Managing the Log" on page 23-9 for more information.

Status

A session, step, task or Load Plan run always has a status. Table 23–2 lists the six possible status values:

Table 23–2 Status Values

Status Name	Status Icon for Sessions	Status Icon for Load Plans	Status Description
Done	✓	✓	The Load Plan, session, step or task was executed successfully.
Done in previous run		✓	The Load Plan step has been executed in a previous Load Plan run. This icon is displayed after a restart.
Error	✗	✗	The Load Plan, session, step or task has terminated due to an error.
Running	⚡	⚡	The Load Plan, session, step or task is being executed.
Waiting	Ⓦ	Ⓦ	The Load Plan, session, step or task is waiting to be executed.
Warning (Sessions and tasks only)	⚠		▪ For Sessions: The session has completed successfully but errors have been detected during the data quality check. ▪ For Tasks: The task has terminated in error, but since errors are allowed on this task, this did not stop the session.
Queued (Sessions only)	●		The session is waiting for an agent to be available for its execution

When finished, a session takes the status of the last executed step (**Done** or **Error**). When finished, the step, takes the status of the last executed task (Except if the task returned a Warning. In this case, the step takes the status **Done**).

A Load Plan is successful (status **Done**) when all its child steps have been executed successfully. It is in **Error** status when at least one of its child steps is in error and has raised its exception to the root step.

Task Types

Tasks are the nodes inside of steps in a session. The types of tasks are shown in Table 23–3.

Table 23–3 Task Types

Task Type	Task Icon for Sessions	Task Description
Normal Task	⋮	This task is executed according to its sequential position in the session. It is marked to DONE status when completed successfully. If it completes in error status, it is failed and marked with the error status.

Table 23–3 (Cont.) Task Types

Task Type	Task Icon for Sessions	Task Description
Serial Task		The children tasks are executed in a sequential order. The serial task is completed and marked to DONE status when all its children tasks have completed successfully. It is considered failed and marked with error status when the first child task completes in error status.
Parallel Task		The children tasks are executed concurrently. The parallel task is considered completed and marked with DONE status when all its children tasks have completed successfully. It is considered failed and marked with ERROR status if any of its child tasks has failed.

Monitoring Executions Results

In Oracle Data Integrator, an execution results in a *session* or in a *Load Plan run* if a Load Plan is executed. A *session* is made up of steps which are made up of tasks. Sessions are viewed and managed in Operator Navigator.

Load Plan runs appear in the Operator Navigator. To review the steps of a Load Plan run, you open the editor for this run. The sessions attached to a Load Plan appear with the rest of the sessions in the Operator Navigator.

Monitoring Sessions

To monitor your sessions:

1. In the Operator Navigator, expand the Session List accordion.
2. Expand the All Executions node and click **Refresh** in the Navigator toolbar.
3. Optionally, activate a Filter to reduce the number of visible sessions. For more information, see "Filtering Sessions" on page 23-9.
4. Review in the list of sessions the status of your session(s).

Monitoring Load Plan Runs

To monitor your Load Plan runs:

1. In the Operator Navigator, expand the Load Plan Executions accordion.
2. Expand the All Executions node and click **Refresh** in the Navigator toolbar.
3. Review in the list the status of your Load Plan run.
4. Double-click this Load Plan run to open the Load Plan Run editor.
5. In the Load Plan Run editor, select the Steps tab.
6. Review the state of the Load Plan steps. On this tab, you can perform the following tasks:
   - Click **Refresh** in the Editor toolbar to update the content of the table.
   - For the Run Scenario steps, you can click in the Session ID column to open the session started by this Load Plan for this step.

Handling Failed Sessions

When your session ends in error or with a warning, you can analyze the error in Operator Navigator.

To analyze an error:

1. In the Operator Navigator, identify the session, the step and the task in error.
2. Double click the task in error. The Task editor opens.
3. On the Definition tab in the Execution Statistics section, the return code and message give the error that stopped the session.
4. On the Code tab, the source and target code for the task is displayed and can be reviewed and edited.

   Optionally, click **Show/Hide Values** to display the code with resolved variable and sequence values. Note that:

   - If the variable values are shown, the code becomes read-only. You are now able to track variable values.
   - Variables used as passwords are never displayed.

   See "Tracking Variables and Sequences" on page 13-21 for more information.
5. On the Connection tab, you can review the source and target connections against which the code is executed.

You can fix the code of the command in the Code tab and apply your changes. Restarting a session (see "Restarting a Session" on page 21-6) is possible after performing this action. The session will restart from the task in error.

Note: Fixing the code in the session's task does not fix the source object that was executed (mapping, procedure, package or scenario). This source object must be fixed in Designer Navigator and the scenario (if any) must be regenerated. Modifying the code within the session is useful for debugging issues.

WARNING: When a session fails, all connections and transactions to the source and target systems are rolled back. As a consequence, uncommitted statements on transactions are not applied.

Reviewing Successful Sessions

When your session ends successfully, you can view the changes performed in Operator Navigator. These changes include record statistics such as the number of inserts, updates, deletes, errors, and the total number of rows as well as execution statistics indicating start and end time of the execution, the duration in seconds, the return code, and the message (if any).

Session level statistics aggregate the statistics of all the steps of this session, and each step's statistics aggregate the statistics of all the tasks within this step.

To review the execution statistics:

1. In the Operator Navigator, identify the session, the step or the task to review.
2. Double click the session, the step or the task. The corresponding editor opens.

3. The record and execution statistics are displayed on the Definition tab. Note that for session steps in which a mapping has been executed or a datastore check has been performed also the target table details are displayed.

Record Statistics

Properties	Description
No. of Inserts	Number of rows inserted during the session/step/task.
No. of Updates	Number of rows updated during the session/step/task.
No. of Deletes	Number of rows deleted during the session/step/task.
No. of Errors	Number of rows in error in the session/step/task.
No. of Rows	Total number of rows handled during this session/step/task.

Execution Statistics

Properties	Description
Start	Start date and time of execution of the session/step/task.
End	End date and time of execution of the session/step/task.
Duration (seconds)	The time taken for execution of the session/step/task.
Return code	Return code for the session/step/task.

Target Table Details

Properties	Description
Table Name	Name of the target datastore.
Model Code	Code of the Model in which the target datastore is stored.
Resource Name	Resource name of the target datastore.
Logical Schema	Logical schema of this datastore.
Forced Context Code	The context of the target datastore.

Handling Failed Load Plans

When a Load Plan ends in error, review the sessions that have failed and caused the Load Plan to fail. Fix the source of the session failure.

You can restart the Load Plan instance. See "Restarting a Load Plan Run" on page 21-13 for more information.

Note that it will restart depending on the Restart Type defined on its steps. See "Handling Load Plan Exceptions and Restartability" on page 15-15 for more information.

You can also change the execution status of a failed Load Plan step from **Error** to **Done** on the Steps tab of the Load Plan run Editor to ignore this particular Load Plan step the next time the Load Pan run is restarted. This might be useful, for example, when the error causing this Load Plan step to fail is not possible to fix at the moment and you want to execute the rest of the Load Plan regardless of this Load Plan step.

Reviewing Successful Load Plans

When your Load Plan ends successfully, you can review the execution statistics from the Load Plan run editor.

You can also review the statistics for each session started for this Load Plan in the session editor.

To review the Load Plan run execution statistics:

1. In the Operator Navigator, identify the Load Plan run to review.
2. Double click the Load Plan run. The corresponding editor opens.
3. The record and execution statistics are displayed on the Steps tab.

Managing your Executions

Managing your development executions takes place in Operator Navigator. You can manage your executions during the execution process itself or once the execution has finished depending on the action that you wish to perform. The actions that you can perform are:

- Managing Sessions
- Managing Load Plan Executions
- Managing the Log
- Managing Scenarios and Load Plans
- Managing Schedules

Managing Sessions

Managing sessions involves the following tasks

- New sessions can be created by executing run-time objects or scenarios. See Chapter 21, "Running Integration Processes," for more information on starting sessions.
- Sessions in progress can be aborted. How to stop sessions is covered in "Stopping a Session" on page 21-9.
- Sessions failed, or stopped by user action can be restarted. Restarting sessions is covered in "Restarting a Session" on page 21-6.

In addition to these tasks, it may be necessary in production to deal with stale sessions.

Cleaning Stale Sessions

Stale sessions are sessions that are incorrectly left in a running state after an agent or repository crash.

The Agent that started a session automatically detects when this session becomes stale and changes it to *Error* status. You can manually request specific Agents to clean stale sessions in Operator Navigator or Topology Navigator.

To clean stale sessions manually:

1. Do one of the following:
   - From the Operator Navigator toolbar menu, select Clean Stale Sessions.

- In Topology Navigator, from the Physical Architecture accordion, select an Agent, right-click and select **Clean Stale Sessions**.

  The Clean Stale Sessions Dialog opens.

2. In the Clean Stale Sessions Dialog specify the criteria for cleaning stale sessions:
   - From the list, select the Agents that will clean their stale sessions.

     Select **Clean all Agents** if you want all Agents to clean their stale sessions.
   - From the list, select the Work Repositories you want to clean.

     Select **Clean all Work Repositories** if you want to clean stale sessions in all Work Repositories.
3. Click **OK** to start the cleaning process. A progress bar indicates the progress of the cleaning process.

Managing Load Plan Executions

Managing Load Plan Executions involves the following tasks:

- New Load Plan Instances and Runs can be created by executing Load Plans. See "Executing a Load Plan" on page 21-10 for more information on starting Load Plans.
- Load Plan Runs in progress can be aborted. How to stop Load Plan runs is covered in "Stopping a Load Plan Run" on page 21-15.
- Load Plan Runs failed, or stopped by user action can be restarted. Restarting Load Plan Runs is covered in "Restarting a Load Plan Run" on page 21-13.

Managing the Log

Oracle Data Integrator provides several solutions for managing your log data:

- Filtering Sessions to display only certain execution sessions in Operator Navigator
- Purging the Log to remove the information of past sessions
- Organizing the Log with Session Folders
- Exporting and Importing Log Data for archiving purposes
- Runtime Logging for ODI components (ODI Studio, ODI Java EE Agent, and ODI Standalone Agent)

Filtering Sessions

Filtering log sessions allows you to display only certain sessions in Operator Navigator, by filtering on parameters such as the user, status or duration of sessions. Sessions that do not meet the current filter are hidden from view, but they are not removed from the log.

To filter out sessions:

1. In the Operator Navigator toolbar menu, click **Filter**. The Define Filter editor opens.
2. In the Define Filter Editor, set the filter criteria according to your needs. Note that the default settings select all sessions.
   - **Session Number**: Use blank to show all sessions.

- **Session Name**: Use `%` as a wildcard. For example `DWH%` matches any session whose name begins with `DWH`.
- Session's execution **Context**
- **Agent** used to execute the session
- **User** who launched the session
- **Status**: Running, Waiting etc.
- **Date** of execution: Specify either a date **From** or a date **To**, or both.
- **Duration greater than** a specified number of seconds

3. Click **Apply** for a preview of the current filter.
4. Click **OK**.

Sessions that do not match these criteria are hidden in the Session List accordion. The Filter button on the toolbar is activated.

To deactivate the filter click **Filter** in the Operator toolbar menu. The current filter is deactivated, and all sessions appear in the list.

Purging the Log

Purging the log allows you to remove past sessions and Load Plan runs from the log. This procedure is used to keeping a reasonable volume of sessions and Load Plans archived in the work repository. It is advised to perform a purge regularly. This purge can be automated using the OdiPurgeLog tool (see: "OdiPurgeLog" on page A-46) in a scenario.

To purge the log:

1. From the Operator Navigator toolbar menu select **Connect Navigator** > **Purge Log...** The Purge Log editor opens.
2. In the Purge Log editor, set the criteria listed in Table 23–4 for the sessions or Load Plan runs you want to delete.

Table 23–4 Purge Log Parameters

Parameter	Description
Purge Type	Select the objects to purge.
From ... To	Sessions and/or Load Plan runs in this time range will be deleted. When you choose to purge session logs only, then the sessions launched as part of the Load Plan runs are not purged even if they match the filter criteria. When you purge Load Plan runs, the Load Plan run which matched the filter criteria and the sessions launched directly as part of the Load Plan run and its child/grand sessions will be deleted.
Context	Sessions and/or Load Plan runs executed in this context will be deleted.
Agent	Sessions and/or Load Plan runs executed by this agent will be deleted.
Status	Session and/or Load Plan runs in this status will be deleted.
User	Sessions and/or Load Plan runs executed by this user will be deleted.

Table 23–4 (Cont.) Purge Log Parameters

Parameter	Description
Name	Sessions and/or Load Plan runs matching this session name will be deleted. Note that you can specify session name masks using % as a wildcard.
Purge scenario reports	If you select **Purge scenario reports**, the scenario reports (appearing under the execution node of each scenario) will also be purged.

Only the sessions and/or Load Plan runs matching the specified filters will be removed:

- When you choose to purge session logs only, then the sessions launched as part of the Load Plan runs are not purged even if they match the filter criteria.
- When you purge Load Plan runs, the Load Plan run which matched the filter criteria and the sessions launched directly as part of Load Plan run and its child/grand sessions will be deleted.
- When a Load Plan run matches the filter, all its attached sessions are also purged irrespective of whether they match the filter criteria or not.

3. Click **OK**.

Oracle Data Integrator removes the sessions and/or Load Plan runs from the log.

Note: It is also possible to delete sessions or Load Plan runs by selecting one or more sessions or Load Plan runs in Operator Navigator and pressing the **Delete** key. Deleting a Load Plan run in this way, deletes the corresponding sessions.

Organizing the Log with Session Folders

You can use **session folders** to organize the log. Session folders automatically group sessions and Load Plan Runs that were launched with certain keywords. Session folders are created under the **Keywords** node on the Session List or Load Plan Executions accordions.

Each session folder has one or more keywords associated with it. Any session launched with all the keywords of a session folder is automatically categorized beneath it.

To create a new session folder:

1. In Operator Navigator, go to the Session List or Load Plan Executions accordion.
2. Right-click the **Keywords** node and select **New Session Folder**.
3. Specify a **Folder Name**.
4. Click **Add** to add a keyword to the list. Repeat this step for every keyword you wish to add.

Note: Only sessions or load plans with all the keywords of a given session folder will be shown below that session folder. Keyword matching is case sensitive.

Table 23–5 lists examples of how session folder keywords are matched.

Table 23–5 Matching of Session Folder Keywords

Session folder keywords	Session keywords	Matches?
DWH, Test, Batch	Batch	No - all keywords must be matched.
Batch	DWH, Batch	Yes - extra keywords on the session are ignored.
DWH, Test	Test, dwh	No - matching is case-sensitive.

To launch a session with keywords, you can for example start a scenario from a command line with the `-KEYWORDS` parameter. Refer to Chapter 21, "Running Integration Processes," for more information.

Note: Session folder keyword matching is dynamic. If the keywords for a session folder are changed or if a new folder is created, existing sessions are immediately re-categorized.

Exporting and Importing Log Data

Export and import log data for archiving purposes.

Exporting Log Data

Exporting log data allows you to export log files for archiving purposes.

To export the log:

1. Select **Export...** from the Designer, Topology, Security or Operator Navigator toolbar menu.
2. In the Export Selection dialog, select **Export the Log**.
3. Click **OK**.
4. In the Export the log dialog, set the log export parameters as described in Table 23–6.

Table 23–6 Log Export Parameters

Properties	Description
Export to directory	Directory in which the export file will be created.
Export to zip file	If this option is selected, a unique compressed file containing all log export files will be created. Otherwise, a set of log export files is created.
Zip File Name	Name given to the compressed export file.
Filters	**This set of options allow to filter the log files to export according to the specified parameters.**
Log Type	From the list, select for which objects you want to retrieve the log. Possible values are: `All|Load Plan runs and attached sessions|Sessions`
From / To	Date of execution: specify either a date From or a date To, or both.
Agent	Agent used to execute the session. Leave the default **All Agents** value, if you do not want to filter based on a given agent.
Context	Session's execution Context. Leave the default **All Contexts** value, if you do not want to filter based on a context.

Table 23–6 (Cont.) Log Export Parameters

Properties	Description
Status	The possible states are `Done`, `Error`, `Queued`, `Running`, `Waiting`, `Warning` and `All States`. Leave the default **All States** value, if you do not want to filter based on a given session state.
User	User who launched the session. Leave the default **All Users** value, if you do not want to filter based on a given user.
Session Name	Use `%` as a wildcard. For example `DWH%` matches any session whose name begins with `DWH`.
Advanced options	**This set of options allow to parameterize the output file format.**
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.

5. Click **OK**.

The log data is exported into the specified location.

Note that you can also automate the log data export using the OdiExportLog tool (see "OdiExportLog" on page A-17).

Importing Log Data

Importing log data allows you to import into your work repository log files that have been exported for archiving purposes.

To import the log:

1. Select **Import...** from the Designer, Topology, Security or Operator Navigator toolbar menu.
2. In the Import Selection dialog, select **Import the Log**.
3. Click **OK**.
4. In the Import of the log dialog:
    1. Select the **Import Mode**. Note that sessions can only be imported in Synonym Mode INSERT mode. Refer to "Import Types" on page 20-3 for more information.
    2. Select whether you want to import the files **From a Folder** or **From a ZIP file.**
    3. Enter the file import folder or zip file.
    4. Click **OK**.

The specified folder or ZIP file is imported into the work repository.

Runtime Logging for ODI components

You can set up runtime logging to trace ODI components or set a verbose level to investigate different issues or to monitor the system. The following ODI components can be traced: ODI Studio, ODI Java EE Agents, and ODI Standalone Agents.

Note: A verbose logging will slow down the ODI performance.

Log Level

The log level can be set against a log_handler and/or logger elements. Note the following when setting the log level:

- If it is set against a log_handler, then it applies to all usages of the log_handler.
- If it is set against a logger, then it applies to all of its handlers and any descendent loggers that do not have an explicit level setting.
- A message is logged if its log-level is:
  - Greater or equal than (>=) the level of its logger AND
  - Greater or equal than (>=) the level of its log_handler

Table 23–7 shows the mapping between the Java log levels and the Oracle Java Debugging level (OJDL).

Table 23–7 Mapping between Java log levels and OJDL log levels

Java log levels	OJDL log levels
SEVERE intValue()+100	INCIDENT_ERROR:1
SEVERE	ERROR:1
WARNING	WARNING:1
INFO	NOTIFICATION:1
CONFIG	NOTIFICATION:16
FINE	TRACE:1
FINER	TRACE:16
FINEST	TRACE:32

Setting Up Runtime Logging

To set up runtime logging you have to enable different ODI loggers and log handlers and set the related log level in the ODI logging system configuration file.

1. Open the ODI logging system configuration file of the ODI component.

   Each component has its own configuration file:

   - ODI Studio:

     ```
     $ODI_HOME/odi/studio/bin/ODI-logging-config.xml
     ```

   - ODI Java EE Agent:

     ```
     <DOMAIN_HOME>/config/fmwconfig/logging/oraclediagent-logging.xml
     ```

   - ODI Standalone Agent:

     ```
     <DOMAIN_HOME>/config/fmwconfig/components/ODI/<INSTANCE_
     NAME>/ODI-logging-config.xml
     ```

2. Make sure that the path to your log files is a valid and existing path. For example:

   ```
   <log_handler name="ODI-file-handler"
   class="oracle.core.ojdl.logging.ODLHandlerFactory"
   level="ALL">
     <property name="format" value="ODL-Text"/>
     <property name="path" value="/u01/oracle/odi11g/oracledi/agent/log/${LOG_
   FILE}"/>
   ```

```
  <property name="maxFileSize" value="1000000"/> <!-- in bytes -->
  <property name="maxLogSize" value="50000000"/> <!-- in bytes -->
  <property name="encoding" value="UTF-8"/>
</log_handler>
```

Note the following concerning the log files path:

- For ODI Standalone Agents the default value points to a non-existing folder. Create a log folder or set another path.
- If you are on Windows, the path could be for example:

  `%ODI_HOME%\oracledi\agent\log\${LOG_FILE}`
- You can use a relative path on Windows and Unix.

3. Enable the logger and set the log level. For example, for logger *oracle.odi.agent* you can enable the most verbose logging setting:

```
<logger name="oracle.odi.agent" level="TRACE:32" useParentHandlers="false">
  <handler name="ODI-file-handler"/>
  <handler name="ODI-console-handler"/>
</logger>
```

4. Save the configuration file to take the changes into account.

Example INFO (NOTIFICATION:1) Message

The INFO `(NORTIFICATION:1)` Message is logged if:

- The logger level (possibly inherited) is `<= NOTIFICATION:1`
- The log_handler level is `<= NOTIFICATION:1` (for example: `TRACE:1`)

The INFO `(NORTIFICATION:1)` Message is *not* logged if:

- The logger level (possibly inherited) is `> NOTIFICATION:1`
- The log_handler level is `> NOTIFICATION:1` (for example: `WARNING:1`)

The Runtime logger is called *oracle.odi.agent*. Its message levels cover the following content:

```
NOTIFICATION:1 (or INFO) Agent Level
NOTIFICATION:16 (or CONFIG) the above + Session Level
TRACE:1 (or FINE) the above + Step Level
TRACE:16 (or FINER) the above + Task + SQL
TRACE:32 (or FINEST) the above + more detail
```

It is recommended that the console level for *oracle.odi.agent* is set so that Agent startup messages are displayed (`NOTIFICATION:1`).

Managing Scenarios and Load Plans

You can also manage your executions in Operator Navigator by using scenarios or Load Plans.

Before running a scenario, you need to generate it in Designer Navigator or import from a file. See Chapter 14, "Using Scenarios." Load Plans are also created using Designer Navigator, but can also be modified using Operator Navigator. See Chapter 15, "Using Load Plans," for more information.

Launching a scenario from Operator Navigator is covered in "Executing a Scenario from ODI Studio" on page 21-3, and how to run a Load Plan is described in "Executing a Load Plan" on page 21-10.

Load Plan and Scenario Folders

In Operator Navigator, scenarios and Load Plans can be grouped into Load Plan and Scenario folders to facilitate organization. Load Plan and Scenario folders can contain other Load Plan and Scenario folders.

To create a Load Plan and Scenario folder:

1. In Operator Navigator go to the Load Plans and Scenarios accordion.
2. From the Load Plans and Scenarios toolbar menu, select **New Load Plan and Scenario Folder**.
3. On the Definition tab of the Load Plan and Scenario Folder editor enter a name for your folder.
4. From the File menu, select **Save**.

You can now reorganize your scenarios and Load Plans. Drag and drop them into the Load Plan and Scenario folder.

Importing Load Plans, Scenarios, and Solutions in Production

A Load Plan or a scenario generated from Designer can be exported and then imported into a development or execution repository. This operation is used to deploy Load Plans and scenarios in a different repository, possibly in a different environment or site.

Importing a Load Plan or scenario in a development repository is performed via Designer or Operator Navigator. With a execution repository, only Operator Navigator is available for this purpose.

See "Importing Scenarios in Production" on page 14-5 for more information on how to import a scenario in production, and "Importing Load Plans" on page 15-19 for more information on the Load Plan import.

Similarly, a solution containing several scenarios can be imported to easily transfer and restore a group of scenarios at once. See Chapter 19, "Using Version Control," for more information. Note that when connected to an execution repository, only scenarios may be restored from solutions.

Managing Schedules

A schedule is always attached to one scenario or one Load Plan. Schedules can be created in Operator Navigator. See "Scheduling Scenarios and Load Plans" on page 21-16 for more information.

You can also import an already existing schedule along with a scenario or Load Plan import. See "Importing Scenarios in Production" on page 14-5 and "Exporting, Importing and Versioning Load Plans" on page 15-18for more information.

You can view the scheduled tasks of all your agents or you can view the scheduled tasks of one particular agent. See "Displaying the Schedule" on page 21-18 for more information.

24 Using Oracle Data Integrator Console

This chapter describes how to work with Oracle Data Integrator Console. An overview of the Console user interface is provided.

This chapter includes the following sections:

Introduction to Oracle Data Integrator Console

Oracle Data Integrator Console is a web-based console for managing and monitoring an Oracle Data Integrator run-time architecture and for browsing design-time objects.

This section contains the following topics:

Oracle Data Integrator Console Concepts

Oracle Data Integrator Console is a web-based console available for different types of users:

- Administrators use Oracle Data Integrator Console to create and import repositories and to configure the Topology (data servers, schemas, and so forth).
- Production operators use Oracle Data Integrator Console to manage scenarios and Load Plans, monitor sessions and Load Plan runs, and manage the content of the error tables generated by Oracle Data Integrator.
- Business users and developers browse development artifacts in this interface, using, for example, the Data Lineage and Flow Map features.

This web interface integrates seamlessly with Oracle Fusion Middleware Control Console and allows Fusion Middleware administrators to drill down into the details of Oracle Data Integrator components and sessions.

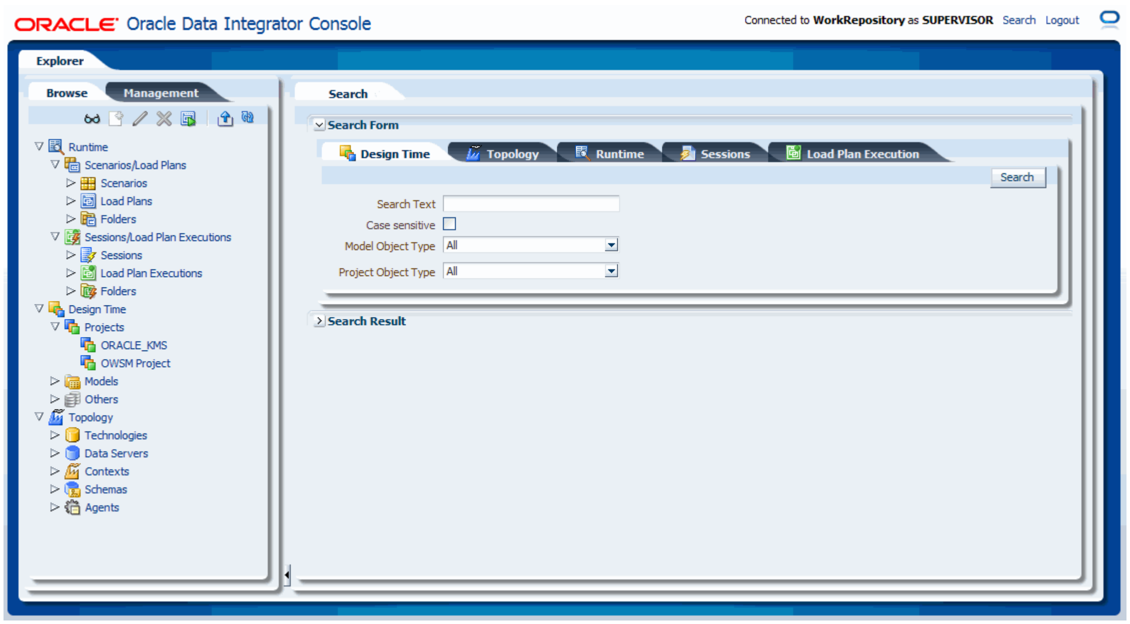
Note: Oracle Data Integrator Console is required for the Fusion Middleware Control Extension for Oracle Data Integrator. It must be installed and configured for this extension to discover and display the Oracle Data Integrator components in a domain.

Oracle Data Integrator Console Interface

Oracle Data Integrator Console is a web interface using the ADF-Faces framework.

Figure 24–1 shows the layout of Oracle Data Integrator Console.

Figure 24–1 Oracle Data Integrator Console



Search Text

Case sensitive ☐

Model Object Type: All

Project Object Type: All

Oracle Data Integrator Console displays the objects available to the current user in two Navigation tabs in the left panel:

- Browse tab displays the repository objects that can be browsed and edited. In this tab you can also manage sessions and error tables.
- Management tab is used to manage the repositories and the repository connections. This tab is available to connection users having Supervisor privileges, or to any user to set up the first repository connections.

The right panel displays the following tabs:

- Search tab is always visible and allows you to search for objects in the connected repository.
- One Master/Details tab is displayed for each object that is being browsed or edited. Note that it is possible to browse or edit several objects at the same time.

The search field above the Navigation tabs allows you to open the search tab when it is closed.

Working with the Navigation Tabs

In the Navigation tabs, you can browse for objects contained in the repository. When an object or node is selected, the Navigation Tab toolbar displays icons for the actions available for this object or node. If an action is not available for this object, the icon is grayed out. For example, you can edit and add data server objects under the Topology node in the Browse Tab, but you cannot edit Projects under the Designer node. Note that the number of tabs that you can open at the same time is limited to ten.

Using Oracle Data Integrator Console

This section explains the different types of operations available in Oracle Data Integrator console. It does not focus on each type of object that can be managed with the console, but gives keys to manage objects with the console.

This section includes the following topics:

- Connecting to Oracle Data Integrator Console
- Generic User Operations
- Managing Scenarios and Sessions
- Managing Load Plans
- Purging the Log
- Using Data Lineage and Flow Map
- Performing Administrative Operations

Note: Oracle Data Integrator Console uses the security defined in the master repository. Operations that are not allowed for a user will appear grayed out for this user.

In addition, the **Management** tab is available only for users with Supervisor privileges.

Connecting to Oracle Data Integrator Console

Oracle Data Integrator console connects to a repository via a Repository Connection, defined by an administrator.

Note that you can only connect to Oracle Data Integrator Console if it has been previously installed. See *Installing and Configuring Oracle Data Integrator* for more information about installing Oracle Data Integrator Console.

Note: The first time you connect to Oracle Data Integrator Console, if no repository connection is configured, you will have access to the **Management** tab to create a first repository connection. See "Creating a Repository Connection" on page 24-12 for more information. After your first repository connection is created, the Management tab is no longer available from the Login page, and is available only for users with Supervisor privileges.

Connecting to Oracle Data Integrator Console

To connect to Oracle Data Integrator Console:

1. Open a web browser, and connect to the URL where Oracle Data Integrator Console is installed. For example: `http://odi_host:8001/odiconsole/`.
2. From the Repository list, select the Repository connection corresponding to the master or work repository you want to connect.
3. Provide a **User ID** and a **Password**.
4. Click **Sign In**.

Generic User Operations

This section describes the generic operations available in Oracle Data Integrator Console for a typical user.

This section includes the following operations:

Note: Creating, editing, and deleting operations are not allowed for Scenarios and Load Plans. For more information on the possible actions that can be performed with these objects in ODI Console, see "Managing Scenarios and Sessions" on page 24-5 and "Managing Load Plans" on page 24-8.

- Viewing an Object
- Editing an Object
- Creating an Object
- Deleting an Object
- Searching for an Object

Viewing an Object

To view an object:

1. Select the object in the **Browse** or **Management** Navigation tab.
2. Click **View** in the Navigation tab toolbar. The simple page or the Master/Detail page for the object opens.

Editing an Object

To edit an object:

1. Select the object in the **Browse** or **Management** Navigation tab.
2. Click **Update** in the Navigation tab toolbar. The edition page for the object opens.
3. Change the value for the object fields.
4. Click **Save** in the edition page for this object.

Creating an Object

To create an object:

1. Navigate to the parent node of the object you want to create in the **Browse** or **Management** Navigation tab. For example, to create a Context, navigate to the **Topology** > **Contexts** node in the **Browse** tab.
2. Click **Create** in the Navigation tab toolbar. An **Add** dialog for this object appears.
3. Provide the values for the object fields.
4. Click **Save** in the Add dialog of this object. The new object appears in the Navigation tab.

Deleting an Object

To delete an object:

1. Select the object in the **Browse** or **Management** Navigation tab.
2. Click **Delete** in the Navigation tab toolbar.

3. Click OK in the confirmation window.

Searching for an Object

To search for an object:

1. In the **Search** tab, select the tab corresponding to the object you want to search:
   - **Design Time** tab allows you to search for design-time objects
   - **Topology** tab allows you to search for topology objects
   - **Runtime** tab allows you to search for run-time objects such as Load Plans, Scenarios, Scenario Folders, or Session Folders
   - **Sessions** tab allows you to search for sessions
   - **Load Plan Execution** tab allows you to search for Load Plan runs
2. Set the search parameters to narrow your search.

   For example when searching design-time or topology objects:

   1. In the **Search Text** field, enter a part of the name of the object that you want to search.
   2. Select **Case sensitive** if you want the search to be case sensitive (this feature is not provided for the sessions or Load Plan execution search.
   3. Select in **Models/Project** (Designer tab) or **Topology** (Topology tab) the type of object you want to search for. Select **All** to search for all objects.
3. Click **Search**.
4. The **Search Result**s appear, grouped by object type. You can click an object in the search result to open its master/details page.

Managing Scenarios and Sessions

This section describes the operations related to scenarios and sessions available in Oracle Data Integrator Console.

This section includes the following operations:

- Importing a Scenario
- Exporting a Scenario
- Running a Scenario
- Stopping a Session
- Restarting a Session
- Cleaning Stale Sessions
- Managing Data Statistics and Erroneous Records

Importing a Scenario

To import a scenario:

1. Select the **Browse** Navigation tab.
2. Navigate to **Runtime > Scenarios/Load Plans > Scenarios**.
3. Click **Import** in the Navigation tab toolbar.
4. Select an **Import Mode** and select an export file in **Scenario XML File**.

5. Click **Import Scenario**.

Exporting a Scenario

To export a scenario:

1. Select the **Browse** Navigation tab.
2. Navigate to **Runtime > Scenarios/Load Plans > Scenarios**.
3. Click **Export** in the Navigation tab toolbar.
4. In the Export Scenario dialog, set the parameters as follows:
   - From the **Scenario Name** list, select the scenario to export.
   - In the **Encoding Java Charset** field, enter the Java character set for the export file.
   - In the **Encoding XML Charset** field, enter the encoding to specify in the export file.
   - In the **XML Version** field, enter the XML Version to specify in the export file.
   - Optionally, select **Include Dependant objects** to export linked child objects.
5. Click **Export Scenario**.

Running a Scenario

To execute a scenario:

1. Select the **Browse** Navigation tab.
2. Navigate to **Runtime > Scenarios/Load Plans > Scenarios**.
3. Select the scenario you want to execute.
4. Click **Execute** in the Navigation tab toolbar.
5. Select an **Agent**, a **Context**, and a **Log Level** for this execution.
6. Click **Execute Scenario**.

Stopping a Session

Note that you can perform a normal or an immediate kill of a running session. Sessions with the status *Done*, *Warning*, or *Error* cannot be killed.

To kill a session:

1. Select the **Browse** Navigation tab.
2. Navigate to **Runtime > Sessions/Load Plan Executions > Sessions**.
3. Select the session you want to stop.
4. Click **Kill** in the Navigation tab toolbar.

Restarting a Session

To restart a session:

1. Select the **Browse** Navigation tab.
2. Navigate to **Runtime > Sessions/Load Plan Executions > Sessions**.
3. Select the session you want to restart.
4. Click **Restart** in the Navigation tab toolbar.

5. In the Restart Session dialog, set the parameters as follows:
   - **Agent**: From the list, select the agent you want to use for running the new session.
   - **Log Level**: From the list, select the log level. 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.
6. Click **Restart Session**.

Cleaning Stale Sessions

To clean stale sessions:

1. Select the **Browse** Navigation tab.
2. Navigate to **Runtime > Sessions/Load Plan Executions > Sessions**.
3. Click **Clean** in the Navigation tab toolbar.
4. In the **Clean Stale Sessions** dialog, select the **Agent** for which you want to clean stale sessions.
5. Click **OK**.

Managing Data Statistics and Erroneous Records

Oracle Data Integrator Console allows you to browse the details of a session, including the record statistics. When a session detects erroneous data during a flow or static check, these errors are isolated into error tables. You can also browse and manage the erroneous rows using Oracle Data Integrator Console.

> **Note:** Sessions with erroneous data detected finish in **Warning** status.

To view the erroneous data:

1. Select the **Browse** Navigation tab.
2. Navigate to a given session using **Runtime > Sessions/Load Plan Executions > Sessions**. Select the session and click **View** in the Navigation tab toolbar.

   The Session page is displayed.
3. In the Session page, go to the **Relationships** section and select the **Record Statistics** tab.

   This tab shows each physical table targeting in this session, as well as the record statistics.
4. Click the number shown in the **Errors** column. The content of the error table appears.
   - You can filter the errors by Constraint Type, Name, Message Content, Detection date, and so forth. Click **Filter Result** to apply a filter.
   - Select a number of errors in the **Query Results** table and click **Delete** to delete these records.
   - Click **Delete All** to delete all the errors.

> **Note:** Delete operations cannot be undone.

Managing Load Plans

This section describes the operations related to Load Plans available in Oracle Data Integrator Console.

This section includes the following operations:

- Importing a Load Plan
- Exporting a Load Plan
- Running a Load Plan
- Stopping a Load Plan Run
- Restarting a Load Plan Run

Importing a Load Plan

To import a Load Plan:

1. Select the **Browse** Navigation tab.
2. Navigate to **Runtime** > **Scenarios/Load Plans** > **Load Plans**.
3. Click **Import** in the Navigation tab toolbar.
4. In the Import Load Plan dialog, select an **Import Mode** and select an export file in the **Select Load Plan XML File** field.
5. Click **Import**.

Note: When you import a Load Plan that has been previously exported, the imported Load Plan does not include the scenarios referenced by the Load Plan. Scenarios used in a Load Plan need to be imported separately. See "Importing a Scenario" on page 24-5 for more information.

Exporting a Load Plan

To export a Load Plan:

1. Select the **Browse** Navigation tab.
2. Navigate to **Runtime** > **Scenarios/Load Plans** > **Load Plans**.
3. Select the Load Plan to export.
4. Click **Export** in the Navigation tab toolbar.
5. In the Export dialog, set the parameters as follows:
   - From the **Load Plan Name** list, select the Load Plan to export.
   - In the **Encoding Java Charset** field, enter the Java character set for the export file.
   - In the **Encoding XML Charset** field, enter the encoding to specify in the export file.
   - In the **XML Version** field, enter the XML Version to specify in the export file.
   - Optionally, select **Include Dependant objects** to export linked child objects.
6. Click **Export**.

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. See "Exporting a Scenario" on page 24-6 for more information.

Running a Load Plan

To run a Load Plan:

1. Select the **Browse** Navigation tab.
2. Navigate to **Runtime** > **Scenarios/Load Plans** > **Load Plans**.
3. Select the Load Plan you want to execute.
4. Click **Execute** in the Navigation tab toolbar.
5. Select a **Logical Agent**, a **Context**, a **Log Level**, and if your Load Plan uses variables, specify the **Startup values** for the Load Plan variables.
6. Click **Execute**.

Stopping a Load Plan Run

Note that you can perform a normal or an immediate kill of a Load Plan run. Any running or waiting Load Plan Run can be stopped.

To stop a Load Plan Run:

1. Select the **Browse** Navigation tab.
2. Navigate to **Runtime > Sessions/Load Plan Executions > Load Plan Executions**.
3. Select the Load Plan run you want to stop.
4. Click **Kill** in the Navigation tab toolbar.

Restarting a Load Plan Run

A Load Plan can only be restarted if the selected run of the current Load Plan instance is in Error status and if there is no other instance of the same Load Plan currently running.

To restart a Load Plan Run:

1. Select the **Browse** Navigation tab.
2. Navigate to **Runtime > Sessions/Load Plan Executions > Load Plan Executions**.
3. Select the Load Plan run you want to restart.
4. In the Restart Load Plan Dialog, select the **Physical Agent** that restarts the Load Plan. Optionally, select a different log level.
5. Click **Restart** in the Navigation tab toolbar.

Purging the Log

This section describes how to purge the log in Oracle Data Integrator Console by removing past sessions and/or Load Plan runs from the log.

To purge the log:

1. Select the **Browse** Navigation tab.
2. Navigate to **Runtime > Sessions/Load Plan Executions**.

3. Click **Purge** in the Navigation tab toolbar.
4. In the Purge Sessions/Load Plan Executions dialog, set the purge parameters listed in Table 24–1.

Table 24–1 Purge Log Parameters

Parameter	Description
Purge Type	Select the objects to purge.
From ... To	Sessions and/or Load Plan runs in this time range will be deleted. When you choose to purge session logs only, then the sessions launched as part of the Load Plan runs are not purged even if they match the filter criteria. When you purge Load Plan runs, the Load Plan run which matched the filter criteria and the sessions launched directly as part of the Load Plan run and its child/grand sessions will be deleted.
Context	Sessions and/or Load Plan runs executed in this context will be deleted.
Agent	Sessions and/or Load Plan runs executed by this agent will be deleted.
Status	Session and/or Load Plan runs in this status will be deleted.
User	Sessions and/or Load Plan runs executed by this user will be deleted.
Name	Sessions and/or Load Plan runs matching this session name will be deleted. Note that you can specify session name masks using % as a wildcard.
Purge scenario reports	If you select **Purge scenario reports**, the scenario reports (appearing under the execution node of each scenario) will also be purged.

Only the sessions and/or Load Plan runs matching the specified filters will be removed:

- When you choose to purge session logs only, then the sessions launched as part of the Load Plan runs are not purged even if they match the filter criteria.
- When you purge Load Plan runs, the Load Plan run which matched the filter criteria and the sessions launched directly as part of Load Plan run and its child/grand sessions will be deleted.
- When a Load Plan run matches the filter, all its attached sessions are also purged irrespective of whether they match the filter criteria or not.

5. Click **OK**.

Oracle Data Integrator Console removes the sessions and/or Load Plan runs from the log.

Using Data Lineage and Flow Map

This section describes how to use the Data Lineage and Flow Map features available in Oracle Data Integrator Console.

- **Data Lineage** provides graph displaying the flows of data from the point of view of a given datastore. In this graph, you can navigate back and forth and follow this data flow.
- **Flow Map** provides a map of the relations that exist between the data structures (models, sub-models and datastores) and design-time objects (projects, folders, packages, mappings). This graph allows you to draw a map made of several data structures and their data flows.

This section includes the following operations:

- Working with the Data Lineage
- Working with the Flow Map

Working with the Data Lineage

To view the Data Lineage:

1. Select the **Browse** Navigation tab.
2. Navigate to **Design Time > Models > Data Lineage**.
3. Click **View** in the Navigation tab toolbar.
4. In the Data Lineage page, select a Model, then a Sub-Model and a datastore in this model.
5. Select **Show Mappings** if you want that mappings are displayed between the datastores nodes.
6. Select the prefix to add in your datastores and mapping names in the **Naming Options** section.
7. Click **View** to draw the Data Lineage graph. This graph is centered on the datastore selected in step 4.

   In this graph, you can use the following actions:

   - Click **Go Back** to return to the Data Lineage options and redraw the graph.
   - Use the **Hand** tool and then click a datastore to redraw the lineage centered on this datastore.
   - Use the **Hand** tool and then click a mapping to view this mapping's page.
   - Use the **Arrow** tool to expand/collapse groups.
   - Use the **Move** tool to move the graph.
   - Use the **Zoom In/Zoom Out** tools to resize the graph.
   - Select **View Options** to change the display options have the graph refreshed with this new option.

Working with the Flow Map

To view the Flow Map:

1. Select the **Browse** Navigation tab.
2. Navigate to **Design Time > Models > Flow Map**.
3. Click **View** in the Navigation tab toolbar.
4. In the Data Lineage page, select one or more **Model**. Select **All** to select all models.
5. Select one of more **Projects**. Select **All** to select all projects.

6. In the **Select the level of details of the map** section, select the granularity of the map. The object that you select here will be the nodes of your graph.

   Check **Do not show Projects, Folders...** if you want the map to show only data structure.

7. Optionally, indicate the grouping for the data structures and design-time objects in the map, using the options in the **Indicate how to group Objects in the Map** section.

8. Click **View** to draw the **Flow Map** graph.

   In this graph, you can use the following actions:

   - Click **Go Back** to return to the Flow Map options and redraw the graph.
   - Use the **Hand** tool and then click a node (representing a datastore, an mapping, and so forth) in the map to open this object's page.
   - Use the **Arrow** tool to expand/collapse groups.
   - Use the **Move** tool to move the graph.
   - Use the **Zoom In/Zoom Out** tools to resize the graph.

Performing Administrative Operations

This section describes the different administrative operations available in Oracle Data Integrator Console. These operations are available for a user with Supervisor privileges.

This section includes the following operations:

- Creating a Repository Connection
- Testing a Data Server or a Physical Agent Connection
- Administering Repositories
- Administering Java EE Agents

Creating a Repository Connection

A *repository connection* is a connection definition for Oracle Data Integrator Console. A connection does not include Oracle Data Integrator user and password information.

To create a repository connection:

1. Navigate to the **Repository Connections** node in the **Management** Navigation tab.

2. Click **Create** in the Navigation tab toolbar. A **Create Repository Connection** dialog for this object appears.

3. Provide the values for the repository connection:

   - **Connection Alias**: Name of the connection that will appear on the Login page.
   - **Master JNDI URL**: JNDI URL of the datasource to connect the master repository database.
   - **Supervisor User Name**: Name of the Oracle Data Integrator user with Supervisor privileges that Oracle Data Integrator Console will use to connect to the repository. This user's password must be declared in the WLS or WAS Credential Store.

- **Work JNDI URL**: JNDI URL of the datasource to connect the work repository database. If no value is given in this field. The repository connection will allow connection to the master only, and the Navigation will be limited to Topology information.
- **JNDI URL**: Check this option if you want to use the environment naming context (ENC). When this option is checked, Oracle Data Integrator Console automatically prefixes the data source name with the string `java:comp/env/` to identify it in the application server's JNDI directory. Note that the JNDI Standard is not supported by Oracle WebLogic Server and for global data sources.
- **Default**: Check this option if you want this Repository Connection to be selected by default on the login page.

4. Click **Save**. The new Repository Connection appears in the **Management** Navigation tab.

Testing a Data Server or a Physical Agent Connection

This sections describes how to test the data server connection or the connection of a physical agent in Oracle Data Integrator Console.

To test the data server connection:

1. Select the **Browse** Navigation tab.
2. Navigate to **Topology > Data Servers**.
3. Select the data server whose connection you want to test.
4. Click **Test Connection** in the Navigation tab toolbar.
5. In the Test Connection dialog, select the:
   - **Physical Agent** that will carry out the test
   - **Transaction** on which you want to execute the command. This parameter is only displayed if there is any On Connect/Disconnect command defined for this data server. The transactions from 0 to 9 and the **Autocommit** transaction correspond to connection created by sessions (by procedures or knowledge modules). The **Client Transaction** corresponds to the client components (ODI Console and Studio).
6. Click **Test**.

A dialog showing "Connection successful!" is displayed if the test has worked. If not, an error message is displayed.

To test the physical agent connection:

1. Select the **Browse** Navigation tab.
2. Navigate to **Topology > Agents > Physical Agents**.
3. Select the physical agent whose connection you want to test.
4. Click **Test Connection** in the Navigation tab toolbar.

A dialog showing "Connection successful!" is displayed if the test has worked. If not, an error message is displayed.

Administering Repositories

Oracle Data Integrator Console provides you with features to perform management operations (create, import, export) on repositories. These operations are available from the **Management** Navigation tab, under the **Repositories** node. These management operations reproduce in a web interface the administrative operations available via the Oracle Data Integrator Studio and allow setting up and maintaining your environment from the ODI Console.

See Chapter 3, "Administering Repositories," and Chapter 20, "Exporting and Importing," for more information on these operations.

Administering Java EE Agents

Oracle Data Integrator Console allows you to add JDBC datasources and create templates to deploy physical agents into WebLogic Server.

See Chapter 4, "Setting Up a Topology," for more information on Java EE Agents, datasources and templates.

To add a datasource to a physical agent:

1. Select the **Browse** Navigation tab.
2. Navigate to **Topology > Agents > Physical Agents**.
3. Select the agent you want to manage.
4. Click **Edit** in the Navigation tab toolbar.
5. Click **Add Datasource**
6. Provide a **JNDI Name** for this datasource and select the **Data Server Name**. This datasource will be used to connect to this data server from the machine into which the Java EE Agent will be deployed.
7. Click **OK**.
8. Click **Save** to save the changes to the physical agent.

To create a template for a physical agent:

1. Select the **Browse** Navigation tab.
2. Navigate to **Topology > Agents > Physical Agents**.
3. Select the agent you want to manage.
4. Click **Edit** in the Navigation tab toolbar.
5. Click **Agent Deployment**.
6. Follow the steps of the **Agent Deployment** wizard. This wizard reproduces in a web interface the Server Template Generation wizard. See "Deploying an Agent in a Java EE Application Server" on page 4-15 for more details.

Part VII

Managing Security Settings

This part describes how to manage the security settings in Oracle Data Integrator.

This part contains the following chapters:

25 Managing Security in Oracle Data Integrator

This chapter describes how to set up security in Oracle Data Integrator. It includes an overview of Oracle Data Integrator security concepts and components.

This chapter includes the following sections:

Introduction to Oracle Data Integrator Security

Oracle Data Integrator security is used to secure any action performed by authenticated users against the design-time and run-time artifacts and components of Oracle Data Integrator.

The Oracle Data Integrator security model is based on granting privileges on methods, objects types, or specific object instances to users.

All the security information for Oracle Data Integrator is stored in the master repository.

This section contains the following topics:

Objects, Instances and Methods

An **object** is a representation of a design-time or run-time artifact handled through Oracle Data Integrator. Examples of objects include agents, projects, models, data stores, scenarios, mappings, and even repositories. Specific objects have a double name, such as Agent/Context and Profile/Method. These objects represent links between objects. These links are also objects. For instance, Agent/Context corresponds to a physical/logical agent association made through the contexts. Privileges on this object enable the ability to change this association in the topology.

An **instance** is a particular occurrence of an object. For example, the `Datawarehouse` project is an instance of the `Project` object.

A **method** is an action that can be performed on an object, such as edit, delete, and so on. Each object has a predefined set of methods.

Note: The notions of object instance and method in Oracle Data Integrator are similar to the same concepts used in object-oriented programming.

CAUTION: Although they appear in the Security Navigator, objects and methods are predefined in Oracle Data Integrator and should not be altered.

Profiles

A **profile** contains a set of privileges for working with Oracle Data Integrator. You can assign one or more profiles to a user, or to a role, to grant the sum of the privileges in those profiles to that user or role.

A **profile method** is an authorization granted to a profile on a method of an object type. Each granted method allows a user with this profile to perform an action on an instance of an object type.

In Oracle Data Integrator Studio, methods granted to a profile appear under this profile in the **Profiles** accordion of the Security Navigator. When a method does not appear for a given profile, this profile does not have access to this method.

A method can be granted as a generic or nongeneric privilege:

- A method granted as a generic privilege is granted by default on all the instances of this object.
- A method granted as a nongeneric privilege is not granted by default on all object instances, but may be granted on a per instance basis.

Generic vs. Nongeneric Profiles

Generic profiles have the generic privilege option selected for all object methods. This implies that a user, or role, with such a profile is by default authorized for all methods of all instances of an object to which the profile is authorized.

Nongeneric profiles are not by default authorized for all methods on the instances because the generic privilege option is not selected for all object methods. You must grant the user, or role, the rights on the methods for each instance.

If you want a user, or role, to have the rights on no instance by default, but want to grant the rights on a per instance basis, the user or role must be given a nongeneric profile.

If you want a user or role to have the rights on all instances of an object type by default, you must give the user or role a generic profile.

Built-In Profiles

Oracle Data Integrator contains built-in profiles that you can assign to the users or roles you create.

Table 25–1 shows the built-in profiles provided by Oracle Data Integrator.

Table 25–1 Built-In Profiles

Profile Name	Description
CONNECT	Profile granted with the basic privileges to connect to Oracle Data Integrator. It should be granted with another profile.
CONSOLE	Profile granted with privileges to use the Oracle Data Integrator Console.
DESIGNER	Profile granted with privileges to perform development operations. Use this profile for users who will work mainly on projects.
METADATA_ADMIN	Profile granted with privileges to manage metadata. Use this profile for users that will work mainly on models.
NG_CONSOLE	Nongeneric version of the CONSOLE profile.
NG_DESIGNER	Nongeneric version of the DESIGNER profile.
NG_METADATA_ADMIN	Nongeneric version of the METATADA_ADMIN profile.
NG_VERSION_ADMIN	Nongeneric version of the VERSION_ADMIN profile.
OPERATOR	Profile granted with privileges to manage run-time objects. Use this profile for production users.
SECURITY_ADMIN	Profile granted with privileges to edit security. Use this profile for security administrators.
TOPOLOGY_ADMIN	Profile granted with privileges to edit the Topology. Use this profile for system or Oracle Data Integrator administrators.
VERSION_ADMIN	Profile granted with privileges to create, restore and edit versions and solutions. Use this profile for project managers, or developers who are entitled to perform version management operations.

Note: Oracle recommends not modifying built-in profiles because they evolve to secure new features of Oracle Data Integrator. If you want to customize your own profiles or existing profiles, Oracle recommends that you create copies of the built-in profiles and then customize those copies.

Users

A **user** is an Oracle Data Integrator user. A user inherits the following privileges:

- All the privileges granted to its various profiles
- Privileges on objects or instances, or both, given to this user

A **user method** is a privilege granted to a user on a method of an object type. Each granted method allows the user to perform an action (edit, delete, and so forth) on instances of an object type (project, model, data store, and so forth). These methods are similar to profiles methods, but applied to users.

You can grant users with privileges on instances on specific work repositories where these instances exist. For example, you may grant a developer user with the edit privilege on the LOAD_DATAWAREHOUSE scenario on the a DEVELOPMENT repository and not on a PRODUCTION repository.

You grant an authorization by object instance to a user on an object instance. It allows you to grant to this user certain methods of this object instance.

An authorization by object instance for a given instance that appears in a user's tree indicates that the user is granted specific privileges on the object methods for the given instance. You specify these privileges in the object instance editor. If an instance is not visible in the tree of the user instances, then the User Method or Profile Method privileges for the object type apply.

Because an instance may be replicated over the different work repositories that are attached to the master repository, you can define authorizations by object instances for one, several, or all your work repositories that are attached to this master repository. For example, you can replicate a LOAD_DATAWAREHOUSE scenario instance (using, for example, versioning) in the DEVELOPMENT, TEST, and PRODUCTION repositories. Privileges on the instance can change depending on the repository. For example, it is common to replicate projects from a development repository to a test repository. You can grant edit privileges to a developer for that developer's project in the development repository, but not in the test repository. In the test repository, the developer is granted with only view privileges on the project.

Roles

If Oracle Data Integrator is configured to use **external authentication** (see "Configuring External Authentication" on page 25-17), you can leverage the **enterprise role integration** feature in Oracle Data Integrator. Enterprise role integration in Oracle Data Integrator is based on the authorization model in Oracle Platform Security Services (OPSS). This feature allows you to map enterprise roles to Oracle Data Integrator roles, enabling enterprise users of a particular enterprise role to access Oracle Data Integrator through the permissions granted to the Oracle Data Integrator roles in the mapping.

Enterprise role integration in Oracle Data Integrator enables you to:

- Create a set of Oracle Data Integrator roles.
- Grant Oracle Data Integrator privileges (instance level and type level) and profiles to Oracle Data Integrator roles.
- Map enterprise principals — that is, users or roles — defined in an external identity store to Oracle Data Integrator roles.
- Determine the privileges granted to a user who is authenticated into Oracle Data Integrator by evaluating the Oracle Data Integrator role to which the user is mapped. (Privileges are automatically calculated during authentication.)

For an introduction to enterprise roles, see "Understanding Users and Roles" in *Securing Applications with Oracle Platform Security Services*. For details about how enterprise role integration works in Oracle Data Integrator and how you can manage roles in Oracle Data Integrator Studio, see "Mapping Principals Defined in an Identity Store to Oracle Data Integrator Roles" on page 25-26.

Setting up a Security Policy

This section explains how to use the different security capabilities to enforce security in Oracle Data Integrator.

This section contains the following topics:

- Security Policy Approaches
- Managing Profiles
- Managing Users
- Managing Privileges

Security Policy Approaches

Two main approaches are available for defining the security in Oracle Data Integrator:

- The strongly secured approach, where users have no default authorizations on objects. This approach uses only nongeneric profiles. The security administrator must grant users authorizations on object instances. This policy is complex to configure because it requires managing privileges by instance.
- The generic approach, where users inherit the privileges of the profiles they have. This policy is suitable in most cases and is simple to configure.

To implement a strongly secured approach:

1. Create the users.
2. Give the users nongeneric profiles (built-in or customized).
3. Grant the users privileges on object instances after those instances are created. This operation must be repeated for every new instance.

To implement a generic approach:

1. Create the users.
2. Give the users the generic profiles (built-in or customized).

Note: It is possible to combine the two approaches by simultaneously using generic and non generic profiles. For example, by using DESIGNER and NG_METADATA_ADMIN for the users, you manage projects in a generic approach, and manage models in a strongly secured approach.

Managing Profiles

You can create, delete, or duplicate profiles. Creating and duplicating profiles enables you to customize the profiles assigned to users. The following topics provide the steps for performing these procedures in Oracle Data Integrator Studio.

Creating a New Profile

To create a profile:

1. In Security Navigator, expand the **Profiles** accordion.
2. Click **New Profile** in the toolbar of the **Profiles** accordion.
3. In the **Name** field, enter a name for your profile.

4. From the **File** main menu, select **Save**.

The new profile appears.

Duplicating a Profile

To duplicate a profile:

1. In Security Navigator expand the **Profiles** accordion.
2. Select the profile that you want to duplicate from the list of profiles.
3. Right-click and select **Duplicate**.

A new profile appears, which is a copy of the original profile.

Deleting a Profile

To delete a profile:

1. In Security Navigator expand the **Profiles** accordion.
2. Select the profile that you want to delete from the list of profiles.
3. Right-click and select **Delete**.
4. Click **OK** in the Confirmation dialog.

The profile disappears from the list. All users granted with this profile lose the privileges attached to this profile.

Managing Users

The way in which you manage users depends, in part, on where the user is defined. By default, a user corresponds to the login name used to connect to a repository and is persisted in the internal repository of Oracle Data Integrator. However, if external authentication is enabled for Oracle Data Integrator components:

- Users authenticated into Oracle Data Integrator are created and managed in an external identity store, such as Oracle Internet Directory. You use the tools and utilities provided with that store to create, update, and delete users.
- To let external users access Oracle Data Integrator, you can do any of the following:
  - Map the external user to an Oracle Data Integrator role.
  - Map any of the enterprise roles to which this external user belongs to an Oracle Data Integrator role.
  - Register the external user in Oracle Data Integrator using the Security Navigator as in prior releases.
- You can assign profiles or other security privileges directly to external users who are registered in Oracle Data Integrator. Or you can assign profiles or other security privileges to an Oracle Data Integrator role, and then map external users (or any of the external users' enterprise roles) to that Oracle Data Integrator role. By doing the latter, an external user can inherit all the profiles or privileges granted to the Oracle Data Integrator role.

  You can also map an enterprise user (or the user's enterprise roles) to multiple Oracle Data Integrator roles. When a user is authenticated, Oracle Data Integrator calculates the user's privileges as a union of all the privileges granted to all the Oracle Data Integrator roles in the mapping.

For more information about external authentication, see "Configuring External Authentication" on page 25-17). For more information about enterprise role integration, see "Mapping Principals Defined in an Identity Store to Oracle Data Integrator Roles" on page 25-26.

Creating a New User in the Internal Repository

To create a user in the internal repository of Oracle Data Integrator:

1. In Security Navigator expand the **Users** accordion.
2. Click **New User** in the toolbar of the **Users** accordion.
3. In the **Name** field, enter a name for your user.
4. Provide the **Initials** for this user.
5. Do one of the following:
   - If using internal authentication, click **Enter the Password** and provide a password for this user.
   - If using external authentication, click **Retrieve GUID** to associate the new user with a user from the external identity store. The user name in Oracle Data Integrator must match the user name in the identity store. If a match is found, a Global Unique ID appears in the **External GUID** field.
6. From the **File** main menu, select **Save**.

The new user appears in the **Users** accordion.

Assigning a Profile to a User

To assign a profile to a user:

1. In Security Navigator expand the **Users** and the **Profiles** accordions.
2. Select the profile that you want to assign, then drag it on the user you want to assign it to.
3. Click **OK** in the Confirmation dialog.

The profile appears under the profiles of this user. The user is immediately granted the privileges attached to this profile.

Note: If external authentication is enabled, you can assign profiles to roles. Users who are defined in a given Oracle Data Integrator role are granted all the profiles assigned to that role. Enterprise role integration gives you a more convenient and coarse-grained approach to managing the privileges granted to users. For more information, see "Assign Privileges or Profiles to Oracle Data Integrator Roles" on page 25-31.

Removing a Profile from a User

To remove a profile from a user:

1. In Security Navigator expand the **Users** accordions.
2. Expand the **Profiles** node under the user.
3. Right-click the profile to be removed.
4. Select **Delete**.

5. Click **OK** in the Confirmation dialog.

The profile disappears from the list of profiles assigned to this user. All privileges granted to this user by this profile are removed.

Note: If external authentication is enabled, you can remove profiles from roles. Users who are defined in a given Oracle Data Integrator role are granted only the profiles assigned to that role. For more information, see "Assign Privileges or Profiles to Oracle Data Integrator Roles" on page 25-31.

Deleting a User from the Internal Repository

To delete a user that is defined in the internal repository of Oracle Data Integrator:

1. In Security Navigator expand the **Users** accordion.
2. From the list of users, select the user that you want to delete.
3. Right-click and select **Delete**.
4. Click **OK** in the Confirmation window.

The user disappears from the list.

Managing Privileges

After creating users and profiles, you can grant privileges to these users and profiles. You can grant profile methods and user methods that apply to object types, and you can also grant authorizations by object instances (for users only) that apply to specific object instances.

Note: If external authentication is enabled for Oracle Data Integrator components, you can manage privileges for users more conveniently by using the enterprise role integration feature. For more information, see "Using the Roles Editor" on page 25-29.

Granting a Profile Method or User Method

To grant a profile method or user method:

1. In Security Navigator expand the **Users** or **Profiles** accordion.
2. Expand the **Objects** accordion, and expand the node of the object for which you want to grant a privilege.
3. Select the method that you want to grant, then drag it on the user or profile you want to grant the method to.
4. Click **OK** in the Confirmation window.

The method is granted to the user or the profile. The method appears either under the objects node of this user or under the profile.

Note: You can grant privileges on all the methods of an object by dragging the object itself onto the user or profile instead of dragging one of its methods.

Revoking a Profile Method or User Method

To revoke a profile method or user method:

1. In Security Navigator expand the **Users** or **Profiles** accordions.
2. Expand the **Profiles** or the **Objects** node under the user for which you want you revoke privileges, then expand the object whose method needs to be revoked.
3. Right-click the method and then select **Delete**.
4. Click **OK** in the Confirmation dialog.

The method is revoked from the user or the profile. It disappears from the methods list under the objects node of this user or profile.

Granting an Authorization by Object Instance

To grant an authorization by object instance to a user:

1. In Security Navigator expand the **Users** accordion.
2. In the Designer, Operator, or Topology Navigator, expand the accordion containing the object to which you want to grant privileges.
3. Select this object, then drag it onto the user in the **Users** accordion. The authorization by object instance editor appears. This editor shows the list of methods available for this instance and the instances it contains. For example, if you grant privileges on a project instance, the folders and mappings contained in this project appear in the editor.
4. Fine tune the privileges granted per object and method. You may want to implement the following simple privilege policies on methods that you select from the list:
   - To grant all these methods in all repositories, click **Allow selected methods in all repositories**.
   - To deny all these methods in all repositories, click **Deny selected methods in all repositories**.
   - To grant all these methods in certain work repositories, click **Allow selected methods in selected repositories**, then select the repositories from the list.
5. From the **File** main menu, select **Save**.

Note: Only certain objects support authorization by object instance. These object types are listed under the **Instances** node for each user.

Methods for which the user has generic privileges are not listed in the Object Instance Editor.

Revoking an Authorization by Object Instance

To revoke an authorization by object instance from a user:

1. In Security Navigator expand the **Users** accordion.
2. Expand the **Instances** node under the user for which you want you revoke privileges.
3. Right-click the instance from which you want to revoke an authorization, and then select **Delete**.
4. Click **OK** in the Confirmation dialog.

The authorizations on this object instance are revoked from the user.

Note: You can also revoke privileges per method by editing the Authorization by Object instance and denying certain methods to this user. After this operation, if the user no longer has any privilege on an instance, the instance automatically disappears from the tree in Security Manager.

Cleaning up Unused Authorizations

Authorizations by object instance are stored in the master repository. However, if objects are deleted from all work repositories, the authorization are not necessarily deleted. You may wish to retain certain unused authorizations; for example, if they refer to objects currently stored in an exported file or in a stored solution.

You should use the Security Clean-up Tool periodically to remove these unused authorizations from the master repository. Unused authorizations are removed if they refer to objects that do not exist in the master repository or in any work repository.

Note: All work repositories attached to the master repository must be accessible so that you can verify the existence of the objects in those repositories

---

To clean up unused authorizations:

1. From the Security Navigator toolbar menu, select **Clean Up Security Settings...**. The Security Clean-up Tool dialog appears.
2. On the **Cleanable** tab, select **Clean** for the cleanable security settings you want to remove. The security settings that cannot be removed are shown on the **Non-cleanable** tab.
3. Click **OK** to clean up the selected security settings.

Using a Password-Protected Wallet for Storing Login Credentials

Prior to logging in to the Oracle Data Integrator Studio internal repository for the first time, you must create a login name for which you specify your Oracle Data Integrator login credentials. These credentials include your Oracle Data Integrator user name and password, and also the Oracle Data Integrator master repository database user name and password. Oracle Data Integrator can save these credentials in either an encrypted login XML file or a password-protected wallet. By default, they are saved in a password-protected wallet.

The use of password-protected wallets for storing database login credentials is strongly recommended by Oracle. To generate password-protected wallets, Oracle Data Integrator leverages the Oracle Wallet features in Oracle Platform Security Services (OPSS). Password-protected wallets use a strong encryption algorithm to secure the wallet's contents.

Note the following about using the password-protected wallet in Oracle Data Integrator Studio:

- When you create the wallet, you specify the wallet password and the days until the password expires. Oracle Data Integrator then stores the internal repository login credentials in the wallet.

- When you connect to an Oracle Data Integrator repository, you need only to enter the wallet password. All repository login credentials are then automatically retrieved from the wallet.
- The wallet file is named `ewallet.p12` and is placed in the following directory:

  Windows: `%APPDATA%\odi\oracledi\ewallet`

  UNIX: `$HOME/.odi/oracledi/ewallet`

  > **Note:** Depending on your environment, you might first need to create the root directory for the wallet. For more information, see "Creating the Password-Protected Wallet" on page 25-11.

- When the wallet password expires, Oracle Data Integrator Studio automatically prompts you to enter a new one when you attempt to log in to Studio.
- You can change the wallet password and expiration at any time from Studio.

The following sections explain how to create and use the password-protected wallet for storing Oracle Data Integrator login credentials:

- Creating the Password-Protected Wallet
- Permitting Repository Access to Other Users
- Changing the Wallet Password
- Customizing How Login Credentials Are Saved

Creating the Password-Protected Wallet

When logging in to Studio for the first time, you can create the password-protected wallet using the following steps:

1. Create the root directory on your machine for storing the wallet, if it does not already exist.

   Windows:

   On Windows machines, you must create the root directory path `odi\oracledi` in the location represented by the environment variable `%APPDATA%`. For example:

   ```
   c:\> mkdir %APPDATA%\odi\oracledi
   ```

   UNIX:

   On UNIX machines, you must create the root directory path `.odi/oracledi` in the location represented by the environment variable `$HOME`. For example:

   ```
   prompt> mkdir -p $HOME/.odi/oracledi/
   ```

2. Change to the *`ORACLE_HOME`*`/odi/studio` directory.
3. Launch Oracle Data Integrator Studio by running the following command:

   Windows:

   ```
   odi.exe
   ```

   UNIX:

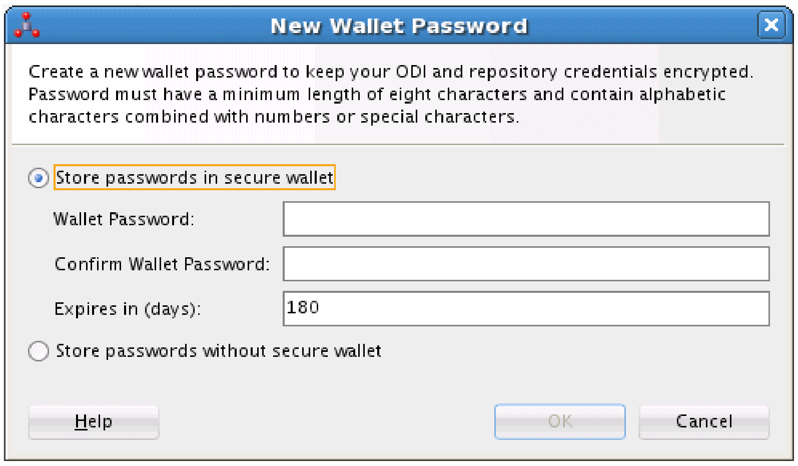
   ```
   ./odi.sh
   ```

The Studio main window is displayed.

4. Click **Connect to Repository...**

   The New Wallet Password dialog box is displayed, shown in Figure 25–1.

Figure 25–1 New Wallet Password Dialog Box



Create a new wallet password to keep your ODI and repository credentials encrypted. Password must have a minimum length of eight characters and contain alphabetic characters combined with numbers or special characters.

[✓ Store passwords in secure wallet]

Wallet Password:

Confirm Wallet Password:

Expires in (days): 180

Store passwords without secure wallet

5. Choose the default option, **Store passwords in a secure wallet**, then enter the password in the **Wallet Password** and **Confirm Wallet Password** fields.

   Optionally, you can change the password expiration.

6. Click **OK**.

   You are then prompted to enter login credentials for the master repository. For more information about the required credentials, see "Creating the Master Repository" on page 3-4.

Subsequently, each time you log in to Studio and connect to a repository, you are prompted for the wallet password.

Permitting Repository Access to Other Users

To give Oracle Data Integrator users access to the repository, you can distribute the password-protected wallet to them in either of the following ways:

- Provide each user with a copy of the `ewallet.p12` file you created, along with the password you specified.
- Create an individual password-protected wallet for each Oracle Data Integrator user. This allows you to set a different password for each wallet.

When creating a wallet for other users, be sure to do the following:

1. Prior to creating a new wallet, move the existing `ewallet.p12` file to a different directory.
2. When creating a wallet for other users, set the expiration to `0` days. This way, when users connect to the repository for the first time, they are prompted to set a new password for their wallets (see "Changing the Wallet Password" on page 25-13).

All password-protected wallet files for Oracle Data Integrator Studio must be named `ewallet.p12` and must be placed in the `~oracledi/ewallet` directory on each machine from which it is used, as explained in "Creating the Password-Protected Wallet" on page 25-11.

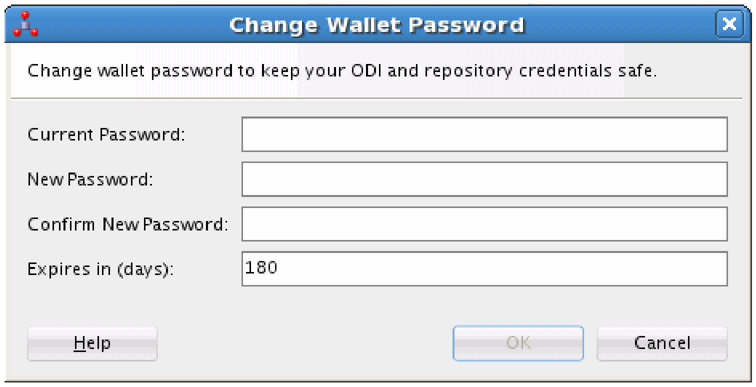
Changing the Wallet Password

When the password-protected wallet expires, you are automatically prompted to enter a new one when you try to connect to a repository. However, you can also change the wallet password from Studio at any time, as follows:

1. Launch Oracle Data Integrator Studio.
2. From the **ODI** menu, choose **Change Wallet Password...**

   The Change Wallet Password dialog box is displayed, shown in Figure 25–2.

Figure 25–2 Change Wallet Password Dialog Box



Change Wallet Password

Change wallet password to keep your ODI and repository credentials safe.

Current Password:

New Password:

Confirm New Password:

Expires in (days): 180

3. Enter the current password and the new password, then confirm your new password.

   Optionally, you can also change the expiration.
4. Click **OK**.

Customizing How Login Credentials Are Saved

Oracle Data Integrator Studio provides the following options for storing login credentials:

- You can enable or disable the saving of login credentials.
- If login credentials are being saved, you can choose between saving them in a password-protected wallet or a login XML file.

To change whether or how login credentials are saved, complete the following steps:

1. Launch Oracle Data Integrator Studio.
2. From the **Tools** menu, choose **Preferences...**

   The Preferences dialog box is displayed.
3. In the Preferences dialog box, expand the **ODI** node, then select **User Interface**.

   The Preferences dialog box, shown in Figure 25–3, displays the following options for saving login credentials:

   - **Save Login Credentials** — Select this option to save credentials.

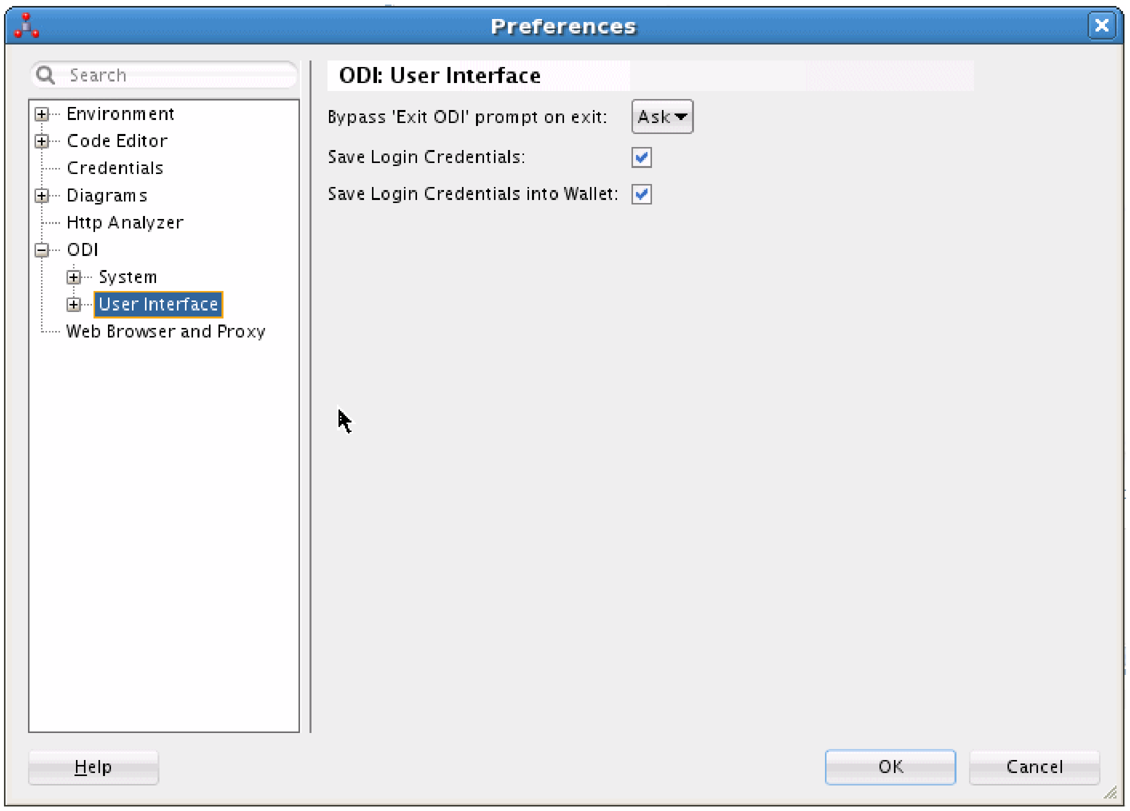
- **Save Login Credentials into Wallet** — Select this option to store the Oracle Data Integrator login credentials in a password-protected wallet.

  If **Save Login Credentials** is selected, but **Save Login Credentials into Wallet** is deselected, login credentials are saved in a login XML file.

  By default, both options are selected, which is the setting recommended by Oracle.

4. If you change any preferences, click **OK**.

Figure 25–3 Preferences Dialog Box for Saving Login Credentials



ODI: User Interface

Bypass 'Exit ODI' prompt on exit: Ask

Save Login Credentials: ✓

Save Login Credentials into Wallet: ✓

Setting Up External Password Storage

Oracle Platform Security Services (OPSS) offers the standard Java Security Model services for authentication and authorization.

Oracle Data Integrator stores by default all security information in the master repository. This password storage option is called **internal password storage**.

Oracle Data Integrator can optionally use OPSS for storing critical security information. When using OPSS with Oracle Data Integrator, the data server passwords and context passwords are stored in the OPSS Credential Store Framework (CSF). This password storage option is called **external password storage**.

Note: When using external password storage, other security details such as user names, password, and privileges remain in the master repository. It is possible to externalize the authentication and have users and password stored in an identity store using external authentication. **Note also:** The *external password storage* option is unrelated to the *external authentication* feature. For more information about external authentication, see "Configuring External Authentication" on page 25-17.

To use the external password storage option:

1. You need to install a WebLogic Server instance configured with OPSS.
2. All Oracle Data Integrator components, including the run-time agent, need to have access to the remote credential store.

See "Configuring Java EE Applications to Use OPSS" in *Securing Applications with Oracle Platform Security Services* for more information.

Setting the Password Storage

There are four ways to set or modify the password storage:

- Importing the master repository (see "Importing the Master Repository" on page 20-18) allows you to change the password storage.
- Creating the master repository (see "Creating the Master Repository" on page 3-4) allows you to define the password storage.
- Switching the Password Storage modifies the password storage for an existing master repository.
- Recovering the Password Storage allows you to recover from a credential store crash.

Switching the Password Storage

Switching the password storage of the Oracle Data Integrator repository changes how data servers and contexts passwords are stored. This operation must be performed by a SUPERVISOR user.

Use the **Switch Password Storage** wizard to change the password storage options of the data server passwords.

Before launching the Switch Password Storage wizard perform the following tasks:

- Disconnect Oracle Data Integrator Studio from the repository.
- Shut down every component using the Oracle Data Integrator repository.

To launch the Switch Password Storage wizard:

1. From the **ODI** main menu, select **Password Storage** > **Switch**...
2. Specify the login details of your Oracle Data Integrator master repository as defined when connecting to the master repository (see: "Connecting to the Master Repository" on page 3-5).
3. Click **Next**.
4. Select the password storage:

- Select **Internal Password Storage** if you want to store passwords in the Oracle Data Integrator repository.
- Select **External Password Storage** if you want use OPSS Credential Store Framework (CSF) to store the data server and context passwords.

  If you select external password storage, you must provide the **MBean Server Parameters** to access the credential store described in Table 25–2, and then click **Test Connection** check the connection to the MBean Server.

Table 25–2 MBean Server Parameters

Server	Host	JMX Port	User	Password
From the list, select the application server.	MBeans server host. For example, *mymachine.oracle.com*	MBeans Server Port. For example, 7001	MBeans Server user name. For example, *weblogic*	MBean server password

5. Click **Finish**.

The password storage options have been changed. You can now reconnect to the Oracle Data Integrator repository.

Recovering the Password Storage

Oracle Data Integrator offers a password recovery service that should be used only in case of an external password storage crash. Using this procedure, password storage is forced to internal password storage because external storage is no longer available. This operation should be performed by a supervisor user.

CAUTION: When performing a password storage recovery, passwords for context, data servers, JDBC password of the work repository and Enterprise Scheduler related passwords are lost and need to be re-entered manually in Topology Navigator.

Use the **Recover Password Storage** wizard to start the password recovery.

To launch the Recover Password Storage wizard:

1. From the **ODI** main menu, select **Password Storage** > **Recover**...
2. Specify the login details of your Oracle Data Integrator master repository defined when connecting to the master repository (see: "Connecting to the Master Repository" on page 3-5).
3. Click **Finish**.
4. Re-enter manually data server and context passwords. Refer to Chapter 4, "Setting Up a Topology," for more information.

Managing the Authentication Mode

By default, Oracle Data Integrator users are authenticated using the identity information contained in the master repository. However, Oracle Data Integrator users can be authenticated using an external authentication service instead: Using Oracle Platform Security Services (OPSS), Oracle Data Integrator users are authenticated against an external enterprise identity store (for example, an LDAP server such as Oracle Internet Directory or Microsoft Active Directory), which contains enterprise users and enterprise roles in a central place.

When external authentication is enabled, the master repository retains the Oracle Data Integrator specific privileges. External users do not need to be created in Oracle Data Integrator's internal repository. Instead, external user credentials rely on a centralized identity store, and authentication always takes place against this external store.

The authentication mode (internal or external) can be defined when you create the repository, and can also be switched for existing repositories, as explained in "Setting the Authentication Mode in the Master Repository" on page 25-17.

For details about configuring external authentication, see "Configuring External Authentication" on page 25-17.

Setting the Authentication Mode in the Master Repository

There are two ways to set the authentication mode:

- When you create the master repository. The Master Repository Creation Wizard contains options for setting the authentication mode. For information about how to set external authentication when creating the master repository, see "Set External Authentication when Creating the Master Repository" on page 25-20. For general information, see also "Creating the Master Repository" on page 3-4.
- When you switch the authentication mode from Studio. The Switch Authentication Mode Wizard allows you to change the authentication mode used by an existing master repository. For detailed steps to switch to external authentication mode in Studio, using this wizard, see "Switching an Existing Master Repository to External Authentication Mode" on page 25-21.

Note: Before you can select external authentication mode in the master repository, you must configure external authentication, as explained in "Configuring External Authentication" on page 25-17.

Configuring External Authentication

By default, Oracle Data Integrator stores all user information as well as users' privileges in the master repository. When a user logs to Oracle Data Integrator, it logs against the master repository. This authentication method is called **internal authentication**.

However, you can customize Oracle Data Integrator to use Oracle Platform Security Services (OPSS) to authenticate users that are defined in an enterprise identity store, which is typically an LDAP server such as Oracle Internet Directory or Microsoft Active Directory. The identity store contains enterprise user information and enterprise roles. Such an identity store is used at the enterprise level by applications to have centralized user definitions and to support single sign-on (SSO). In Oracle Data Integrator, this authentication method is called **external authentication**. Configuring external authentication enables you to leverage the standard Java Security Model authentication and authorization services that are provided by OPSS.

To use external authentication, you need to configure an enterprise identity store and also configure each Oracle Data Integrator component to refer to that identity store.

Note: When using external authentication, only users and passwords are externalized. Oracle Data Integrator privileges remain within the repository.

The following sections explain how to set up external authentication for the Studio, and the Standalone Agent:

- Configuring Oracle Data Integrator Studio for External Authentication
- Switching an Existing Master Repository to External Authentication Mode
- Reactivating Users After Switching to External Authentication
- Configuring the Standalone Agent for External Authentication
- Mapping Principals Defined in an Identity Store to Oracle Data Integrator Roles

Configuring Oracle Data Integrator Studio for External Authentication

To configure Oracle Data Integrator Studio with external authentication, you must do the following:

1. Set Up the OPSS Configuration File
2. Create a Wallet File for an LDAP Bootstrap User
3. Set External Authentication when Creating the Master Repository

Set Up the OPSS Configuration File

The configuration to connect and use the identity store is contained in the OPSS configuration file, called `jps-config-jse.xml`. To configure Oracle Data Integrator components with external authentication, you must set up this configuration file to correspond to the external identity store that you plan to use.

To set up the OPSS configuration file, complete the following steps:

1. Create the `odi/oracledi` directory on your machine, if it does not already exist. For example:

   Windows:

   ```
   c:\> mkdir %APPDATA%\odi\oracledi
   ```

   UNIX:

   ```
   prompt> mkdir -p $HOME/.odi/oracledi/
   ```

2. Copy the following files into the `odi/oracledi` directory:

   - *`ORACLE_HOME`*`/oracle_common/modules/oracle.jps_12.1.2/domain_config/jse/jps-config.xml`
   - *`ORACLE_HOME`*`/oracle_common/modules/oracle.jps_12.1.2/domain_config/jse/system-jazn-data.xml`

3. Change to the `odi/oracledi` directory.
4. Rename `jps-config.xml` to `jps-config-jse.xml`.
5. Open `jps-config-jse.xml` in an editor.
6. Add a service instance for the identity store provider and include the following properties:

   - The service instance name. For example, for Oracle Internet Directory, `name="idstore.oid"`.
   - The `idstore.type` property.
   - The `bootstrap.security.principal.map` property.

- The `bootstrap.security.principal.key` property.

  For an example service instance configuration, see Example 25–1.

  For details about specifying these properties for your identity store, see "LDAP Identity Store Properties" in *Securing Applications with Oracle Platform Security Services*.

7. In the default jps context, change the `serviceInstanceRef name="idstore.xml"` value to `"idstore.<your-idstore-type>"`, as in the following example that sets the identity store type for Oracle Internet Directory:

```
 <serviceInstanceRef ref="idstore.oid"/>
```

8. Comment out the `keystore` and `audit` service instance references in the default jps context. For example:

```
<jpsContext name="default">
 <serviceInstanceRef ref="credstore"/>
<!--  <serviceInstanceRef ref="keystore"/>  -->
<serviceInstanceRef ref="policystore.xml"/>
<!--  <serviceInstanceRef ref="audit"/>  -->
```

9. Set any additional property attribute values as appropriate for your identity store. For complete details about the OPSS configuration file, see "Configuring Java SE Applications to Use OPSS" in *Securing Applications with Oracle Platform Security Services* for more information.

Example 25–1 shows the configuration of Oracle Internet Directory in the service instance section of the configuration file.

Example 25–1 Example Service Instance Configuration

```
<!-- OID LDAP Identity Store Service Instance -->
  <serviceInstance name="idstore.oid" provider="idstore.ldap.provider">
  <property name="subscriber.name" value="dc=us,dc=oracle,dc=com" />
  <property name="idstore.type" value="OID" />
   <property name="bootstrap.security.principal.map" value="BOOTSTRAP_JPS"/>
   <property name="bootstrap.security.principal.key" value="bootstrap_idstore"/>
  <property name="username.attr" value="uid"/>
  <property name="groupname.attr" value="cn"/>
  <property name="ldap.url" value="ldap://hostname:port" />
  <extendedProperty>
   <name>user.search.bases</name>
   <values>
    <value>cn=users,dc=us,dc=oracle,dc=com</value>
   </values>
  </extendedProperty>
  <extendedProperty>
   <name>group.search.bases</name>
   <values>
    <value>cn=groups,dc=us,dc=oracle,dc=com</value>
   </values>
  </extendedProperty>
 </serviceInstance>
   .
   .
   .
 <serviceInstanceRef ref="idstore.oid"/>
```

Create a Wallet File for an LDAP Bootstrap User

You must create a wallet file to store the identity store bootstrap user's credentials. This wallet file is referenced from the OPSS configuration file and is used by Oracle Data Integrator to establish the connection to the identity store that is configured in the OPSS configuration file.

To create this wallet file, complete the following steps:

1. Change to the `ORACLE_HOME/odi/sdk/bin` directory.
2. Run the `odi_credtool` script.

   You are prompted to enter the following parameters:

For the following parameter . . .	. . . enter the following value
`User name`	The Distinguished Name (DN) of the bootstrap user account used to connect to the identity store with necessary privileges. For example, for Microsoft Active Directory, specify this user as follows: `CN=Administrator,CN=Users,DC=ad,DC=vm,DC=`*`mycompany`*`,DC=com`
`Password`	The password for the bootstrap user account that is used to connect to the identity store.

After you enter the correct credentials for the bootstrap user account, the following message is displayed:

```
The credentials have been successfully added to the boostrap credential store.
```

Set External Authentication when Creating the Master Repository

To create the master repository and set it to use external authentication:

1. Change to the `ORACLE_HOME/odi/studio` directory.
2. Launch Oracle Data Integrator Studio by running the following command:

   Windows:

   ```
   odi.exe
   ```

   UNIX:

   ```
   ./odi.sh
   ```

3. In Oracle Data Integrator Studio, select **File** > **New**, select **Master Repository Creation Wizard**, and click **OK**.
4. In the repository selection screen of the Master Repository Creation Wizard, enter the connection parameters for the database that is to contain the Master Repository, and click **Test Connection** to ensure the parameters are correct. For details about configuring this database, see "Configuring Oracle Data Integrator in a WebLogic Server Domain" in *Installing and Configuring Oracle Data Integrator.*

   When the connection test is successful, click **OK**.
5. In the authentication screen of the Master Repository Creation Wizard, select **Use External Authentication**.

For information about how to select principals who have supervisor privileges in the master repository, see "Mapping Principals Defined in an Identity Store to Oracle Data Integrator Roles" on page 25-26.

For information about how to update an existing master repository to use external authentication, see "Switching an Existing Master Repository to External Authentication Mode" on page 25-21.

Switching an Existing Master Repository to External Authentication Mode

To switch the authentication mode from internal to external in an existing master repository, complete the following steps:

1. Change to the *ORACLE_HOME*/odi/studio directory.
2. Launch Oracle Data Integrator Studio by running the following command:

   Windows:

   ```
   odi.exe
   ```

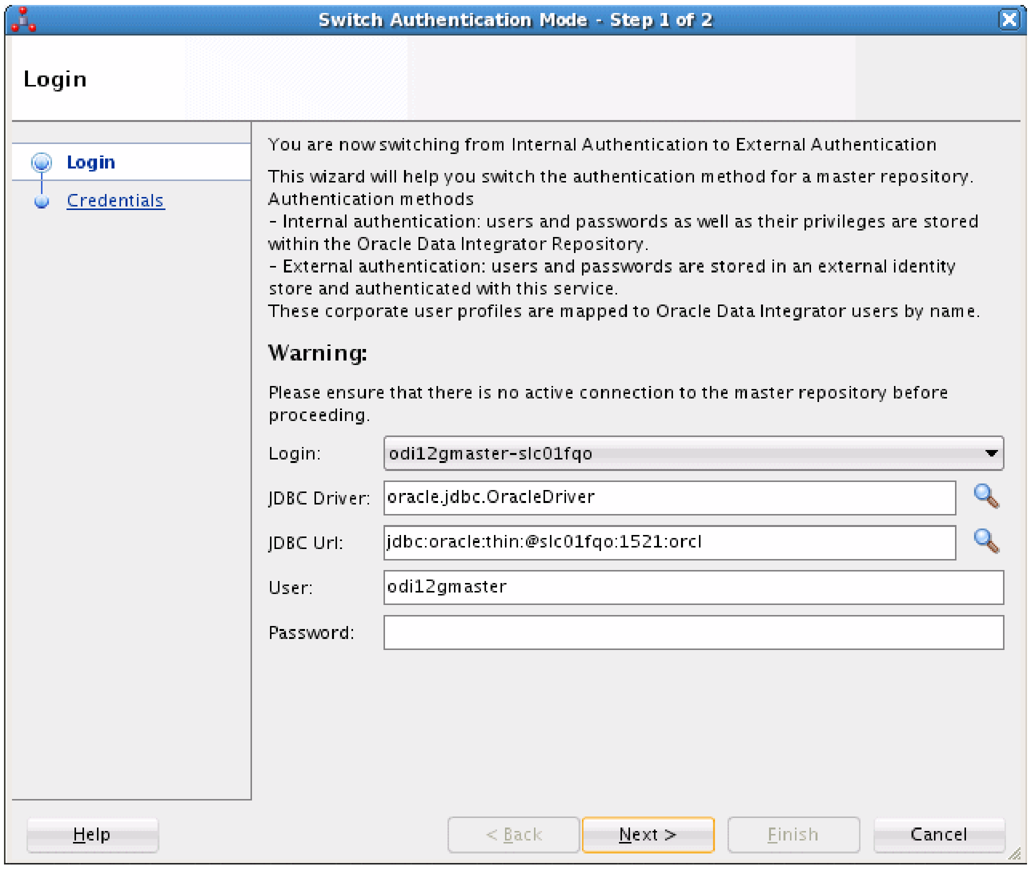
   UNIX:

   ```
   ./odi.sh
   ```

3. In Oracle Data Integrator Studio, select **ODI** > **Switch Authentication Mode...**.

   The Login screen of the Switch Authentication Mode wizard is displayed, as shown in Figure 25–4.

Figure 25–4 Login Screen of Switch Authentication Mode Dialog Box



You are now switching from Internal Authentication to External Authentication

This wizard will help you switch the authentication method for a master repository.
Authentication methods
- Internal authentication: users and passwords as well as their privileges are stored within the Oracle Data Integrator Repository.
- External authentication: users and passwords are stored in an external identity store and authenticated with this service.
These corporate user profiles are mapped to Oracle Data Integrator users by name.

Warning:

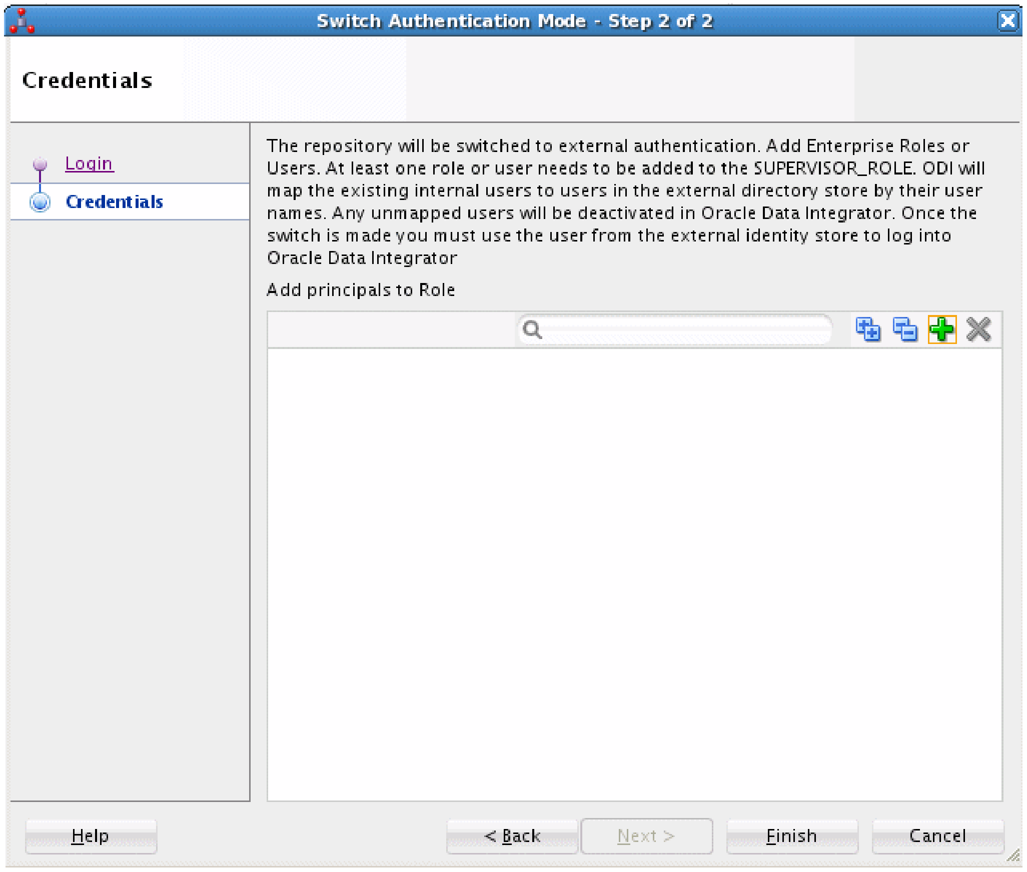
Please ensure that there is no active connection to the master repository before proceeding.

Login: odi12gmaster-slc01fqo
JDBC Driver: oracle.jdbc.OracleDriver
JDBC Url: jdbc:oracle:thin:@slc01fqo:1521:orcl
User: odi12gmaster
Password:

4. Enter the login name of the master repository, the JDBC connection parameters, and the user name and password for the master repository database user, then click **Next**.

   The Credentials screen of the Switch Authentication Mode wizard is displayed, as shown in Figure 25–5.

Figure 25–5 Credentials Screen of Switch Authentication Mode Wizard

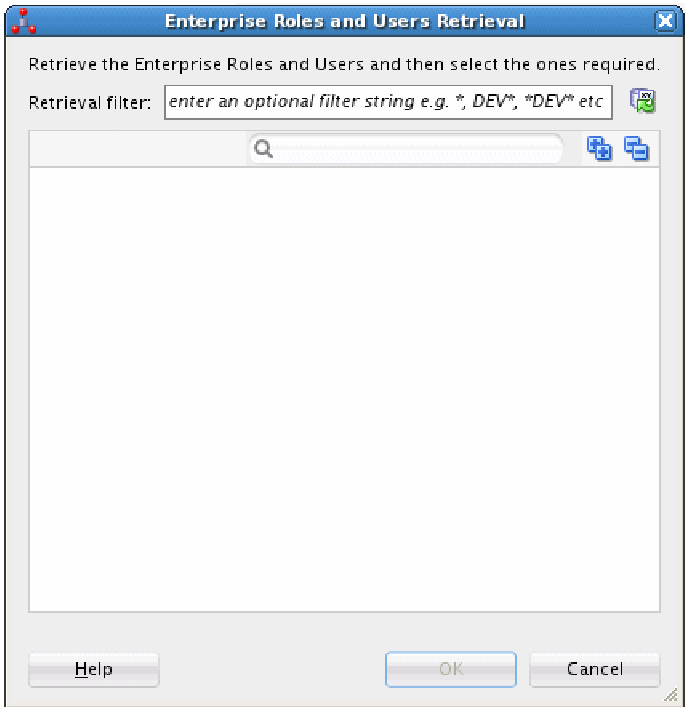


The repository will be switched to external authentication. Add Enterprise Roles or Users. At least one role or user needs to be added to the SUPERVISOR_ROLE. ODI will map the existing internal users to users in the external directory store by their user names. Any unmapped users will be deactivated in Oracle Data Integrator. Once the switch is made you must use the user from the external identity store to log into Oracle Data Integrator

Add principals to Role

5. Click the **Add** button, which is shown as the green plus symbol adjacent to the search field, to display the Enterprise Roles and Users Retrieval dialog box, shown in Figure 25–6.

Figure 25–6 Enterprise Roles and Users Retrieval Dialog Box


Enterprise Roles and Users Retrieval
Retrieve the Enterprise Roles and Users and then select the ones required.
Retrieval filter:
enter an optional filter string e.g. *, DEV*, *DEV* etc
Help
OK
Cancel

You can use the Enterprise Roles and Users Retrieval dialog box to either:

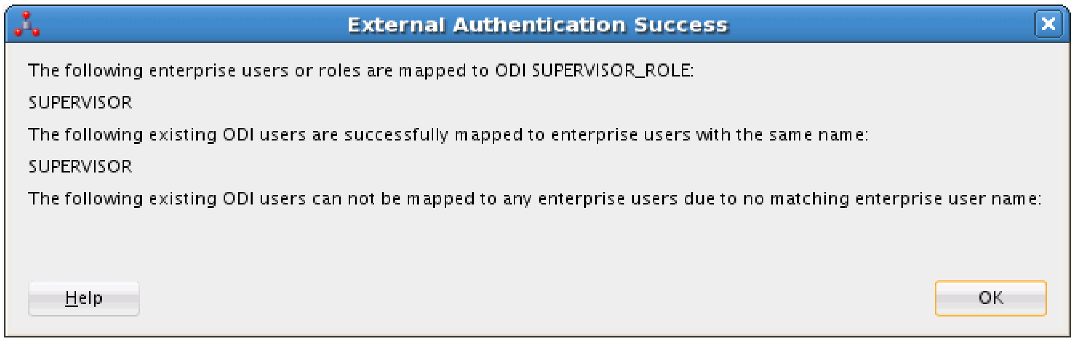
- Enter a filter expression to display the roles and users in the identity store that match the filter.
- Search the identity store for a specific role or user name.

You can expand the Enterprise Roles and Enterprise Users nodes to browse the available enterprise roles and users in the identity store.

6. Select the user or role you wish to assign to the `SUPERVISOR_ROLE` role, then click **Finish**.

If the external authentication mode switch is successful, a dialog box similar to the one in Figure 25–7 is displayed:

Figure 25–7 Successful Switch to External Authentication Mode Message Box



The following enterprise users or roles are mapped to ODI SUPERVISOR_ROLE:

SUPERVISOR

The following existing ODI users are successfully mapped to enterprise users with the same name:

SUPERVISOR

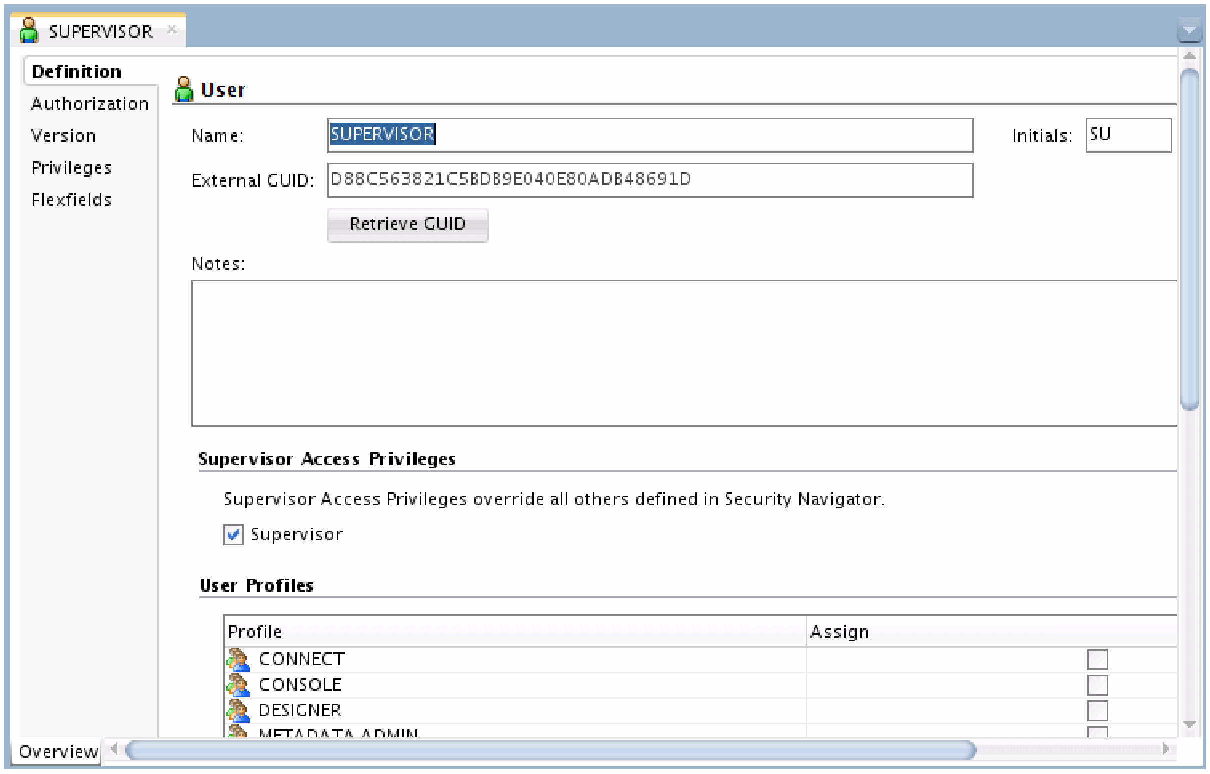
The following existing ODI users can not be mapped to any enterprise users due to no matching enterprise user name:

Reactivating Users After Switching to External Authentication

If you have an existing set of Oracle Data Integrator users defined, you can re-enable them after switching to external authentication as explained in this section. To re-enable users, first reconnect to Studio as a user with supervisor privileges—for example, `SUPERVISOR`—using the password specified in the external identity store, and not the one stored in the internal Oracle Data Integrator repository. Then complete the following steps:

1. In the Security Navigator, expand the **Users** accordion.
2. From the list of users displayed, select the user that you want to re-enable.
3. Right-click and select **Open**. The User editor appears, shown in Figure 25–8.

Figure 25–8 Open USER Administrator Dialog Box



Name: SUPERVISOR
Initials: SU
External GUID: D88C563821C5BDB9E040E80ADB48691D

Notes:

Supervisor Access Privileges

Supervisor Access Privileges override all others defined in Security Navigator.

[✓] Supervisor

User Profiles

Profile	Assign
CONNECT	☐
CONSOLE	☐
DESIGNER	☐
METADATA_ADMIN	☐

The **Name** field displays the user you selected in Step 2.

4. Click **Retrieve GUID**. If the user name has a match in the identity store, this external user's GUID appear in the External GUID field.
5. From the **File** main menu, select **Save** and disconnect.
6. Reconnect as this user (for example, the user `SUPERVISOR` shown in Figure 25–8) using the password in the external identity store.

You should now be able to connect to Oracle Data Integrator Studio using the external authentication.

Note the following:

- For troubleshooting LDAP server issues, it is very useful to use any LDAP client to browse the LDAP tree. You can download an LDAP client from the following URL:

http://www.ldapbrowser.com/

- For any LDAP directory other than Microsoft Active Directory, make sure that the property `user.filter.object.classes` is set correctly to the user's object class, which by default is set to `USER` if not specified.
- If Oracle Data Integrator Console and Java EE agent are installed in your environment, and you have switched Oracle Data Integrator Studio to external authentication, you might be unable to log in to Oracle Data Integrator Console and the Java EE agent also might fail. If this occurs, you must also configure Oracle Data Integrator Console and Java EE agent for external authentication using the same external identity store. The procedure for doing this is documented in Note 1510434.1 at My Oracle Support, available at the following URL:

  https://support.oracle.com/epmos/faces/DocumentDisplay?id=1510434.1

Configuring the Standalone Agent for External Authentication

To configure the Standalone Agent for external authentication, complete the following steps:

1. Copy the following files into the `DOMAIN_HOME/config/fmwconfig` directory, if they do not already exist there, where *`DOMAIN_HOME`* represents the root directory of your Oracle Data Integrator domain.

   - `ORACLE_HOME/oracle_common/modules/oracle.jps_12.1.2/domain_config/jse/jps-config.xml`
   - `ORACLE_HOME/oracle_common/modules/oracle.jps_12.1.2/domain_config/jse/system-jazn-data.xml`

   Note: For information about creating an Oracle Data Integrator domain, see "Configuring Oracle Data Integrator in a WebLogic Server Domain" in *Installing and Configuring Oracle Data Integrator*.

2. Change to the `DOMAIN_HOME/config/fmwconfig` directory.
3. Rename `jps-config.xml` to `jps-config-jse.xml`.
4. Open `jps-config-jse.xml` in an editor.
5. Add a service instance for the identity store provider and include the following properties:

   - The service instance name. For example, for Oracle Internet Directory, `name="idstore.oid"`.
   - The `idstore.type` property.
   - The `bootstrap.security.principal.map` property.
   - The `bootstrap.security.principal.key` property.

   For an example service instance configuration, see Example 25–1.

   For details about specifying these properties for your identity store, see "LDAP Identity Store Properties" in *Securing Applications with Oracle Platform Security Services*.

6. In the default jps context, change the `serviceInstanceRef` `name="idstore.xml"` value to `"idstore.<`*`your-idstore-type`*`>"`, as in the following example that sets the identity store type for Oracle Internet Directory:

```
 <serviceInstanceRef ref="idstore.oid"/>
```

7. Comment out the `keystore` and `audit` service instance references in the default jps context element. For example:

```
<jpsContext name="default">
 <serviceInstanceRef ref="credstore"/>
<!--  <serviceInstanceRef ref="keystore"/>  -->
<serviceInstanceRef ref="policystore.xml"/>
<!--  <serviceInstanceRef ref="audit"/>  -->
```

8. Save your changes to `jps-config-jse.xml` and exit from your editor.
9. Change to the *`DOMAIN_HOME`*`/bin` directory.
10. Run the `odi_credtool` script.

    You are prompted to enter the following parameters:

For the following parameter . . .	. . . enter the following value
`User name`	This is the Distinguished Name (DN) of the bootstrap user account used to connect to the identity store with Administrator privileges. For example, for Microsoft Active Directory, specify this user as follows: `CN=Administrator,CN=Users,DC=ad,DC=vm,DC=`*`mycompany`*`, DC=com`
`Password`	The password for the bootstrap user account that is used to connect to the identity store.

After you enter the correct credentials for the bootstrap user account, the following message is displayed:

```
The credentials have been successfully added to the boostrap credential store.
```

Mapping Principals Defined in an Identity Store to Oracle Data Integrator Roles

Oracle Data Integrator's enterprise role integration feature leverages the authorization model in Oracle Platform Security Services (OPSS) to allow you to map enterprise roles to Oracle Data Integrator roles. Enterprise users who belong to the mapped enterprise role can access Oracle Data Integrator by inheriting the privileges that are granted to the mapped Oracle Data Integrator role. Enterprise role integration simplifies both the management of users and also of privileges because you do not need to register users individually in Oracle Data Integrator to grant them access, and you do not need to manage privileges on a per user basis.

The following sections explain how enterprise role integration works and how you can use it to manage the privileges that are assigned to objects, instances, and profiles.

- How Enterprise Role Integration Works
- Defining and Managing Oracle Data Integrator Roles

How Enterprise Role Integration Works

When external authentication is enabled, users are managed in an external identity store, such as Oracle Internet Directory or Microsoft Active Directory.

When using enterprise role integration:

- Any enterprise principal in the identity store that can be mapped to an Oracle Data Integrator role is entitled to the privileges granted to that role. For example, if the enterprise role `DESIGNERS` is mapped to the Oracle Data Integrator role `DESIGNERS_ROLE`, any enterprise user who is a member of `DESIGNERS` in the identity store is given all the privileges granted to `DESIGNERS_ROLE` when authenticated into Oracle Data Integrator.
- You can map multiple enterprise users and roles to an Oracle Data Integrator role, and you can also map an enterprise user or role to multiple Oracle Data Integrator roles. (Note that you cannot map an Oracle Data Integrator role to another Oracle Data Integrator role.
- When a user is authenticated into Oracle Data Integrator, privileges granted to that user are determined by a union of all the Oracle Data Integrator roles to which the user is mapped.

  If the external user is also registered directly in Oracle Data Integrator, as in previous releases, the overall set of privileges given to the user consists of a union of:

  – The Oracle Data Integrator roles to which the user is mapped

  – Any privileges granted directly to the user
- Users do not need to be created and registered in Oracle Data Integrator in order to enable them to be granted access to Oracle Data Integrator.
- Any existing privileges and profiles assigned to individual users, as described in "Users" on page 25-3, remain in effect. This enables backward compatibility with previous releases of Oracle Data Integrator.

Defining and Managing Oracle Data Integrator Roles

At the time you create the master repository and configure it to use external authentication, the following occurs:

1. The Oracle Data Integrator role `SUPERVISOR_ROLE` is automatically created. This is the main administrative role in Oracle Data Integrator.
2. You are prompted to select one or more enterprise roles, or enterprise users, from the identity store to be mapped to this `SUPERVISOR_ROLE`. Note that Oracle Data Integrator can retrieve all available enterprise roles or enterprise users existing in the identity store that is configured in the `jps-config-jse.xml` file, as explained in "Set Up the OPSS Configuration File" on page 25-18.

Once you can be authenticated into Oracle Data Integrator and mapped to the `SUPERVISOR_ROLE` role, you can use the Roles Editor, available from Studio, to map additional enterprise principals to this role. Consequently, any user mapped to this role can then use the Roles Editor to create additional Oracle Data Integrator roles, assign privileges and profiles to those roles, and create other principal mappings from the identity store.

The following sections explain how to map enterprise principals to the SUPERVISOR_ ROLE role and how to use the Roles Editor to create, update, and remove Oracle Data Integrator roles:

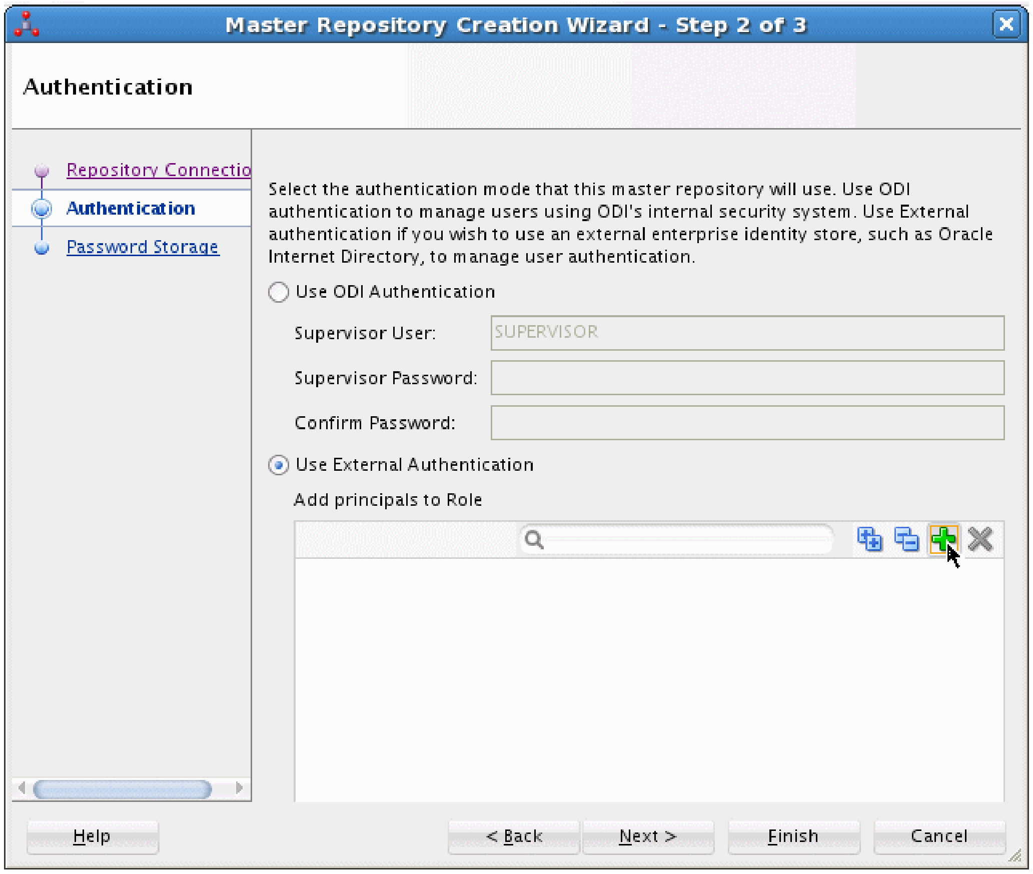
Configuring Enterprise Roles for the Supervisor Role

When you select **Use External Authentication** when creating the master repository, as explained in "Set External Authentication when Creating the Master Repository" on page 25-20, you need to configure enterprise roles. The initial enterprise role you configure is the principal (one or more) in the identity store to be mapped to the Oracle Data Integrator `SUPERVISOR_ROLE`.

To configure enterprise roles for the `SUPERVISOR_ROLE` in the master repository:

1. Start the Master Repository Creation Wizard, as explained in "Set External Authentication when Creating the Master Repository" on page 25-20.
2. In the **Master Repository Creation Wizard — Step 2 of 3** page, click the **Add** button retrieve the available enterprise roles and users from the external identity store, as shown in Figure 25–9:

Figure 25–9 Authentication Screen of the Master Repository Creation Wizard



Select the authentication mode that this master repository will use. Use ODI authentication to manage users using ODI's internal security system. Use External authentication if you wish to use an external enterprise identity store, such as Oracle Internet Directory, to manage user authentication.

Use ODI Authentication

Supervisor User: SUPERVISOR

Supervisor Password:

Confirm Password:

[✓ Use External Authentication]

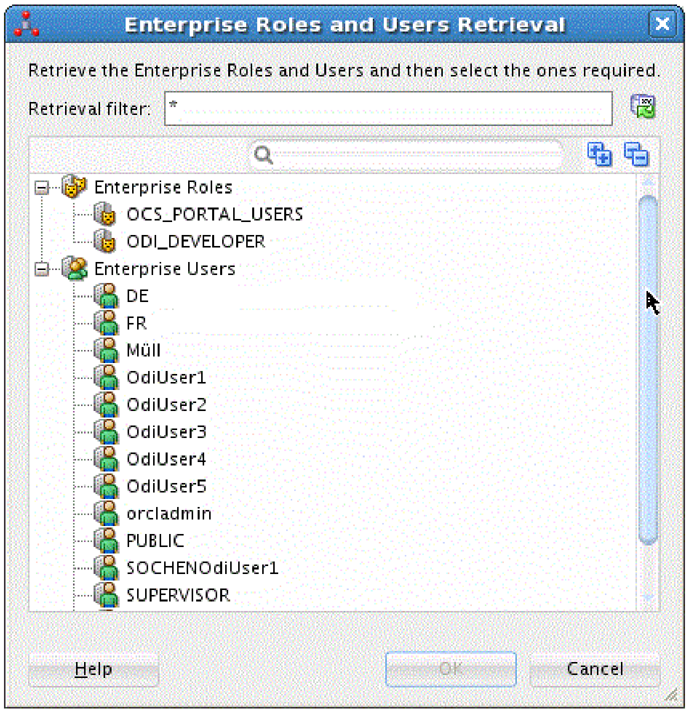
Add principals to Role

When you click the **Add** button, the Enterprise Roles and Users Retrieval dialog box is displayed, shown in Figure 25–10, in which you can either:

- Enter a filter expression to display the roles and users in the identity store that match the filter.
- Search the identity store for a specific role or user name.

You can expand the **Enterprise Roles** and **Enterprise Users** nodes to display the individual roles and users, respectively, that are defined in the identity store.

Figure 25–10 Enterprise Roles and Users Retrieval Dialog Box



Enterprise Roles and Users Retrieval

Retrieve the Enterprise Roles and Users and then select the ones required.

Retrieval filter: *

- Enterprise Roles
  - OCS_PORTAL_USERS
  - ODI_DEVELOPER
- Enterprise Users
  - DE
  - FR
  - Müll
  - OdiUser1
  - OdiUser2
  - OdiUser3
  - OdiUser4
  - OdiUser5
  - orcladmin
  - PUBLIC
  - SOCHENOdiUser1
  - SUPERVISOR

3. Select the enterprise user or role from the Enterprise Roles and Enterprise Users dialog box that you want to assign to the `SUPERVISOR_ROLE` role, and click **OK**.

4. Finish the creation of the master repository, as explained in "Creating the Master Repository" on page 3-4.

Using the Roles Editor

The following sections explain how to use the Roles Editor to manage enterprise role integration with Oracle Data Integrator:

- Start the Roles Editor
- Create a Role in Oracle Data Integrator
- Map Enterprise Principals to Oracle Data Integrator Roles
- Assign Privileges or Profiles to Oracle Data Integrator Roles
- Delete Oracle Data Integrator Roles

Start the Roles Editor You can use the Roles Editor to create, update, and delete Oracle Data Integrator roles.

To start the Roles Editor:

1. In Oracle Data Integrator Studio, click **Connect to Repository**, if necessary, and enter the required credentials to connect to the repository you want to use.

2. In Studio, choose the **Security Navigator** and select the **Roles** accordion.

3. Start the Roles Editor in either of the following ways:

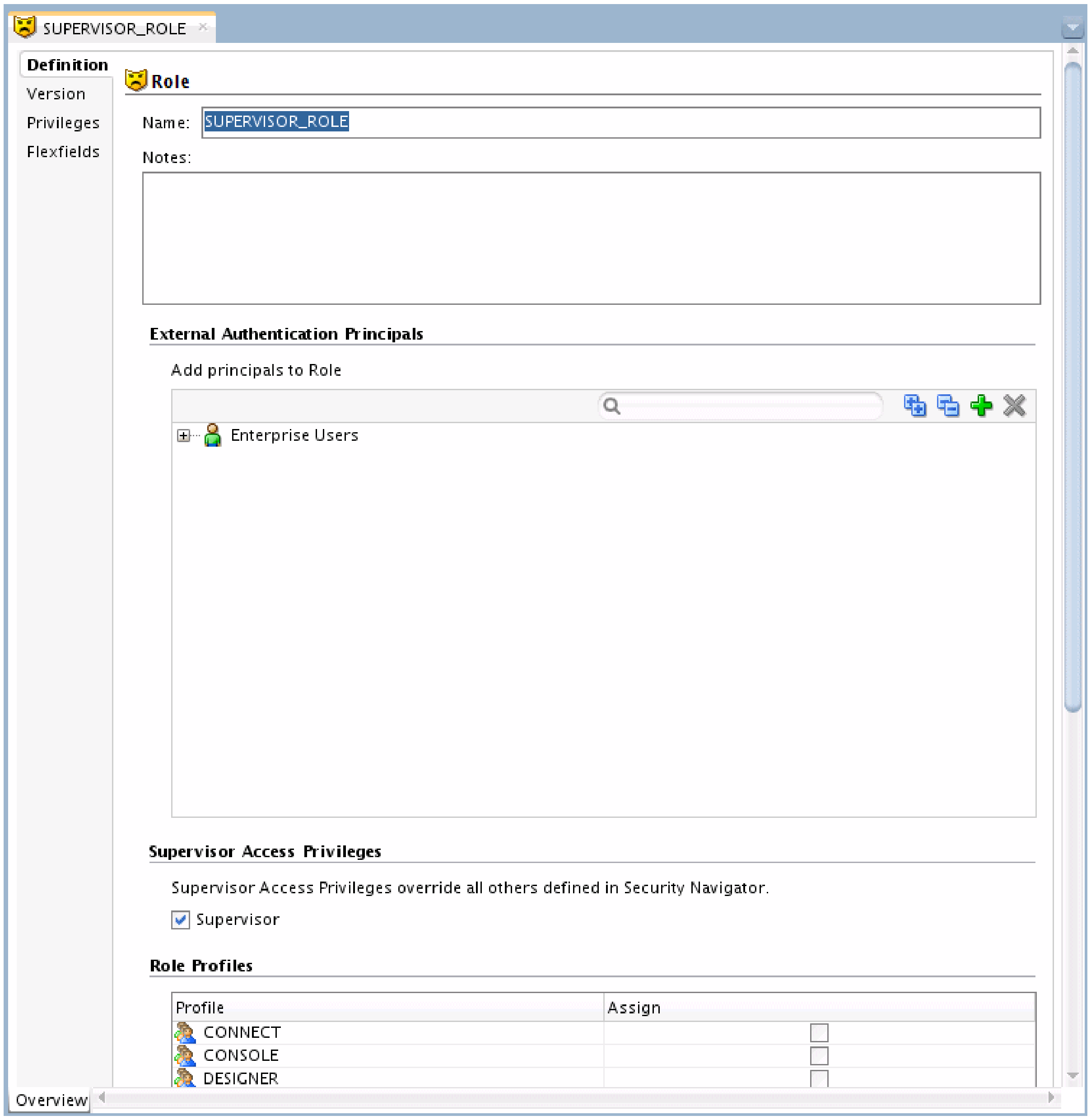
   - Click **New Role** in the toolbar of the Roles accordion. This enables you to create a new Oracle Data Integrator role.

- From the list of roles that is displayed in the Roles accordion, right-click a role name and select **Open**. This enables you to update an existing role.

4. Right-click and select **Open**.

The Roles Editor is shown in Figure 25–11.

Figure 25–11 Roles Editor



Name: SUPERVISOR_ROLE

Notes:

External Authentication Principals

Add principals to Role

Supervisor Access Privileges

Supervisor Access Privileges override all others defined in Security Navigator.

[✓] Supervisor

Role Profiles

Profile	Assign
CONNECT	☐
CONSOLE	☐
DESIGNER	☐

Create a Role in Oracle Data Integrator To create an Oracle Data Integrator role:

1. Start the Roles Editor in Studio, if necessary.
2. Choose the **Security Navigator** and select the **Roles** accordion.
3. Click **New Role** in the toolbar of the **Roles** accordion.

4. In the **Name** field, enter the name of the Oracle Data Integrator role you want to create.
5. From the **File** main menu, select **Save**.

The new role name is displayed in the **Roles** accordion.

Map Enterprise Principals to Oracle Data Integrator Roles To map enterprise principals to an Oracle Data Integrator role:

1. If the Oracle Integrator Role you want to modify is not already open in the Roles Editor, right-click the role name in the **Roles** accordion and select **Open**.
2. In the **Add Principals to Role** panel of the Roles Editor, click the **Add** button (plus sign) to display the Enterprise Roles and Users Retrieval dialog box.

   From this dialog box, you can either:

   - Enter a filter expression to display the roles and users in the identity store that match the filter.
   - Search the identity store for a specific role or user name.
3. Do either or both of the following:
   - To select one or more enterprise roles to map to the Oracle Data Integrator role, expand the Enterprise Roles node and select those roles.
   - To select one or more enterprise users to map to the Oracle Data Integrator role, expand the Enterprise Users node and select those users.
4. From the **File** main menu, select **Save**.

Assign Privileges or Profiles to Oracle Data Integrator Roles Studio supports two ways to assign privileges or profiles to an Oracle Data Integrator role:

- Using the Roles Editor
- Clicking and dragging privilege, profiles, and instance objects onto the role name in the Roles accordion

Using the Roles Editor:

1. If the Oracle Data Integrator role to which you want to assign privileges is not already open in the Roles Editor, right-click the role name in the **Roles** accordion and select **Open**.
2. If you want to assign supervisor privileges to the role, select **Supervisor** in the **Supervisor Access Privileges** section of the Roles Editor.
3. In the **Role Profiles** panel of the Roles Editor, select each profile you want to assign to the role.

   Note: If you have assigned supervisor privileges to the role, this step is unnecessary.

4. From the **File** main menu, select **Save**.

Clicking and Dragging:

Privileges on any of the following items can be added to a role by clicking it and dragging it onto a role name in the **Roles** accordion:

- An object node, or an entry within an object node, listed in the **Objects** accordion.

- A profile node, or an entry with a profile node, listed in the **Profiles** accordion.
- An object instance listed in an accordion in the Designer, Operator, or Topology Navigator. Select the instance and drag it onto the desired role name the same way as you grant an object instance privilege to an Oracle Data Integrator user (see "Granting an Authorization by Object Instance" on page 25-9).

After updating the privileges for an Oracle Data Integrator role, select **Save** from the **File** main menu.

Delete Oracle Data Integrator Roles To delete an Oracle Data Integrator role:

1. In the **Roles** accordion of the Security Navigator, select the role you want to delete.
2. Right-click and select **Delete**.
3. When prompted to confirm your choice, click **Yes**.

Enforcing Password Policies

The password policies consist of a set of rules applied to user passwords when using internal authentication. This set of rules is applied when the password is defined or modified by the user.

To define the password policy:

1. From the Security Navigator toolbar menu, select **Password Policy...**

   The Password Policy dialog appears. This dialog displays a list of rules.
2. If you want your password to expire automatically, check **Password are valid for (days)**, and set a number of days after which passwords need to be modified.
3. Click **Add a Policy**. The Policy Definition dialog appears. A policy is a set of conditions that are checked on passwords.
4. Set a **Name** and a **Description** for this new policy.
5. In the Rules section, add several conditions on the password text or length. You can define, for example, a minimum length for the passwords.
6. From the **Condition to match** list, select whether you want the password to meet at least one or all conditions.
7. Click **OK**.
8. Add as many policies as necessary, and select **Active** for those of the rules that you want to keep active. Only passwords that meet all the policies are considered as valid for the current policy.
9. Click **OK** to update the password policy.

Configuring Single Sign-On (SSO) for Oracle Data Integrator Console and Enterprise Manager using Oracle Access Manager 11*g*

This section describes how to optionally configure Single Sign-On for Oracle Data Integrator Console and Enterprise Manager with Oracle Access Manager (OAM) 11*g*.

To configure Single Sign-On for Oracle Data Integrator Console and Enterprise Manager:

1. Log in to the OAM Console using your browser:

   ```
   http://host:port/oamconsole
   ```

2. Go to **Policy Configuration** > **Application Domains**.

   The Policy Manager pane displays.

3. Locate the application domain you created using the name while registering the WebGate agent.

4. Expand the Resources node and click **Create**.

   The Resource page displays.

5. Add the resources that must be secured. For each resource:

   a. Select `http` as the **Resource Type**.

   b. Select the **Host Identifier** created while registering the WebGate agent.

   c. Enter the **Resource URL** as follows:

Resource URL	ODI Component
`/odiconsole*`	ODI Console
`/odiconsole/.../*`	ODI Console
`/em*`	Enterprise Manager
`/em/.../*`	Enterprise Manager

   d. Enter a **Description** for the resource and click **Apply**.

6. Go to **Authentication Policies** > **Protected Resource Policy** and add the newly created resources.

7. Do the same under **Authorization Policies** > **Protected Resource Policy**.

8. In your WebTier, modify the `mod_wl_ohs.conf` file (in `WT_ORACLE_HOME/instances/<your_instance>/config/OHS/ohs1/`) to include the ODI Console and Enterprise Manager, by adding two additional Location entries using the actual host ID for the WebCenter Portal Administration Server for WebLogicHost.

```
<Location /odiconsole*>
      SetHandler weblogic-handler
      WebLogicHost webcenter.example.com
      WebLogicPort 7001
</Location>
<Location /em*>
      SetHandler weblogic-handler
      WebLogicHost webcenter.example.com
      WebLogicPort 7001
</Location>
```

9. Restart the Oracle HTTP Server for your changes to take effect.

   You should now be able to access the ODI Console and Enterprise Manager with the following links:

   `http://host:OHS port/odiconsole`

   `http://host:OHS port/em`

   and be prompted with the OAM SSO login form.

For more information about installing and configuring OAM 11*g* for ODI, see "Integrating ODI with Oracle Access Manager 11*g*" in the *Installing and Configuring Oracle Data Integrator*.

Best Security Practices for Oracle Data Integrator

Table 25–3 provides a list of best practices for securing the Oracle Data Integrator environment.

Table 25–3 Best Security Practices in an Oracle Data Integrator Environment

Security Action	Description
Secure Oracle Data Integrator Studio	To provide optimal security for Oracle Data Integrator Studio, configure the following: ■ Password-protected wallet file — Create a password-protected wallet for storing the Oracle Data Integrator Studio login credentials. For more information, see "Using a Password-Protected Wallet for Storing Login Credentials" on page 25-10. ■ External authentication — Use external authentication mode instead of the default internal authentication mode. You can select external authentication mode as one of the options when creating the master repository. For more information, see "Configuring External Authentication" on page 25-17. ■ External password storage — For dataserver connection passwords, Oracle Data Integrator provides two options: internal password storage, and external password storage. Oracle recommends external password storage for secure production environments. You can choose the external password storage option at the time you create the master repository. For more information, see "Setting Up External Password Storage" on page 25-14.
Configure the Standalone Agent to use a secure transport	Clients communicate to the Standalone Agent to issue requests on executing Oracle Data Integrator jobs. By default, this communication is transmitted over HTTP. However, for production environments, Oracle recommends using a secure transport, such as HTTPS, for client communications with the Standalone Agent. You can use the WebLogic Server Administration Console to configure the secure transport used by the Standalone Agent. For example, in WebLogic Server you can: **1.** Configure SSL in the Managed Server instances hosting the Standalone Agent. For information, see "Configuring SSL" in *Administering Security for Oracle WebLogic Server*. **2.** Create a virtual host. **3.** Specify the Standalone Agent as an application to be served by the virtual host. **4.** Make sure that the listen ports used by the Standalone Agent are configured for SSL. For information about virtual hosting, see "Configuring Virtual Hosting" in *Administering Server Environments for Oracle WebLogic Server*.
Configure the Oracle Data Integrator Console to use a secure transport	Oracle recommends the use of a secure transport for access to the Oracle Data Integrator Console. For example, you can configure the Oracle Data Integrator Console to use port 7002, which is the default SSL listen port. Users would then access the Oracle Data Integrator Console using the following URL: `https://<host>:7002/odiconsole` You can use the WebLogic Server Administration Console to secure the transport used for the Oracle Data Integrator Console. For information, see "Configuring SSL" in *Administering Security for Oracle WebLogic Server*.

Part VIII

Appendices

Part VIII contains the following appendices:

A

Oracle Data Integrator Tools Reference

This appendix provides a reference of Oracle Data Integrator (ODI) tools. It is intended for application developers who want to use these tools to design integration scenarios.

This appendix includes the following sections:

- Using Oracle Data Integrator Tools
- Using Open Tools
- Developing Open Tools
- Oracle Data Integrator Tools by Category
- Alphabetical List of Oracle Data Integrator Tools

Using Oracle Data Integrator Tools

Oracle Data Integrator tools (also called Oracle Data Integrator commands) are commands provided for performing specific tasks at runtime. These tasks can be as simple as waiting for a certain time or producing a sound, or as sophisticated as executing Ant scripts or reading email from a server.

Oracle Data Integrator tools are used in Packages, Procedure Commands, Knowledge Modules Commands, or directly from a command line.

Note: Previous versions of Oracle Data Integrator supported calling built-in tools from Jython or Java scripts using their internal Java classes (such as `SnpsSendMail` and `SendMail`). This approach is no longer supported.

Note: Carriage returns in commands are not permitted.

Using a Tool in a Package

Adding and using an Oracle Data Integrator tool in a Package is described in "Adding Oracle Data Integrator Tool Steps" on page 10-6.

You can sequence the tool steps within the package and organize them according to their success and failure. For more information about sequencing, see "Defining the Sequence of Steps" on page 10-9 and "Arranging the Steps Layout" on page 10-8.

You can use variable values, sequences, or Oracle Data Integrator substitution method calls directly in tool parameters. See Chapter 13, "Creating and Using Procedures, Variables, Sequences, and User Functions," for more information.

Using a Tool in a Knowledge Module or Procedure Command

Using an Oracle Data Integrator tool in a Knowledge Module or Procedure is described in "Working with Procedures" on page 13-1.

You can use variable values, sequences, Oracle Data Integrator substitution method calls, or the results from a `SELECT` statement directly in tool parameters. See Chapter 13, "Creating and Using Procedures, Variables, Sequences, and User Functions," for more information.

Using a Tool From a Command Line

Command line scripts for Oracle Data Integrator tools are run from the *DOMAIN_HOME*`/bin` directory. To run a tool from a command line, you must first create an ODI Physical Agent instance in the ODI Topology and configure an ODI Standalone Agent instance in a Domain. For more information about performing these tasks, see *Installing and Configuring Oracle Data Integrator*.

When you run a tool from a command line, you must specify the `-INSTANCE` parameter, where `<agent_name>` is the name of the physical agent you configured (for example, `OracleDIAgent1`).

To use an Oracle Data Integrator tool from a command line:

1. Launch the command shell for your environment (Windows or UNIX).
2. Navigate to the *DOMAIN_HOME*`/bin` directory.
3. Launch the `startcmd.cmd` (Windows) or `startcmd.sh` (UNIX) command and run an Oracle Data Integrator tool with the following syntax:

   ```
   startcmd.<cmd|sh> -INSTANCE=<agent_name> <command_name> [<command_parameters>]*
   ```

Command names and command parameters are case-sensitive.

Important Notes

Note the following:

- On Windows platforms, command arguments that contain equal (=) signs or spaces must be surrounded with double quotation marks. This differs from the UNIX command call. For example:

  ```
  startcmd.cmd OdiSleep "-INSTANCE=OracleDIAgent1" "-DELAY=5000"
  ./startcmd.sh OdiSleep -INSTANCE=OracleDIAgent1 -DELAY=5000
  ```

- The following tools do not support direct invocation through a command line:
  - OdiRetrieveJournalData
  - OdiRefreshJournalCount

Using Open Tools

The Open Tools feature provides an extensible platform for developing custom third-party tools that you can use in Packages and Procedures. As with the standard tools delivered with Oracle Data Integrator, Open Tools can interact with the operating system and manipulate data.

Open Tools are written in Java. Writing your own Open Tools is described in "Developing Open Tools" on page A-4.

Open Tools are delivered as a Java package (`.zip` or `.jar`) that contains several files:

- A compiled Java `.class` file
- Other resources, such as icon files

Installing and Declaring an Open Tool

Before you can use an Open Tool, you must install and add it.

Installing an Open Tool

To install an Open Tool, you must add the Open Tool JAR into the classpath or the component using the tool.

Open Tool JARs must be added to the *DOMAIN_HOME*`/lib` directory. Drivers are added to the same location.

To deploy an Open Tool JAR with a Java EE agent, generate a server template for this agent. The Open Tool displays in the **Libraries and Drivers** list in the Template Generation Wizard. See "Creating a Server Template for the Java EE Agent" on page 4-15 for more information.

Note: This operation must be performed for each Oracle Data Integrator Studio from which the tool is being used, and for each agent that will run sessions using this tool.

Declaring a New Open Tool

This operation declares an Open Tool in a master repository and enables the tool to display in Oracle Data Integrator Studio.

To declare a new tool:

1. In Oracle Data Integrator Studio, select the **ODI** menu and then select **Add Remove/Open Tools**. The **Add Open Tools** dialog displays.
2. Enter the name of the class in the **Open Tool Class Name** field.

or:

1. Click **Find in the ClassPath**, then browse to the name of the Open Tool's Java class. To search for the class by name, enter part of the name in the field at the top.
2. Click **OK**.

   Note that all classes currently available to Oracle Data Integrator are displayed, including those that are not Open Tools. You must know the name of your class in order to add it.
3. Click **Add Open Tool**.
4. Select the line containing your Open Tool.
   - If the tool was correctly found on the classpath, the supplied icons, and the tool's syntax, description, provider, and version number are displayed.
   - If the tool was not found, an error message is displayed. Change the classpath, or move the Open Tool to the correct directory.

Note: This operation to declare a new Open Tool must be performed only once for a given master repository.

Note: An Open Tool name cannot start with `Snp` or `Odi`. An Open Tool with a name that starts with these strings is ignored.

Using Open Tools in a Package or Procedure

You can use Open Tools in a Package or Procedure, similar to the tools provided with Oracle Data Integrator.

Developing Open Tools

An Open Tool is a Java package that contains a compiled Java class that implements the interface `oracle.odi.sdk.opentools.IOpenTool`. For a complete description of classes and methods, see the *Oracle Data Integrator Open Tools Java API Reference* (JavaDoc).

An Open Tool package typically should also contain two icons, which are used to represent the Open Tool in the Oracle Data Integrator graphical interface.

Classes

The following table lists and describes Open Tool classes and interfaces.

Class or Interface	Description
`IOpenTool`	Interface that every Open Tool must implement.
`OpenToolAbstract`	Abstraction of the interface with some helper methods. Preferably extend this class rather than implementing the interface directly.
`IOpenToolParameter`	Interface that parameters used by Open Tools must implement. In most cases, `OpenToolParameter` should be used rather than implementing this interface.
`OpenToolParameter`	Complete implementation of `IOpenToolParameter`. Each `OpenToolParameter` holds one parameter.
`OpenToolsExecutionException`	Exception class that should be thrown if necessary by Open Tool methods.
`SimpleOpenToolExample`	A simple example of an Open Tool, which can be used as a starting point.

Developing a New Open Tool

The following steps describe the development of a basic Open Tool, SimpleMessageBox. The source code for this class is available in the `demo/plugins/src` directory.

1. Define the syntax. In this example, the Open Tool is called as follows:

   ```
   SimpleMessageBox "-TEXT=<text message>" "-TITLE=<window title>"
   ```

2. Create 16x16 and 32x32 icons (usually in `.gif` format).
3. Create and implement the class. See "Implementing the Class" on page A-5.

4. Compile the class and create a package with the two icon files.
5. Install and declare the Open Tool as described in "Installing and Declaring an Open Tool" on page A-3.

Implementing the Class

Implementing the class consists of the following steps:

1. Declaration
2. Importing Packages
3. Defining the Parameters
4. Implementing Informational Functions
5. Execution

Declaration Before you declare the class, you must name the package.

Naming the Package

Put the class in a package named appropriately. The package name is used to identify the Open Tool when installing it.

```
package com.myCompany.OpenTools;
```

Declaring the Class

There are two basic approaches to developing an Open Tool:

- Extend an existing class that you want to convert into an Open Tool. In this case, simply implement the interface `IOpenTool` directly on the existing class.
- Develop a new class. In this case, it is easiest to extend the abstract class `OpenToolAbstract`. This abstract class also contains additional helper methods for working with parameters.

  ```
  public class SimpleMessageBox extends OpenToolAbstract {
  ```

Importing Packages Almost every Open Tool must import the following Open Tool SDK packages:

```
import oracle.odi.sdk.opentools.IOpenTool; /* All Open Tool classes need these
three classes */

import oracle.odi.sdk.opentools.IOpenToolParameter;

import oracle.odi.sdk.opentools.OpenToolExecutionException;

import oracle.odi.sdk.opentools.OpenToolAbstract; /* The abstract extended for the
Open Tool */

import oracle.odi.sdk.opentools.OpenToolParameter; /* The class used for
parameters */
```

In this particular example, a package to create the message box is also needed:

```
import javax.swing.JOptionPane; /* Needed for the message box used in this example
*/
```

Defining the Parameters Add a property to store the `OpenToolParameter` objects. This is used to both define them for the syntax, and to retrieve the values of the parameters from the eventual user. It is easiest to define the parameters of the Open Tool with a static array as follows. This array should be private, as it will be accessed through an accessor function.

```
private static final IOpenToolParameter[] mParameters = new IOpenToolParameter[]
{
    new OpenToolParameter("-TEXT", "Message text", "Text to show in the messagebox
(Mandatory).", true),
    new OpenToolParameter("-TITLE", "Messagebox title", "Title of the
messagebox.", false)
};
```

The four parameters passed to the `OpenToolParameter()` constructor are as follows:

1. The code of the parameter, including the initial hyphen. This code must correspond to the syntax returned by `getSyntax()`.
2. The user-friendly name, which is used if the user is using the graphical interface to set parameters.
3. A descriptive help text.
4. Whether the parameter is mandatory. This is an indication to the user.

> **Note:** Oracle Data Integrator does not enforce the mandatory flag on parameters. Your class must be able to handle any combination of parameters being provided.

You must implement the accessor function `getParameters()` to retrieve the parameters:

```
public IOpenToolParameter[] getParameters()
{
      return mParameters;
}
```

Implementing Informational Functions Implement functions to return information about your Open Tool: `getDescription()`, `getVersion()`, `getProvider()`.

```
public String getDescription() { return "This Open Tool displays a message box
when executed."; }
public String getVersion() { return "v1.0"; }
public String getProvider() { return "My Company, Inc."; }
```

The `getSyntax()` function determines the name of the Open Tool as it is displayed in the Oracle Data Integrator graphical interface, and also the initial values of the parameter. Make sure the names of the parameters here match the names of the parameters returned by `getParameters()`.

```
public String getSyntax()
{
        return "SimpleMessageBox \"-TEXT=<text message>\" \"-TITLE=<window
title>\"";
}
```

The `getIcon()` method should then return paths to two appropriately sized images. It should look something like this:

```
public String getIcon(int pIconType)
{
        switch (pIconType)
        {
               case IOpenTool.SMALL_ICON:
               return "/com/myCompany/OpenTools/images/SimpleMessageBox_16.gif";
             case IOpenTool.BIG_ICON:
             return "/com/myCompany/OpenTools/images/SimpleMessageBox_32.gif";
             default:
             return "";
     }
}
```

Execution Finally, the `execute()` method, which carries out the functionality provided by the Open Tool. In this case, a message box is shown. If you are extending the `OpenToolAbstract` class, use the `getParameterValue()` method to easily retrieve the values of parameters, as they are set at runtime.

> **Note:** You must catch all exceptions and only raise an `OpenToolExecutionException`.

```
public void execute() throws OpenToolExecutionException
{
     try
     {
     if (getParameterValue("-TITLE") == null ||
getParameterValue("-TITLE").equals("")) /* title was not filled in by user */
     {
             JOptionPane.showMessageDialog(null, (String)
getParameterValue("-TEXT"), (String) "Message", JOptionPane.INFORMATION_MESSAGE);
          } else
          {
             JOptionPane.showMessageDialog(null, (String)
getParameterValue("-TEXT"),
                     (String) getParameterValue("-TITLE"),
                     JOptionPane.INFORMATION_MESSAGE);
          }
     }
     /* Traps any exception and throw them as OpenToolExecutionException */
     catch (IllegalArgumentException e)
     {
          throw new OpenToolExecutionException(e);
     }
}
```

Open Tools at Runtime

In general, your Open Tool class is instantiated only very briefly, and is used in the following ways.

Installation

When the user chooses to install an Open Tool, Oracle Data Integrator instantiates the class and calls the methods `getDescription()`, `getProvider()`, `getIcon()`, and `getVersion()` to retrieve information about the class.

Use in a Package

When the Open Tool is used in a package, the class is instantiated briefly to call the methods `getDescription()`, `getProvider()`, `getIcon()`, and `getVersion()`. Additionally, `getSyntax()` is called to retrieve the code name of the Open Tool and its default arguments. The method `getParameters()` is called to display the list of arguments to the user.

Execution

Each time the Open Tool is executed in a package or procedure, the class is instantiated again; it has no persistence after its execution. The `execute()` method is called just once.

Tip: See also "Using Open Tools" on page A-2 and Open Tools SDK documentation (JavaDoc).

Oracle Data Integrator Tools by Category

This section lists Oracle Data Integrator tools by category.

Metadata

- OdiReverseGetMetaData
- OdiReverseManageShortcut
- OdiReverseResetTable
- OdiReverseSetMetaData

Oracle Data Integrator Objects

- OdiDeleteScen
- OdiExportAllScen
- OdiExportEnvironmentInformation
- OdiExportLog
- OdiExportMaster
- OdiExportObject
- OdiExportScen
- OdiExportWork
- OdiGenerateAllScen
- OdiImportObject
- OdiImportScen

Utilities

- OdiAnt
- OdiBeep
- OdiEnterpriseDataQuality
- OdiKillAgent

- OdiOSCommand
- OdiPingAgent
- OdiPurgeLog
- OdiReinitializeSeq
- OdiRemoveTemporaryObjects
- OdiStartLoadPlan
- OdiStartOwbJob
- OdiStartScen
- OdiUpdateAgentSchedule

Internet Related Tasks

- OdiFtp
- OdiFtpGet
- OdiFtpPut
- OdiInvokeWebService
- OdiReadMail
- OdiScpGet
- OdiScpPut
- OdiSftp
- OdiSftpGet
- OdiSftpPut
- OdiSendMail

Files

- OdiFileAppend
- OdiFileCopy
- OdiFileDelete
- OdiFileMove
- OdiFileWait
- OdiMkDir
- OdiOutFile
- OdiSqlUnload
- OdiUnZip
- OdiZip

SAP

- OdiSAPALEClient and OdiSAPALEClient3
- OdiSAPALEServer and OdiSAPALEServer3

XML

- OdiXMLConcat
- OdiXMLSplit

Event Detection

- OdiFileWait
- OdiReadMail
- OdiSleep
- OdiWaitForChildSession
- OdiWaitForData
- OdiWaitForLoadPlans
- OdiWaitForLogData
- OdiWaitForTable

Changed Data Capture

- OdiManageOggProcess
- OdiRefreshJournalCount
- OdiRetrieveJournalData
- OdiWaitForData
- OdiWaitForLogData
- OdiWaitForTable

Alphabetical List of Oracle Data Integrator Tools

This section lists Oracle Data Integrator tools in alphabetical order.

- OdiAnt
- OdiBeep
- OdiDeleteScen
- OdiEnterpriseDataQuality
- OdiExportAllScen
- OdiExportEnvironmentInformation
- OdiExportLog
- OdiExportMaster
- OdiExportObject
- OdiExportScen
- OdiExportWork
- OdiFileAppend
- OdiFileCopy
- OdiFileDelete

- OdiFileMove
- OdiFileWait
- OdiFtp
- OdiFtpGet
- OdiFtpPut
- OdiGenerateAllScen
- OdiImportObject
- OdiImportScen
- OdiInvokeWebService
- OdiKillAgent
- OdiManageOggProcess
- OdiMkDir
- OdiOSCommand
- OdiOutFile
- OdiPingAgent
- OdiPurgeLog
- OdiReadMail
- OdiRefreshJournalCount
- OdiReinitializeSeq
- OdiRemoveTemporaryObjects
- OdiRetrieveJournalData
- OdiReverseGetMetaData
- OdiReverseManageShortcut
- OdiReverseResetTable
- OdiReverseSetMetaData
- OdiSAPALEClient and OdiSAPALEClient3
- OdiSAPALEServer and OdiSAPALEServer3
- OdiScpGet
- OdiScpPut
- OdiSendMail
- OdiSftp
- OdiSftpGet
- OdiSftpPut
- OdiSleep
- OdiSqlUnload
- OdiStartLoadPlan
- OdiStartOwbJob

- OdiStartScen
- OdiUnZip
- OdiUpdateAgentSchedule
- OdiWaitForChildSession
- OdiWaitForData
- OdiWaitForLoadPlans
- OdiWaitForLogData
- OdiWaitForTable
- OdiXMLConcat
- OdiXMLSplit
- OdiZip

OdiAnt

Use this command to execute an Ant buildfile.

For more details and examples of Ant buildfiles, refer to the online documentation: `http://jakarta.apache.org/ant/manual/index.html`

Usage

```
OdiAnt -BUILDFILE=<file> -LOGFILE=<file> [-TARGET=<target>]
[-D<property name>=<property value>]* [-PROJECTHELP] [-HELP]
[-VERSION] [-QUIET] [-VERBOSE] [-DEBUG] [-EMACS]
[-LOGGER=<classname>] [-LISTENER=<classname>] [-FIND=<file>]
```

Parameters

Parameters	Mandatory	Description
`-BUILDFILE=<file>`	Yes	Ant buildfile. XML file containing the Ant commands.
`-LOGFILE=<file>`	Yes	Use given file for logging.
`-TARGET=<target>`	No	Target of the build process.
`-D<property name>=<property value>`	No	List of properties with their values.
`-PROJECTHELP`	No	Displays the help on the project.
`-HELP`	No	Displays Ant help.
`-VERSION`	No	Displays Ant version.
`-QUIET`	No	Run in nonverbose mode.
`-VERBOSE`	No	Run in verbose mode.
`-DEBUG`	No	Prints debug information.
`-EMACS`	No	Displays the logging information without adornments.
`-LOGGER=<classname>`	No	Java class performing the logging.
`-LISTENER=<classname>`	No	Adds a class instance as a listener.

Parameters	Mandatory	Description
`-FIND=<file>`	No	Looks for the Ant buildfile from the root of the file system and uses it.

Examples

Download the `*.html` files from the directory `/download/public` using FTP from `ftp.mycompany.com` to the directory `C:\temp`.

Step 1: Generate the Ant buildfile:

```
OdiOutFile -FILE=c:\temp\ant_cmd.xml
<?xml version="1.0"?>
<project name="myproject" default="ftp" basedir="/">
     <target name="ftp">
          <ftp action="get" remotedir="/download/public"
          server="ftp.mycompany.com" userid="anonymous"
          password="me@mycompany.com">
                <fileset dir="c:\temp">
           <include name="**/*.html"/>
                </fileset>
     </ftp>
   </target>
</project>
```

Step 2: Run the Ant buildfile:

```
OdiAnt -BUILDFILE=c:\temp\ant_cmd.xml -LOGFILE=c:\temp\ant_cmd.log
```

OdiBeep

Use this command to play a default beep or sound file on the machine hosting the agent.

The following file formats are supported by default:

- WAV
- AIF
- AU

Note: To play other file formats, you must add the appropriate JavaSound Service Provider Interface (JavaSound SPI) to the application classpath.

Usage

```
OdiBeep [-FILE=<sound_file>]
```

Parameters

Parameters	Mandatory	Description
`-FILE`	No	Path and file name of sound file to play. If not specified, the default beep sound for the machine is used.

Examples

Play the sound file `c:\wav\alert.wav`.

```
OdiBeep -FILE=c:\wav\alert.wav
```

OdiDeleteScen

Use this command to delete a given scenario version.

Usage

```
OdiDeleteScen -SCEN_NAME=<name> -SCEN_VERSION=<version>
```

Parameters

Parameters	Mandatory	Description
`-SCEN_NAME=<name>`	Yes	Name of the scenario to delete.
`-SCEN_VERSION=<version>`	Yes	Version of the scenario to delete.

Examples

Delete the `DWH` scenario in version `001`.

```
OdiDeleteScen -SCEN_NAME=DWH -SCEN_VERSION=001
```

OdiEnterpriseDataQuality

Use this command to invoke an Oracle Enterprise Data Quality (Datanomic) job.

Note: The OdiEnterpriseDataQuality tool supports Oracle Enterprise Data Quality version 8.1.6 and later.

Usage

```
OdiEnterpriseDataQuality "-JOB_NAME=<EDQ job name>"
"-PROJECT_NAME=<EDQ project name>" "-HOST=<EDQ server host name>"
"-PORT=<EDQ server JMX port>" "-USER=<EDQ user>"
"-PASSWORD=<EDQ user's password>" "-SYNCHRONOUS=<yes|no>" "-DOMAIN=-<EDQ_DOMAIN>"
```

Parameters

Parameters	Mandatory	Description
`-JOB_NAME=<EDQ job name>`	Yes	Name of the Enterprise Data Quality job.
`-PROJECT_NAME=<EDQ project name>`	Yes	Name of the Enterprise Data Quality project.
`-HOST=<EDQ server host name>`	Yes	Host name of the Enterprise Data Quality server. Example: `localhost`
`-PORT=<EDQ server JMX port>`	Yes	JMX port of the Enterprise Data Quality server. Example: `9005`
`-USER=<EDQ user>`	Yes	User name of the Enterprise Data Quality server user. Example: `dnadmin`

Parameters	Mandatory	Description
`-PASSWORD=<EDQ user's password>`	Yes	Password of the Enterprise Data Quality user.
`-SYNCHRONOUS=<yes|no>`	No	If set to Yes (default), the tool waits for the quality process to complete before returning, with possible error code. If set to No, the tool ends immediately with success and does not wait for the quality process to complete.
`-DOMAIN=<EDQ_DOMAIN>`	No	Name of the MBean domain. The default value is `dndirector`.

Examples

Execute the Enterprise Data Quality job `CLEANSE_CUSTOMERS` located in the project `CUSTOMERS`.

```
EnterpriseDataQuality "-JOB_NAME=CLEANSE_CUSTOMERS" "-PROJECT_NAME=CUSTOMERS"
"-HOST=machine.oracle.com" "-PORT=9005" "-USER=odi" "-PASSWORD=odi"
"DOMAIN=dndirector"
```

OdiExportAllScen

Use this command to export a group of scenarios from the connected repository.

The export files are named `SCEN_<scenario name><scenario version>.xml`. This command reproduces the behavior of the export feature available in Designer Navigator and Operator Navigator.

Usage

```
OdiExportAllScen -TODIR=<directory> [-FORCE_OVERWRITE=<yes|no>]
[-FROM_PROJECT=<project_id>] [-FROM_FOLDER=<folder_id>]
[-FROM_PACKAGE=<package_id>] [-RECURSIVE_EXPORT=<yes|no>]
[-XML_VERSION=<1.0>] [-XML_CHARSET=<charset>]
[-JAVA_CHARSET=<charset>] [-EXPORT_MAPPING=<yes|no>]
[-EXPORT_PACK=<yes|no>] [-EXPORT_POP=<yes|no>]
[-EXPORT_TRT=<yes|no>] [-EXPORT_VAR=<yes|no>]
```

Parameters

Parameters	Mandatory	Description
`-TODIR=<directory>`	Yes	Directory into which the export files are created.
`-FORCE_OVERWRITE=<yes|no>`	No	If set to Yes, existing export files are overwritten without warning. The default value is No.
`-FROM_PROJECT=<project_id>`	No	ID of the project containing the scenarios to export. This value is the **Global ID** that displays in the **Version** tab of the project window in Studio. If this parameter is not set, scenarios from all projects are taken into account for the export.
`-FROM_FOLDER=<folder_id>`	No	ID of the folder containing the scenarios to export. This value is the **Global ID** that displays in the **Version** tab of the folder window in Studio. If this parameter is not set, scenarios from all folders are taken into account for the export.

Parameters	Mandatory	Description
`-FROM_PACKAGE=<package_id>`	No	ID of the source package of the scenarios to export. This value is the **Global ID** that displays in the **Version** tab of the package window in Studio. If this parameter is not set, scenarios from all components are taken into account for the export.
`-RECURSIVE_EXPORT=<yes|no>`	No	If set to Yes (default), all child objects (schedules) are exported with the scenarios.
`-XML_VERSION=<1.0>`	No	Sets the XML version shown in the XML header. The default value is `1.0`.
`-XML_CHARSET=<charset>`	No	Encoding specified in the XML export file in the tag `<?xml version="1.0" encoding="ISO-8859-1"?>`. The default value is `ISO-8859-1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
`-JAVA_CHARSET=<charset>`	No	Target file encoding. The default value is `ISO8859_1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
`-EXPORT_MAPPING=<yes|no>`	No	Indicates if the mapping scenarios should be exported. The default value is No.
`-EXPORT_PACK=<yes|no>`	No	Indicates if the scenarios attached to packages should be exported. The default value is Yes.
`-EXPORT_POP=<yes|no>`	No	Indicates if the scenarios attached to mappings should be exported. The default value is No.
`-EXPORT_TRT=<yes|no>`	No	Indicates if the scenarios attached to procedures should be exported. The default value is No.
`-EXPORT_VAR=<yes|no>`	No	Indicates if the scenarios attached to variables should be exported. The default value is No.

Examples

Export all scenarios from the `DW01` project of **Global ID** `2edb524d-eb17-42ea-8aff-399ea9b13bf3` into the `/temp/` directory, with all dependent objects.

```
OdiExportAllScen -FROM_PROJECT=2edb524d-eb17-42ea-8aff-399ea9b13bf3 -TODIR=/temp/
-RECURSIVE_EXPORT=yes
```

OdiExportEnvironmentInformation

Use this command to export the details of the technical environment into a comma separated (`.csv`) file into the directory of your choice. This information is required for maintenance or support purposes.

Usage

```
OdiExportEnvironmentInformation -TODIR=<toDir> -FILE_NAME=<FileName>
[-CHARSET=<charset>] [-SNP_INFO_REC_CODE=<row_code>]
[-MASTER_REC_CODE=<row_code>] [-WORK_REC_CODE=<row_code>]
[-AGENT_REC_CODE=<row_code>] [-TECHNO_REC_CODE=<row_code>]
```

```
[-RECORD_SEPARATOR_HEXA=<rec_sep>]
[-FIELD_SEPARATOR_HEXA=<field_sep] [-TEXT_SEPARATOR=<text_sep>]
```

Parameters

Parameters	Mandatory	Description
`-TODIR=<toDir>`	Yes	Target directory for the export.
`-FILE_NAME=<FileName>`	Yes	Name of the CSV export file. The default value is `snps_tech_inf.csv`.
`-CHARSET=<charset>`	No	Character set of the export file.
`-SNP_INFO_REC_CODE=<row_code>`	No	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. The default value is `SUNOPSIS`.
`-MASTER_REC_CODE=<row_code>`	No	Code for rows containing information about the master repository. The default value is `MASTER`.
`-WORK_REC_CODE=<row_code>`	No	Code for rows containing information about the work repository. The default value is `WORK`.
`-AGENT_REC_CODE=<row_code>`	No	Code for rows containing information about the various agents that are running. The default value is `AGENT`.
`-TECHNO_REC_CODE=<row_code>`	No	Code for rows containing information about the data servers, their versions, and so on. The default value is `TECHNO`.
`-RECORD_SEPARATOR_HEXA=<rec_sep>`	No	One or several characters in hexadecimal code separating lines (or records) in the file. The default value is `O0D0A`.
`-FIELD_SEPARATOR_HEXA=<field_sep>`	No	One or several characters in hexadecimal code separating the fields in a record. The default value is `2C`.
`-TEXT_SEPARATOR=<text_sep>`	No	Character in hexadecimal code delimiting a `STRING` field. The default value is `22`.

Examples

Export the details of the technical environment into the `/temp/snps_tech_inf.csv` export file.

```
OdiExportEnvironmentInformation "-TODIR=/temp/"
"-FILE_NAME=snps_tech_inf.csv" "-CHARSET=ISO8859_1"
"-SNP_INFO_REC_CODE=SUNOPSIS" "-MASTER_REC_CODE=MASTER"
"-WORK_REC_CODE=WORK" "-AGENT_REC_CODE=AGENT"
"-TECHNO_REC_CODE=TECHNO" "-RECORD_SEPARATOR_HEXA=0D0A"
"-FIELD_SEPARATOR_HEXA=2C" "-TEXT_SEPARATOR_HEXA=22"
```

OdiExportLog

Use this command to export the execution log into a ZIP export file.

Usage

```
OdiExportLog -TODIR=<toDir> [-EXPORT_TYPE=<logsToExport>]
[-ZIPFILE_NAME=<zipFileName>] [-XML_CHARSET=<charset>]
[-JAVA_CHARSET=<charset>] [-FROMDATE=<from_date>] [-TODATE=<to_date>]
[-AGENT=<agent>] [-CONTEXT=<context>] [-STATUS=<status>]
[-USER_FILTER=<user>] [-NAME=<sessionOrLoadPlanName>]
```

Parameters

Parameters	Mandatory	Description
`-EXPORT_TYPE=<logsToExport>`	No	Export the log of: ■ `LOAD_PLAN_RUN`: All Load Plan runs that match the export criteria are exported, including all sessions launched by the Load Plan runs along the child session's hierarchy. ■ `SESSION`: All session logs that match the export filter criteria are exported. All Load Plan sessions will be excluded when exporting the session logs. ■ `ALL`: All Load Plan runs and session logs that match the filter criteria are exported.
`-TODIR=<toDir>`	Yes	Target directory for the export.
`-ZIPFILE_NAME=<zipFileName>`	No	Name of the compressed file.
`-XML_CHARSET=<charset>`	No	XML version specified in the export file. Parameter `xml version` in the XML file header. `<?xml version="1.0" encoding="ISO-8859-1"?>`. The default value is `ISO-8859-1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
`-JAVA_CHARSET=<charset>`	No	Result file Java character encoding. The default value is `ISO8859_1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
`-FROMDATE=<from_date>`	No	Beginning date for the export, using the format yyyy/MM/dd hh:mm:ss. All sessions from this date are exported.
`-TODATE=<to_date>`	No	End date for the export, using the format yyyy/MM/dd hh:mm:ss. All sessions to this date are exported.
`-AGENT=<agent>`	No	Exports only sessions executed by the agent `<agent>`.
`-CONTEXT=<context>`	No	Exports only sessions executed in the context code `<context>`.
`-STATUS=<status>`	No	Exports only sessions in the specified state. Possible states are Done, Error, Queued, Running, Waiting, and Warning.
`-USER_FILTER=<user>`	No	Exports only sessions launched by `<user>`.
`-NAME=<sessionOrLoadPlanName>`	No	Name of the session or Load Plan to be exported.

Examples

Export and compress the log into the `/temp/log2.zip` export file.

```
OdiExportLog "-EXPORT_TYPE=ALL" "-TODIR=/temp/" "-ZIPFILE_NAME=log2.zip"
"-XML_CHARSET=ISO-8859-1" "-JAVA_CHARSET=ISO8859_1"
```

OdiExportMaster

Use this command to export the master repository to a directory or ZIP file. The versions and/or solutions stored in the master repository are optionally exported.

Usage

```
OdiExportMaster -TODIR=<toDir> [-ZIPFILE_NAME=<zipFileName>]
[-EXPORT_SOLUTIONS=<yes|no>] [-EXPORT_VERSIONS=<yes|no>]
[-XML_CHARSET=<charset>] [-JAVA_CHARSET=<charset>]
```

Parameters

Parameters	Mandatory	Description
`-TODIR=<toDir>`	Yes	Target directory for the export.
`-ZIPFILE_NAME=<zipFileName>`	No	Name of the compressed file.
`-EXPORT_SOLUTIONS=<yes|no>`	No	Exports all solutions that are stored in the repository. The default value is No.
`-EXPORT_VERSIONS=<yes|no>`	No	Exports all versions of objects that are stored in the repository. The default value is No.
`-XML_CHARSET=<charset>`	No	XML version specified in the export file. Parameter `xml version` in the XML file header. `<?xml version="1.0" encoding="ISO-8859-1"?>`. The default value is `ISO-8859-1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
`-JAVA_CHARSET=<charset>`	No	Result file Java character encoding. The default value is `ISO8859_1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html

Examples

Export and compress the master repository into the `export.zip` file located in the `/temp/` directory.

```
OdiExportMaster "-TODIR=/temp/" "-ZIPFILE_NAME=export.zip"
"-XML_CHARSET=ISO-8859-1" "-JAVA_CHARSET=ISO8859_1"
"-EXPORT_VERSIONS=YES"
```

OdiExportObject

Use this command to export an object from the current repository. This command reproduces the behavior of the export feature available in the user interface.

Usage

```
OdiExportObject -CLASS_NAME=<class_name> -I_OBJECT=<object_id>
[-EXPORT_DIR=<directory>] [-EXPORT_NAME=<export_name>|-FILE_NAME=<file_name>]
[-FORCE_OVERWRITE=<yes|no>] [-RECURSIVE_EXPORT=<yes|no>] [-XML_VERSION=<1.0>]
[-XML_CHARSET=<charset>] [-JAVA_CHARSET=<charset>]
```

Parameters

Parameters	Mandatory	Description
`-CLASS_NAME=<class_name>`	Yes	Class of the object to export (see the following list of classes).
`-I_OBJECT=<object_id>`	Yes	Object identifier. This value is the **Global ID** that displays in the **Version** tab of the object edit window.
`-FILE_NAME=<file_name>`	No	Export file name. Absolute path or relative path from `EXPORT_DIR`. This file name may or may not comply with the Oracle Data Integrator standard export file prefix and suffix. To comply with these standards, use the `-EXPORT_NAME` parameter instead. This parameter cannot be used if `-EXPORT_NAME` is set.
`-EXPORT_DIR=<directory>`	No	Directory where the object will be exported. The export file created in this directory is named based on the `-FILE_NAME` and `-EXPORT_NAME` parameters. If `-FILE_NAME` or `-EXPORT_NAME` are not specified, the export file is automatically named `<object_prefix>_<object_name>.xml`. For example, a project named `Datawarehouse` would be exported to `PRJ_Datawarehouse.xml`.
`-EXPORT_NAME=<export_name>`	No	Export name. Use this parameter to generate an export file named `<object_prefix>_<export_name>.xml`. This parameter cannot be used with `-FILE_NAME`.
`-FORCE_OVERWRITE=<yes|no>`	No	If set to Yes, an existing export file with the same name is forcibly overwritten. The default value is No.
`-RECURSIVE_EXPORT=<yes|no>`	No	If set to Yes (default), all child objects are exported with the current object. For example, if exporting a project, all folders, KMs, and so on in this project are exported into the project export file.
`-XML_VERSION=<1.0>`	No	Sets the XML version that appears in the XML header. The default value is `1.0`.
`-XML_CHARSET=<charset>`	No	Encoding specified in the XML file, in the tag `<?xml version="1.0" encoding="ISO-8859-1"?>`. The default value is `ISO-8859-1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
`-JAVA_CHARSET=<charset>`	No	Target file encoding. The default value is `ISO8859_1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html

List of Classes

Object	Class Name
Column	SnpCol
Condition/Filter	SnpCond
Context	SnpContext
Data Server	SnpConnect
Datastore	SnpTable
Folder	SnpFolder
Interface	SnpPop
Language	SnpLang
Loadplan	SnpLoadPlan
Mapping	SnpMapping
Model	SnpModel
Package	SnpPackage
Physical Schema	SnpPschema
Procedure or KM	SnpTrt
Procedure or KM Option	SnpUserExit
Project	SnpProject
Reference	SnpJoin
Reusable Mapping	SnpMapping
Scenario	SnpScen
Sequence	SnpSequence
Step	SnpStep
Sub-Model	SnpSubModel
Technology	SnpTechno
User Functions	SnpUfunc
Variable	SnpVar
Version of an Object	SnpVer

Examples

Export the `DW01` project of **Global ID** `2edb524d-eb17-42ea-8aff-399ea9b13bf3` into the `/temp/dw1.xml` export file, with all dependent objects.

```
OdiExportObject -CLASS_NAME=SnpProject
-I_OBJECT=2edb524d-eb17-42ea-8aff-399ea9b13bf3
-FILE_NAME=/temp/dw1.xml -FORCE_OVERWRITE=yes
-RECURSIVE_EXPORT=yes
```

OdiExportScen

Use this command to export a scenario from the current work repository.

Usage

```
OdiExportScen -SCEN_NAME=<scenario_name> -SCEN_VERSION=<scenario_version>
[-EXPORT_DIR=<directory>] [-FILE_NAME=<file_name>|EXPORT_NAME=<export_name>]
[-FORCE_OVERWRITE=<yes|no>] [-RECURSIVE_EXPORT=<yes|no>] [-XML_VERSION=<1.0>]
[-XML_CHARSET=<encoding>] [-JAVA_CHARSET=<encoding>]
```

Parameters

Parameters	Mandatory	Description
`-SCEN_NAME=<scenario_name>`	Yes	Name of the scenario to be exported.
`-SCEN_VERSION=<scenario_version>`	Yes	Version of the scenario to be exported.
`-FILE_NAME=<file_name>`	Yes	Export file name. Absolute path or relative path from `-EXPORT_DIR`. This file name may or not comply with the Oracle Data Integrator standard export file prefix and suffix for scenarios. To comply with these standards, use the `-EXPORT_NAME` parameter instead. This parameter cannot be used if `-EXPORT_NAME` is set.
`-EXPORT_DIR=<directory>`	No	Directory where the scenario will be exported. The export file created in this directory is named based on the `-FILE_NAME` and `-EXPORT_NAME` parameters. If `-FILE_NAME` or `-EXPORT_NAME` are not specified, the export file is automatically named `SCEN_<scenario_name><scenario_version>.xml`.
`-EXPORT_NAME=<export_name>`	No	Export name. Use this parameter to generate an export file named `SCEN_<export_name>.xml`. This parameter cannot be used with `-FILE_NAME`.
`-FORCE_OVERWRITE=<yes|no>`	No	If set to Yes, overwrites the export file if it already exists. The default value is No.
`-RECURSIVE_EXPORT=<yes|no>`	No	Forces the export of the objects under the scenario. The default value is Yes.
`-XML_VERSION=<1.0>`	No	Version specified in the generated XML file, in the tag `<?xml version="1.0" encoding="ISO-8859-1"?>`. The default value is `1.0`.
`-XML_CHARSET=<encoding>`	No	Encoding specified in the XML file, in the tag `<?xml version="1.0" encoding="ISO-8859-1"?>`. The default value is `ISO-8859-1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
`-JAVA_CHARSET=<encoding>`	No	Target file encoding. The default value is `ISO8859_1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html

Examples

Export the `LOAD_DWH` scenario in version `1` into the `/temp/load_dwh.xml` export file, with all dependent objects.

```
OdiExportScen -SCEN_NAME=LOAD_DWH -SCEN_VERSION=1
-FILE_NAME=/temp/load_dwh.xml -RECURSIVE_EXPORT=yes
```

OdiExportWork

Use this command to export the work repository to a directory or ZIP export file.

Usage

```
OdiExportWork -TODIR=<directory> [-ZIPFILE_NAME=<zipFileName>]
[-XML_CHARSET=<charset>] [-JAVA_CHARSET=<charset>]
```

Parameters

Parameters	Mandatory	Description
`-TODIR=<directory>`	Yes	Target directory for the export.
`-ZIPFILE_NAME=<zipFileName>`	No	Name of the compressed file.
`-XML_CHARSET=<charset>`	No	XML version specified in the export file. Parameter `xml version` in the XML file header. `<?xml version="1.0" encoding="ISO-8859-1"?>`. The default value is `ISO-8859-1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
`-JAVA_CHARSET=<charset>`	No	Result file Java character encoding. The default value is `ISO8859_1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html

Examples

Export and compress the work repository into the `/temp/workexport.zip` export file.

```
OdiExportWork "-TODIR=/temp/" "-ZIPFILE_NAME=workexport.zip"
```

OdiFileAppend

Use this command to concatenate a set of files into a single file.

Usage

```
OdiFileAppend -FILE=<file> -TOFILE=<target_file> [-OVERWRITE=<yes|no>]
[-CASESENS=<yes|no>] [-HEADER=<n>] [-KEEP_FIRST_HEADER=<yes|no]
```

Parameters

Parameters	Mandatory	Description
`-FILE=<file>`	Yes	Full path of the files to concatenate. Use `*` to specify generic characters. Examples: `/var/tmp/*.log` (all files with the `log` extension in the folder `/var/tmp`) `arch_*.lst` (all files starting with `arch_` and with the extension `lst`)
`-TOFILE=<target_file>`	Yes	Target file.
`-OVERWRITE=<yes|no>`	No	Indicates if the target file must be overwritten if it already exists. The default value is No.
`-CASESENS=<yes|no>`	No	Indicates if file search is case-sensitive. By default, Oracle Data Integrator searches files in uppercase (set to No).
`-HEADER=<n>`	No	Number of header lines to be removed from the source files before concatenation. By default, no lines are removed. When the `-HEADER` parameter is omitted, the concatenation does not require file edition, and therefore runs faster.
`-KEEP_FIRST_HEADER=<yes|no>`	No	Keep the header lines of the first file during the concatenation. The default value is Yes.

Examples

Concatenate the files `*.log` of the folder `/var/tmp` into the file `/home/all_files.log`.

```
OdiFileAppend -FILE=/var/tmp/*.log -TOFILE=/home/all_files.log
```

Concatenate the files of the daily sales of each shop while keeping the header of the first file.

```
OdiFileAppend -FILE=/data/store/sales_*.dat -TOFILE=/data/all_stores.dat
-OVERWRITE=yes -HEADER=1 -KEEP_FIRST_HEADER=yes
```

OdiFileCopy

Use this command to copy files or folders.

Usage

```
OdiFileCopy -DIR=<directory> -TODIR=<target_directory> [-OVERWRITE=<yes|no>]
[-RECURSE=<yes|no>] [-CASESENS=<yes|no>]

OdiFileCopy -FILE=<file> -TOFILE=<target_file>|-TODIR=<target_directory>
[-OVERWRITE=<yes|no>] [-RECURSE=<yes|no>] [-CASESENS=<yes|no>]
```

Parameters

Parameters	Mandatory	Description
`-DIR=<directory>`	Yes if `-FILE` is omitted	Directory (or folder) to copy.
`-FILE=<file>`	Yes if `-DIR` is omitted	The full path of the files to copy. Use `*` to specify the generic character. Examples: `/var/tmp/*.log` (all files with the `log` extension in folder `/var/tmp`) `arch_*.lst` (all files starting with `arch_` and with the extension `lst`)
`-TODIR=<target_directory>`	Yes if `-DIR` is specified	Target directory for the copy. If a directory is copied (`-DIR`), this parameter indicates the name of the copied directory. If one or several files are copied (`-FILE`), this parameter indicates the destination directory.
`-TOFILE=<target_file>`	Yes if `-TODIR` is omitted	Destination file(s). This parameter cannot be used with parameter `-DIR`. This parameter contains: ■ The name of the destination file if only one file is copied (no generic character). ■ The mask of the new name of the destination files if several files are copied. Note that `-TODIR` and `-TOFILE` are exclusive parameters. If both are specified, only `-TODIR` is taken into account, and `-TOFILE` is ignored.
`-OVERWRITE=<yes|no>`	No	Indicates if the files of the folder are overwritten if they already exist. The default value is No.
`-RECURSE=<yes|no>`	No	Indicates if files are copied recursively when the directory contains other directories. The value No indicates that only the files within the directory are copied, not the subdirectories. The default value is Yes.
`-CASESENS=<yes|no>`	No	Indicates if file search is case-sensitive. By default, Oracle Data Integrator searches files in uppercase (set to No).

Examples

Copy the file `hosts` from the directory `/etc` to the directory `/home`.

```
OdiFileCopy -FILE=/etc/hosts -TOFILE=/home/hosts
```

Copy all `*.csv` files from the directory `/etc` to the directory `/home` and overwrite.

```
OdiFileCopy -FILE=/etc/*.csv -TODIR=/home -OVERWRITE=yes
```

Copy all `*.csv` files from the directory `/etc` to the directory `/home` while changing their extension to `.txt`.

```
OdiFileCopy -FILE=/etc/*.csv -TOFILE=/home/*.txt -OVERWRITE=yes
```

Copy the directory `C:\odi` and its subdirectories into the directory `C:\Program Files\odi`.

```
OdiFileCopy -DIR=C:\odi "-TODIR=C:\Program Files\odi" -RECURSE=yes
```

OdiFileDelete

Use this command to delete files or directories.

The most common uses of this tool are described in the following table where:

- x means is supplied
- o means is omitted

-DIR	-FILE	-RECURSE	Behavior
x	x	x	Every file with the name or with a name matching the mask specified in `-FILE` is deleted from `-DIR` and from all of its subdirectories.
x	o	x	The subdirectories from `-FILE` are deleted.
x	x	o	Every file with the name or with a name matching the mask specified in `-FILE` is deleted from `-DIR`.
x	o	o	The `-DIR` is deleted.

Usage

```
OdiFileDelete -DIR=<directory> -FILE=<file> [-RECURSE=<yes|no>]
[-CASESENS=<yes|no>] [-NOFILE_ERROR=<yes|no>] [-FROMDATE=<from_date>]
[-TODATE=<to_date>]
```

Parameters

Parameters	Mandatory	Description
`-DIR=<directory>`	Yes if `-FILE` is omitted	If `-FILE` is omitted, specifies the name of the directory (folder) to delete. If `-FILE` is supplied, specifies the path where files should be deleted from.
`-FILE=<file>`	Yes if `-DIR` is omitted	Name or mask of file(s) to delete. If `-DIR` is not specified, provide the full path. Use `*` to specify wildcard characters. Examples: `/var/tmp/*.log` (all files with the `log` extension of the directory `/var/tmp`) `/arch_*.lst` (all files starting with `arch_` and with the extension `lst`)

Parameters	Mandatory	Description
`-RECURSE=<yes|no>`	No	If `-FILE` is omitted, the `-RECURSE` parameter has no effect: all subdirectories are implicitly deleted. If `-FILE` is supplied, the `-RECURSE` parameter specifies if the files should be deleted from this directory and from all of its subdirectories. The default value is Yes.
`-CASESENS=<yes|no>`	No	Specifies that Oracle Data Integrator should distinguish uppercase and lowercase when matching file names. The default value is No.
`-NOFILE_ERROR=<yes|no>`	Yes	Indicates that an error should be generated if the specified directory or files are not found. The default value is Yes.
`-FROMDATE=<from_date>`	No	All files with a modification date later than this date are deleted. Use the format yyyy/MM/dd hh:mm:ss. The `-FROM_DATE` is not inclusive. If `-FROMDATE` is omitted, all files with a modification date earlier than the `-TODATE` date are deleted. If both `-FROMDATE` and `-TODATE` are omitted, all files matching the `-FILE` parameter value are deleted.
`-TODATE=<to_date>`	No	All files with a modification date earlier than this date are deleted. Use the format yyyy/MM/dd hh:mm:ss. The `TO_DATE` is not inclusive. If `-TODATE` is omitted, all files with a modification date later than the `-FROMDATE` date are deleted. If both `-FROMDATE` and `-TODATE` parameters are omitted, all files matching the `-FILE` parameter value are deleted.

Note: You cannot delete a file and a directory at the same time by combining the `-DIR` and `-FILE` parameters. To achieve that, you must make two calls to OdiFileDelete.

Examples

Delete the file `my_data.dat` from the directory `c:\data\input`, generating an error if the file or directory is missing.

```
OdiFileDelete -FILE=c:\data\input\my_data.dat -NOFILE_ERROR=yes
```

Delete all `.txt` files from the `bin` directory, but not `.TXT` files.

```
OdiFileDelete "-FILE=c:\Program Files\odi\bin\*.txt" -CASESENS=yes
```

This statement has the same effect:

```
OdiFileDelete "-DIR=c:\Program Files\odi\bin" "-FILE=*.txt" -CASESENS=yes
```

Delete the directory `/bin/usr/nothingToDoHere`.

```
OdiFileDelete "-DIR=/bin/usr/nothingToDoHere"
```

Delete all files under the `C:\temp` directory whose modification time is between `10/01/2008 00:00:00` and `10/31/2008 22:59:00`, where `10/01/2008` and `10/31/2008` are not inclusive.

```
OdiFileDelete -DIR=C:\temp -FILE=* -NOFILE_ERROR=NO -FROMDATE=FROMDATE=10/01/2008
00:00:00 -TODATE=10/31/2008 22:59:00
```

Delete all files under the `C:\temp` directory whose modification time is earlier than `10/31/2008 17:00:00`.

```
OdiFileDelete -DIR=C:\temp -FILE=* -NOFILE_ERROR=YES -TODATE=10/31/2008 17:00:00
```

Delete all files under the `C:\temp` directory whose modification time is later than `10/01/2008 08:00:00`.

```
OdiFileDelete -DIR=C:\temp -FILE=* -NOFILE_ERROR=NO -FROMDATE=10/01/2008 08:00:00
```

OdiFileMove

Use this command to move or rename files or a directory into files or a directory.

Usage

```
OdiFileMove -FILE=<file> -TODIR=<target_directory> -TOFILE=<target_file>
[-OVERWRITE=<yes|no>] [-RECURSE=<yes|no>] [-CASESENS=<yes|no>]

OdiFileMove -DIR=<directory> -TODIR=<target_directory> [-OVERWRITE=<yes|no>]
[-RECURSE=<yes|no>] [-CASESENS=<yes|no>]
```

Parameters

Parameters	Mandatory	Description
`-DIR=<directory>`	Yes if `-FILE` is omitted	Directory (or folder) to move or rename.
`-FILE=<file>`	Yes if `-DIR` is omitted	Full path of the file(s) to move or rename. Use `*` for generic characters. Examples: `/var/tmp/*.log` (all files with the `log` extension in the directory `/var/tmp`) `arch_*.lst` (all files starting with `arch_` and with the extension `lst`)
`-TODIR=<target_directory>`	Yes if `-DIR` is specified	Target directory of the move. If a directory is moved (`-DIR`), this parameter indicates the new name of the directory. If a file or several files are moved (`-FILE`), this parameter indicates the target directory.

Parameters	Mandatory	Description
`-TOFILE=<target_file>`	Yes if `-TODIR` is omitted	Target file(s). This parameter cannot be used with parameter `-DIR`. This parameter is: ■ The new name of the target file if one single file is moved (no generic character). ■ The mask of the new file names if several files are moved.
`-OVERWRITE=<yes|no>`	No	Indicates if the files or directory are overwritten if they exist. The default value is No.
`-RECURSE=<yes|no>`	No	Indicates if files are moved recursively when the directory contains other directories. The value No indicates that only files contained in the directory to move (not the subdirectories) are moved. The default value is Yes.
`-CASESENS=<yes|no>`	No	Indicates if file search is case-sensitive. By default, Oracle Data Integrator searches files in uppercase (set to No).

Examples

Rename the `hosts` file to `hosts.old`.

```
OdiFileMove -FILE=/etc/hosts -TOFILE=/etc/hosts.old
```

Move the file `hosts` from the directory `/etc` to the directory `/home/odi`.

```
OdiFileMove -FILE=/etc/hosts -TOFILE=/home/odi/hosts
```

Move all files `*.csv` from directory `/etc` to directory `/home/odi` with overwrite.

```
OdiFileMove -FILE=/etc/*.csv -TODIR=/home/odi -OVERWRITE=yes
```

Move all `*.csv` files from directory `/etc` to directory `/home/odi` and change their extension to `.txt`.

```
OdiFileMove -FILE=/etc/*.csv -TOFILE=/home/odi/*.txt -OVERWRITE=yes
```

Rename the directory `C:\odi` to `C:\odi_is_wonderful`.

```
OdiFileMove -DIR=C:\odi -TODIR=C:\odi_is_wonderful
```

Move the directory `C:\odi` and its subfolders into the directory `C:\Program Files\odi`.

```
OdiFileMove -DIR=C:\odi "-TODIR=C:\Program Files\odi" -RECURSE=yes
```

OdiFileWait

Use this command to manage file events. This command regularly scans a directory and waits for a number of files matching a mask to appear, until a given timeout is reached. When the specified files are found, an action on these files is triggered.

Usage

```
OdiFileWait -DIR=<directory> -PATTERN=<pattern>
```

```
[-ACTION=<DELETE|COPY|MOVE|APPEND|ZIP|NONE>] [-TODIR=<target_directory>]
[-TOFILE=<target_file>] [-OVERWRITE=<yes|no>] [-CASESENS=<yes|no>]
[-FILECOUNT=<n>] [-TIMEOUT=<n>] [-POLLINT=<n>] [-HEADER=<n>]
[-KEEP_FIRST_HEADER=<yes|no>] [-NOFILE_ERROR=<yes|no>]
```

Parameters

Parameters	Mandatory	Description
`-ACTION=` `<DELETE|COPY|MOVE|APPEND|ZIP|NONE>`	No	Action taken on the files found: `DELETE`: Delete the files found. `COPY`: Copy the files found into the directory `-TODIR`. `MOVE`: Move or rename the files found into folder `-TODIR` by naming them as specified by `-TOFILE`. `APPEND`: Concatenates all files found and creates a result file `-TOFILE`. Source files are deleted. `ZIP`: Compress the files found and store them in ZIP file `-TOFILE`. `NONE` (default): No action is performed.
`-DIR=<directory>`	Yes	Directory (or folder) to scan.
`-PATTERN=<pattern>`	Yes	Mask of file names to scan. Use `*` to specify the generic characters. Examples: `*.log` (all files with the `log` extension) `arch_*.lst` (all files starting with `arch_` and with the extension `lst`)
`-TODIR=<target_directory>`	No	Target directory of the action. When the action is: `COPY`: Directory where the files are copied. `MOVE`: Directory where the files are moved.
`-TOFILE=<target_file>`	No	Destination file(s). When the action is: `MOVE`: Renaming mask of the moved files. `APPEND`: Name of the file resulting from the concatenation. `ZIP`: Name of the resulting ZIP file. `COPY`: Renaming mask of the copied files. Renaming rules: ■ Any alphanumeric character is replaced in the original file name with the alphanumeric characters specified for `<target_file>`. ■ `?` at `-TOFILE` leaves origin symbol on this position. ■ `*` at `-TOFILE` means all remaining symbols from origin file name.

Parameters	Mandatory	Description
`-OVERWRITE=<yes|no>`	No	Indicates if the destination file(s) will be overwritten if they exist. The default value is No. Note that if this option is used with `APPEND`, the target file will only contain the contents of the latest file processed.
`-CASESENS=<yes|no>`	No	Indicates if file search is case-sensitive. By default, Oracle Data Integrator searches files in uppercase (set to No).
`-FILECOUNT=<n>`	No	Maximum number of files to wait for (the default value is 0). If this number is reached, the command ends. The value 0 indicates that Oracle Data Integrator waits for all files until the timeout is reached. If this parameter is 0 and the timeout is also 0, this parameter is then forced implicitly to 1.
`-TIMEOUT=<n>`	No	Maximum waiting time in milliseconds (the default value is 0). If this delay is reached, the command yields control to the following command and uses its value `-FILECOUNT`. The value 0 is used to specify an infinite waiting time (wait until the maximum number of messages to read as specified in the parameter `-FILECOUNT`).
`-POLLINT=<n>`	No	Interval in milliseconds to search for new files. The default value is 1000 (1 second), which means that Oracle Data Integrator looks for new messages every second. Files written during the OdiFileWait are taken into account only after being closed (file size unchanged) during this interval.
`-HEADER=<n>`	No	This parameter is valid only for the `APPEND` action. Number of header lines to suppress from the files before concatenation. The default value is 0 (no processing).
`-KEEP_FIRST_HEADER=<yes|no>`	No	This parameter is valid only for the `APPEND` action. Keeps the header lines of the first file during the concatenation. The default value is Yes.
`-NOFILE_ERROR=<yes|no>`	No	Indicates the behavior if no file is found. The default value is No, which means that no error is generated if no file is found.

Examples

Wait indefinitely for file `flag.txt` in directory `c:\events` and proceed when this file is detected.

```
OdiFileWait -ACTION=NONE -DIR=c:\events -PATTERN=flag.txt -FILECOUNT=1
-TIMEOUT=0 -POLLINT=1000
```

Wait indefinitely for file `flag.txt` in directory `c:\events` and suppress this file when it is detected.

```
OdiFileWait -ACTION=DELETE -DIR=c:\events -PATTERN=flag.txt -FILECOUNT=1
-TIMEOUT=0 -POLLINT=1000
```

Wait for the sales files `*.dat` for 5 minutes and scan every second in directory `c:\sales_in`, then concatenate into file `sales.dat` in directory `C:\sales_ok`. Keep the header of the first file.

```
OdiFileWait -ACTION=APPEND -DIR=c:\sales_in -PATTERN=*.dat
TOFILE=c:\sales_ok\sales.dat -FILECOUNT=0 -TIMEOUT=350000 -POLLINT=1000
-HEADER=1 -KEEP_FIRST_HEADER=yes -OVERWRITE=yes
```

Wait for the sales files `*.dat` for 5 minutes every second in directory `c:\sales_in`, then copy these files into directory `C:\sales_ok`. Do not overwrite.

```
OdiFileWait -ACTION=COPY -DIR=c:\sales_in -PATTERN=*.dat -TODIR=c:\sales_ok
-FILECOUNT=0 -TIMEOUT=350000 -POLLINT=1000 -OVERWRITE=no
```

Wait for the sales files `*.dat` for 5 minutes every second in directory `c:\sales_in` and then archive these files into a ZIP file.

```
OdiFileWait -ACTION=ZIP -DIR=c:\sales_in -PATTERN=*.dat
-TOFILE=c:\sales_ok\sales.zip -FILECOUNT=0 -TIMEOUT=350000
-POLLINT=1000 -OVERWRITE=yes
```

Wait for the sales files `*.dat` for 5 minutes every second into directory `c:\sales_in`, then move these files into directory `C:\sales_ok`. Do not overwrite. Append `.bak` to the file names.

```
OdiFileWait -ACTION=MOVE -DIR=c:\sales_in -PATTERN=*.dat
-TODIR=c:\sales_ok -TOFILE=*.bak -FILECOUNT=0 -TIMEOUT=350000
-POLLINT=1000 -OVERWRITE=no
```

OdiFtp

Use this command to use the FTP protocol to connect to a remote system and to perform standard FTP commands on the remote system. Trace from the script is recorded against the task representing the OdiFtp step in Operator Navigator.

Usage

```
OdiFtp -HOST=<ftp server host name> -USER=<ftp user>
[-PASSWORD=<ftp user password>] -REMOTE_DIR=<remote dir on ftp host>
-LOCAL_DIR=<local dir> [-PASSIVE_MODE=<yes|no>] [-TIMEOUT=<time in seconds>]
[-STOP_ON_FTP_ERROR=<yes|no>] -COMMAND=<command>
```

Parameters

Parameters	Mandatory	Description
`-HOST=<ftp server host name>`	Yes	Host name of the FTP server.
`-USER=<ftp user>`	Yes	User on the FTP server.
`-PASSWORD=<ftp user password>`	No	Password of the FTP user.

Parameters	Mandatory	Description
`-REMOTE_DIR=<remote dir on ftp host>`	Yes	Directory path on the remote FTP host.
`-LOCAL_DIR=<local dir>`	Yes	Directory path on the local machine.
`-PASSIVE_MODE=<yes|no>`	No	If set to No, the FTP session uses Active Mode. The default value is Yes, which means the session runs in passive mode.
`-TIMEOUT=<time in seconds>`	No	Time in seconds after which the socket connection times out.
`-STOP_ON_FTP_ERROR=<yes|no>`	No	If set to Yes (default), the step stops when an FTP error occurs instead of running to completion.
`-COMMAND=<command>`	Yes	Raw FTP command to execute. For a multiline command, pass the whole command as raw text after the OdiFtp line without the `-COMMAND` parameter. Supported commands: `APPE`, `CDUP`, `CWD`, `DELE`, `LIST`, `MKD`, `NLST`, `PWD`, `QUIT`, `RETR`, `RMD`, `RNFR`, `RNTO`, `SIZE`, `STOR`

Examples

Execute a script on a remote host that makes a directory, changes directory into the directory, puts a file into the directory, and checks its size. The script appends another file, checks the new size, and then renames the file to `dailyData.csv`. The `-STOP_ON_FTP_ERROR` parameter is set to `No` so that the script continues even if the directory exists.

```
OdiFtp -HOST=machine.oracle.com -USER=odiftpuser -PASSWORD=<password>
-LOCAL_DIR=/tmp -REMOTE_DIR=c:\temp -PASSIVE_MODE=YES -STOP_ON_FTP_ERROR=No
MKD dataDir
CWD dataDir
STOR customers.csv
SIZE customers.csv
APPE new_customers.csv customers.csv
SIZE customers.csv
RNFR customers.csv
RNTO dailyData.csv
```

OdiFtpGet

Use this command to download a file from an FTP server.

Usage

```
OdiFtpGet -HOST=<ftp server host name> -USER=<ftp user>
[PASSWORD=<ftp user password>] -REMOTE_DIR=<remote dir on ftp host>
[-REMOTE_FILE=<file name under the -REMOTE_DIR>] -LOCAL_DIR=<local dir>
[-LOCAL_FILE=<file name under the -LOCAL_DIR>] [-PASSIVE_MODE=<yes|no>]
[-TIMEOUT=<time in seconds>]
```

Parameters

Parameters	Mandatory	Description
`-HOST=<host name of the ftp server>`	Yes	Host name of the FTP server.
`-USER=<host name of the ftp user>`	Yes	User on the FTP server.
`-PASSWORD=<password of the ftp user>`	No	Password of the FTP user.
`-REMOTE_DIR=<dir on the ftp host>`	Yes	Directory path on the remote FTP host.
`-REMOTE_FILE=<file name under -REMOTE DIR>`	No	File name under the directory specified in the `-REMOTE_DIR` argument. If this argument is missing, the file is copied with the `-LOCAL_FILE` file name. If the `-LOCAL_FILE` argument is also missing, the `-LOCAL_DIR` is copied recursively to the `-REMOTE_DIR`.
`-LOCAL_DIR=<local dir path>`	Yes	Directory path on the local machine.
`-LOCAL_FILE=<local file>`	No	File name under the directory specified in the `-LOCAL_DIR` argument. If this argument is missing, all files and directories under the `-LOCAL_DIR` are copied recursively to the `-REMOTE_DIR`. To filter the files to be copied, use `*` to specify the generic characters. Examples: ■ `*.log` (all files with the `log` extension) ■ `arch_*.lst` (all files starting with `arch_` and with the extension `lst`)
`-PASSIVE_MODE=<yes|no>]`	No	If set to No, the FTP session uses Active Mode. The default value is Yes, which means the session runs in passive mode.
`-TIMEOUT=<time in seconds>`	No	The time in seconds after which the socket connection times out.

Examples

Copy the remote directory `/test_copy555` on the FTP server recursively to the local directory `C:\temp\test_copy`.

```
OdiFtpGet -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password> -LOCAL_
DIR=C:\temp\test_copy -REMOTE_DIR=/test_copy555
```

Copy all files matching the `Sales*.txt` pattern under the remote directory `/` on the FTP server to the local directory `C:\temp\` using Active Mode for the FTP connection.

```
OdiFtpGet -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password> -LOCAL_
DIR=C:\temp -LOCAL_FILE=Sales*.txt -REMOTE_DIR=/ -PASSIVE_MODE=NO
```

OdiFtpPut

Use this command to upload a local file to an FTP server.

Usage

```
OdiFtpPut -HOST=<ftp server host name> -USER=<ftp user>
[PASSWORD=<ftp user password>] -REMOTE_DIR=<remote dir on ftp host>
[-REMOTE_FILE=<file name under the -REMOTE_DIR>] -LOCAL_DIR=<local dir>
```

```
[-LOCAL_FILE=<file name under the -LOCAL_DIR>] [-PASSIVE_MODE=<yes|no>]
[-TIMEOUT=<time in seconds>]
```

Parameters

Parameters	Mandatory	Description
`-HOST=<host name of the ftp server>`	Yes	Host name of the FTP server.
`-USER=<host name of the ftp user>`	Yes	User on the FTP server.
`-PASSWORD=<password of the ftp user>`	No	Password of the FTP user.
`-REMOTE_DIR=<dir on the ftp host>`	Yes	Directory path on the remote FTP host.
`-REMOTE_FILE=<file name under -REMOTE DIR>`	No	File name under the directory specified in the `-REMOTE_DIR` argument. If this argument is missing, the file is copied with the `-LOCAL_FILE` file name. If the `-LOCAL_FILE` argument is also missing, the `-LOCAL_DIR` is copied recursively to the `-REMOTE_DIR`.
`-LOCAL_DIR=<local dir path>`	Yes	Directory path on the local machine.
`-LOCAL_FILE=<local file>`	No	File name under the directory specified in the `-LOCAL_DIR` argument. If this argument is missing, all files and directories under the `-LOCAL_DIR` are copied recursively to the `-REMOTE_DIR`. To filter the files to be copied, use * to specify the generic characters. Examples: ▪ `*.log` (all files with the `log` extension) ▪ `arch_*.lst` (all files starting with `arch_` and with the extension `lst`)
`-PASSIVE_MODE=<yes|no>`	No	If set to No, the FTP session uses Active Mode. The default value is Yes, which means the session runs in passive mode.
`-TIMEOUT=<time in seconds>`	No	The time in seconds after which the socket connection times out.

Examples

Copy the local directory `C:\temp\test_copy` recursively to the remote directory `/test_copy555` on the FTP server.

```
OdiFtpPut -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password>
 -LOCAL_DIR=C:\temp\test_copy -REMOTE_DIR=/test_copy555"
```

Copy all files matching the `Sales*.txt` pattern under the local directory `C:\temp\` to the remote directory `/` on the FTP server.

```
OdiFtpPut -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password> -LOCAL_
DIR=C:\temp -LOCAL_FILE=Sales*.txt -REMOTE_DIR=/
```

Copy the `Sales1.txt` file under the local directory `C:\temp\` to the remote directory `/` on the FTP server as a `Sample1.txt` file.

```
OdiFtpPut -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password> -LOCAL_
DIR=C:\temp -LOCAL_FILE=Sales1.txt -REMOTE_DIR=/Sample1.txt
```

OdiGenerateAllScen

Use this command to generate a set of scenarios from design-time components (Packages, Mappings, Procedures, or Variables) contained in a folder or project, filtered by markers.

Usage

```
OdiGenerateAllScen -PROJECT=<project_id> [-FOLDER=<folder_id>]
[-MODE=<REPLACE|CREATE>] [-GRPMARKER=<marker_group_code>
[-MARKER=<marker_code>] [-MATERIALIZED=<yes|no>]
[-GENERATE_MAP=<yes|no>] [-GENERATE_PACK=<yes|no>]
[-GENERATE_POP=<yes|no>] [-GENERATE_TRT=<yes|no>]
[-GENERATE_VAR=<yes|no>]
```

Parameters

Parameters	Mandatory	Description
`-PROJECT=<project_id>`	Yes	ID of the Project containing the components to generate scenarios for.
`-FOLDER=<folder_id>`	No	ID of the Folder containing the components to generate scenarios for.
`-MODE=<REPLACE|CREATE>`	No	Scenario generation mode: ■ `REPLACE` (default): Causes the last scenario generated for the component to be replaced by the new one generated, with no change of name or version. Any schedules linked to this scenario are deleted. If no scenario exists, a new one is generated. ■ `CREATE`: Creates a new scenario with the same name as the latest scenario generated for the component, with the version number automatically incremented (if the latest version is an integer) or set to the current date (if the latest version is not an integer). If no scenario has been created 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.
`-GRPMARKER=<marker_group_code>`	No	Group containing the marker used to filter the components for which scenarios must be generated. When `-GRPMARKER` and `-MARKER` are specified, scenarios will be (re-)generated only for components flagged with the marker identified by the marker code and the marker group code.

Parameters	Mandatory	Description
`-MARKER=<marker_code>`	No	Marker used to filter the components for which scenarios must be generated. When `-GRPMARKER` and `-MARKER` are specified, scenarios will be (re-)generated only for components flagged with the marker identified by the marker code and the marker group code.
`-MATERIALIZED=<yes|no>`	No	Specifies whether scenarios should be generated as if all underlying objects are materialized. The default value is No.
`-GENERATE_MAP=<yes|no>`	No	Specifies whether scenarios should be generated from the mapping. The default value is No.
`-GENERATE_PACK=<yes|no>`	No	Specifies whether scenarios attached to packages should be (re-)generated. The default value is Yes.
`-GENERATE_POP=<yes|no>`	No	Specifies whether scenarios attached to mappings should be (re-)generated. The default value is No.
`-GENERATE_TRT=<yes|no>`	No	Specifies whether scenarios attached to procedures should be (re-)generated. The default value is No.
`-GENERATE_VAR=<yes|no>`	No	Specifies whether scenarios attached to variables should be (re-)generated. The default value is No.

Examples

Generate all scenarios in the project whose ID is `1003` for the current repository.

```
OdiGenerateAllScen -PROJECT=1003
```

OdiImportObject

Use this command to import the contents of an export file into a repository. This command reproduces the behavior of the import feature available from the user interface.

Use caution when using this tool. It may work incorrectly when importing objects that depend on objects that do not exist in the repository. It is recommended that you use this API for importing high-level objects (projects, models, and so on).

WARNING: The import type and the order in which objects are imported into a repository should be carefully specified. Refer to Chapter 20, "Exporting and Importing," for more information on import.

Usage

```
OdiImportObject -FILE_NAME=<FileName> [-WORK_REP_NAME=<workRepositoryName>]
-IMPORT_MODE=<DUPLICATION|SYNONYM_INSERT|SYNONYM_UPDATE|SYNONYM_INSERT_UPDATE>]
[-IMPORT_SCHEDULE=<yes|no>] [-UPGRADE_KEY=<upgradeKey>]
```

Parameters

Parameters	Mandatory	Description
`-FILE_NAME=<FileName>`	Yes	Name of the XML export file to import.

Parameters	Mandatory	Description
`-WORK_REP_NAME=<workRepositoryName>`	No	Name of the work repository into which the object must be imported. This work repository must be defined in the connected master repository. If this parameter is not specified, the object is imported into the current master or work repository.
`-IMPORT_MODE=<DUPLICATION|SYNONYM_INSERT|SYNONYM_UPDATE|SYNONYM_INSERT_UPDATE>`	Yes	Import mode for the object. The default value is `DUPLICATION`. For more information about import types, see "Import Types" on page 20-3.
`-IMPORT_SCHEDULE=<yes|no>`	No	If the selected file is a scenario export, imports the schedules contained in the scenario export file. The default value is No.
`-UPGRADE_KEY=<upgradeKey>`	No	Upgrade key to import repository objects from earlier versions of Oracle Data Integrator (pre-12*c*).

Examples

Import the `/temp/DW01.xml` export file (a project) into the `WORKREP` work repository using `DUPLICATION` mode.

```
OdiImportObject -FILE_NAME=/temp/DW01.xml -WORK_REP_NAME=WORKREP
-IMPORT_MODE=DUPLICATION
```

OdiImportScen

Use this command to import a scenario into the current work repository from an export file.

Usage

```
OdiImportScen -FILE_NAME=<FileName>
[-IMPORT_MODE=<DUPLICATION|SYNONYM_INSERT|SYNONYM_UPDATE|SYNONYM_INSERT_UPDATE>]
[-IMPORT_SCHEDULE=<yes|no>] [-FOLDER=<parentFolderGlobalId>]
[-UPGRADE_KEY=<upgradeKey>]
```

Parameters

Parameters	Mandatory	Description
`-FILE_NAME=<FileName>`	Yes	Name of the export file.
`-IMPORT_MODE=<DUPLICATION|SYNONYM_INSERT|SYNONYM_UPDATE|SYNONYM_INSERT_UPDATE>`	No	Import mode of the scenario. The default value is `DUPLICATION`. For more information about import types, see "Import Types" on page 20-3.
`-IMPORT_SCHEDULE=<yes|no>`	No	Imports the schedules contained in the scenario export file. The default value is No.
`-FOLDER=<parentFolderGlobalId>`	No	Global ID of the parent scenario folder.

Parameters	Mandatory	Description
`-UPGRADE_KEY=<upgradeKey>`	No	Upgrade key to import repository objects from earlier versions of Oracle Data Integrator (pre-12*c*).

Examples

Import the `/temp/load_dwh.xml` export file (a scenario) into the current work repository using `DUPLICATION` mode.

```
OdiImportScen -FILE_NAME=/temp/load_dwh.xml -IMPORT_MODE=DUPLICATION
```

OdiInvokeWebService

Note: This tool replaces the OdiExecuteWebService tool.

Use this command to invoke a web service over HTTP/HTTPS and write the response to an XML file.

This tool invokes a specific *operation* on a *port* of a web service whose description file (WSDL) *URL* is provided.

If this operation requires a web service request, it is provided either in a request file, or directly written out in the tool call (`<XML Request>`). This request file can have two different formats (`XML`, which corresponds to the XML body only, or `SOAP`, which corresponds to the full-formed SOAP envelope including a SOAP header and body) specified in the `-RESPONSE_FILE_FORMAT` parameter. The response of the web service request is written to an XML file that can be processed afterwards in Oracle Data Integrator. If the web service operation is one-way and does not return any response, no response file is generated.

Note: This tool cannot be executed in a command line with `startcmd`.

Usage

```
OdiInvokeWebService -URL=<url> -PORT=<port> -OPERATION=<operation>
[<XML Request>] [-REQUEST_FILE=<xml_request_file>]
[-RESPONSE_MODE=<NO_FILE|NEW_FILE|FILE_APPEND>]
[-RESPONSE_FILE=<xml_response_file>] [-RESPONSE_XML_ENCODING=<charset>]
[-RESPONSE_FILE_CHARSET=<charset>] [-RESPONSE_FILE_FORMAT=<XML|SOAP>]
[-HTTP_USER=<user>]
[-HTTP_PASS=<password>] [-TIMEOUT=<timeout>]
```

Parameters

Parameters	Mandatory	Description
`-URL=<url>`	Yes	URL of the Web Service Description File (WSDL) file describing the web service.
`-PORT_TYPE=<port_type>`	Yes	Name of the WSDL port type to invoke.
`-OPERATION=<operation>`	Yes	Name of the web service operation to invoke.

Parameters	Mandatory	Description
`<XML Request>`	No	Request message in SOAP (Simple Object Access Protocol) format. This message should be provided on the line immediately following the OdiInvokeWebService call. The request can alternately be passed through a file whose location is provided with the `-REQUEST_FILE` parameter.
`-REQUEST_FILE=<xml_request_file>`	No	Location of the XML file containing the request message in SOAP format. The request can alternately be directly written out in the tool call (`<xmlRequest>`).
`-RESPONSE_MODE=<NO_FILE|NEW_FILE|FILE_APPEND>`	No	Generation mode for the response file. This parameter takes the following values: ■ `NO_FILE` (default): No response file is generated. ■ `NEW_FILE`: A new response file is generated. If the file already exists, it is overwritten. ■ `FILE_APPEND`: The response is appended to the file. If the file does not exist, it is created.
`-RESPONSE_FILE=<file>`	Depends	The name of the result file to write. Mandatory if `-RESPONSE_MODE` is `NEW_FILE` or `APPEND`.
`-RESPONSE_FILE_CHARSET=<charset>`	Depends	Response file character encoding. See the following table. Mandatory if `-RESPONSE_MODE` is `NEW_FILE` or `APPEND`.
`-RESPONSE_XML_ENCODING=<charset>`	Depends	Character encoding that will be indicated in the XML declaration header of the response file. See the following table. Mandatory if `-RESPONSE_MODE` is not `NO_FILE`.
`-RESPONSE_FILE_FORMAT=<XML|SOAP>`	No	Format of the request and response file. ■ If `XML` is selected (default), the request is processed as a SOAP body. The tool adds a default SOAP header and envelope content to this body before sending the request. The response is stripped from its SOAP envelope and headers and only the response's body is written to the response file. ■ If `SOAP` is selected, the request is processed as a full-formed SOAP envelope and is sent as is. The response is also written to the response file with no processing.
`-HTTP_USER=<user>`	No	User account authenticating on the HTTP server.
`-HTTP_PASS=<password>`	No	Password of the HTTP user.

Parameters	Mandatory	Description
`-TIMEOUT=<timeout>`	No	The web service request waits for a reply for this amount of time before considering that the server will not provide a response and an error is produced. The default value is 15 seconds.

The following table lists the most common XML/Java character encoding schemes. For a more complete list, see:

http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html

XML Charset	Java Charset
US-ASCII	ASCII
UTF-8	UTF8
UTF-16	UTF-16
ISO-8859-1	ISO8859_1

Examples

The following web service call returns the capital city for a given country (the ISO country code is sent in the request). Note that the request and response format, as well as the port and operations available, are defined in the WSDL passed in the URL parameter.

```
OdiInvokeWebService -
-URL=http://www.oorsprong.org/websamples.countryinfo/CountryInfoService.wso
?WSDL -PORT_TYPE=CountryInfoServiceSoapType -OPERATION=CapitalCity
-RESPONSE_MODE=NEW_FILE -RESPONSE_XML_ENCODING=ISO-8859-1
"-RESPONSE_FILE=/temp/result.xml" -RESPONSE_FILE_CHARSET=ISO8859_1 -RESPONSE_FILE_
FORMAT=XML
<CapitalCityRequest>
<sCountryISOCode>US</sCountryISOCode>
</CapitalCityRequest>
```

The generated `/temp/result.xml` file contains the following:

```
<CapitalCityResponse>
<m:CapitalCityResponse>
<m:CapitalCityResult>Washington</m:CapitalCityResult>
</m:CapitalCityResponse>
</CapitalCityResponse>
```

Packages

Oracle Data Integrator provides a special graphical interface for calling OdiInvokeWebService in packages. See Chapter 16, "Using Web Services," for more information.

OdiKillAgent

Use this command to stop a standalone agent.

Java EE Agents deployed in an application server cannot be stopped using this tool and must be stopped using the application server utilities.

Usage

```
OdiKillAgent (-PORT=<TCP/IP Port>|-NAME=<physical_agent_name>)
[-IMMEDIATE=<yes|no>] [-MAX_WAIT=<timeout>]
```

Parameters

Parameters	Mandatory	Description
`-PORT=<TCP/IP Port>`	No	If this parameter is specified, the agent running on the local machine with the specified port is stopped.
`-NAME=<physical_agent_name>`	Yes	If this parameter is specified, the physical agent whose name is provided is stopped. This agent may be a local or remote agent. It must be declared in the master repository.
`-IMMEDIATE=<yes|no>`	No	If this parameter is set to Yes, the agent is stopped without waiting for its running sessions to complete. If this parameter is set to No, the agent is stopped after its running sessions reach completion or after the `-MAX_WAIT` timeout is reached. The default value is No.
`-MAX_WAIT=<timeout>`	No	This parameter can be used when `-IMMEDIATE` is set to No. The parameter defines a timeout in milliseconds after which the agent is stopped regardless of the running sessions. The default value is 0, which means no timeout and the agent is stopped after its running sessions complete.

Examples

Stop the `ODI_AGT_001` physical agent immediately.

```
OdiKillAgent -NAME=ODI_AGT_001 -IMMEDIATE=yes
```

OdiManageOggProcess

Use this command to start and stop Oracle GoldenGate processes.

The `-NB_PROCESS` parameter specifies the number of processes on which to perform the operation and applies only to Oracle GoldenGate Delivery processes.

If `-NB_PROCESS` is not specified, the name of the physical process is derived from the logical process. For example, if logical schema `R1_LS` maps to physical process `R1`, an Oracle GoldenGate process named `R1` is started or stopped.

If `-NB_PROCESS` is specified with a positive value, sequence numbers are appended to the process and all processes are started or stopped with the new name. For example, if the value is set to `3`, and logical schema `R2_LS` maps to physical process `R2`, processes `R21`, `R22` and `R23` are started or stopped.

If Start Journal is used to start the CDC (Changed Data Capture) process with Oracle GoldenGate JKMs (Journalizing Knowledge Modules), Oracle Data Integrator generates the Oracle GoldenGate Delivery process with the additional sequence number in the process name. For example, if Delivery process `RP` is used for the Start Journal action, Start Journal generates an Oracle GoldenGate Delivery process named `RP1`. To stop and start the process using the OdiManageOggProcess tool, set `-NB_PROCESS` to `1`. The maximum value of `-NB_PROCESS` is the value of the `-NB_APPLY_PROCESS` parameter of the JKM within the model.

Usage

```
OdiManageOggProcess -OPERATION=<start|stop>
-PROCESS_LSCHEMA=<OGG logical schema> [-NB_PROCESS=<number of processes>]
```

Parameters

Parameters	Mandatory	Description
`-OPERATION=<start|stop>`	Yes	Operation to perform on the process.
`-PROCESS_LSCHEMA=<OGG logical schema>`	Yes	Logical schema of the process.
`-NB_PROCESS=<number of processes>`	No	Number of processes on which to perform the operation.

Examples

Start Oracle GoldenGate process `R1`, which maps to logical schema `R1_LS`.

```
OdiManageOggProcess "-OPERATION=START" "-PROCESS_LSCHEMA=R1_LS
```

Start Oracle GoldenGate processes `R21`, `R22`, and `R23`.

```
OdiManageOggProcess "-OPERATION=START" "-PROCESS_LSCHEMA=R2_LS" "-NB_PROCESS=3"
```

OdiMkDir

Use this command to create a directory structure.

If the parent directory does not exist, this command recursively creates the parent directories.

Usage

```
OdiMkDir -DIR=<directory>
```

Parameters

Parameters	Mandatory	Description
`-DIR=<directory>`	Yes	Directory (or folder) to create.

Examples

Create the directory `odi` in `C:\temp`. If `C:\temp` does not exist, it is created.

```
OdiMkDir "-DIR=C:\temp\odi"
```

OdiOSCommand

Use this command to invoke an operating system command shell to carry out a command, and redirect the output result to files.

The following operating systems are supported:

- Windows operating systems, using `cmd`
- POSIX-compliant operating systems, using `sh`

The following operating systems are not supported:

- Mac OS

Usage

```
OdiOSCommand [-OUT_FILE=<stdout_file>] [-ERR_FILE=<stderr_file>]
[-FILE_APPEND=<yes|no>] [-WORKING_DIR=<workingdir>] [-SYNCHRONOUS=<yes|no>]
[CR/LF <command> | -COMMAND=<command>]
```

Parameters

Parameters	Mandatory	Description
`-COMMAND=<command>`	Yes	Command to execute. For a multiline command, pass the whole command as raw text after the OdiOSCommand line without the `-COMMAND` parameter.
`-OUT_FILE=<stdout_file>`	No	Absolute name of the file to redirect standard output to.
`-ERR_FILE=<stderr_file>`	No	Absolute name of the file to redirect standard error to.
`-FILE_APPEND=<yes|no>`	No	Whether to append to the output files, rather than overwriting them. The default value is Yes.
`-WORKING_DIR=<workingdir>`	No	Directory in which the command is executed.
`-SYNCHRONOUS=<yes|no>`	No	If set to Yes (default), the session waits for the command to terminate. If set to No, the session continues immediately with error code 0. The default is synchronous mode.

Examples

Execute the file `c:\work\load.bat` (on a Windows machine) and append the output streams to files.

```
OdiOSCommand "-OUT_FILE=c:\work\load-out.txt"
"-ERR_FILE=c:\work\load-err.txt" "-FILE_APPEND=YES"
"-WORKING_DIR=c:\work" c:\work\load.bat
```

OdiOutFile

Use this command to write or append content to a text file.

Usage

```
OdiOutFile -FILE=<file_name> [-APPEND] [-CHARSET_ENCODING=<encoding>]
[-XROW_SEP=<hexadecimal_line_break>] [CR/LF <text> | -TEXT=<text>]
```

Parameters

Parameters	Mandatory	Description
`-FILE=<file_name>`	Yes	Target file. Its path may be absolute or relative to the execution agent location.

Parameters	Mandatory	Description
-APPEND	No	Indicates whether <text> must be appended at the end of the file. If this parameter is not specified, the file is overwritten if it exists.
-CHARSET_ENCODING=<encoding>	No	Target file encoding. The default value is ISO-8859-1. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
-XROW_SEP=<hexadecimal_line_break>	No	Hexadecimal code of the character used as a line separator (line break). The default value is 0A (UNIX line break). For a Windows line break, the value is 0D0A.
CR/LF <text> or -TEXT=<text>	No	Text to write in the file. This text can be typed on the line following the OdiOutFile command (a carriage return - CR/LF - indicates the beginning of the text), or can be defined with the -TEXT parameter. The -TEXT parameter should be used when calling this Oracle Data Integrator command from an OS command line. The text can contain variables or substitution methods.

Examples

Generate the file /var/tmp/my_file.txt on the UNIX system of the agent that executed it.

```
OdiOutFile -FILE=/var/tmp/my_file.txt
Welcome to Oracle Data Integrator
This file has been overwritten by <%=odiRef.getSession("SESS_NAME")%>
```

Add the entry PLUTON into the file hosts of the Windows system of the agent that executed it.

```
OdiOutFile -FILE=C:\winnt\system32\drivers\etc\hosts -APPEND
195.10.10.6 PLUTON pluton
```

OdiPingAgent

Use this command to perform a test on a given agent. If the agent is not started, this command raises an error.

Usage

```
OdiPingAgent -AGENT_NAME=<physical_agent_name>
```

Parameters

Parameters	Mandatory	Description
-AGENT_NAME=<physical_agent_name>	Yes	Name of the physical agent to test.

Examples

Test the physical agent AGENT_SOLARIS_DEV.

```
OdiPingAgent -AGENT_NAME=AGENT_SOLARIS_DEV
```

OdiPurgeLog

Use this command to purge the execution logs.

The OdiPurgeLog tool purges all session logs and/or Load Plan runs that match the filter criteria.

The `-PURGE_TYPE` parameter defines the objects to purge:

- Select `SESSION` to purge all session logs matching the criteria. Child sessions and grandchild sessions are purged if the parent session matches the criteria. Note that sessions launched by a Load Plan execution, including the child sessions, are *not* purged.
- Select `LOAD_PLAN_RUN` to purge all load plan logs matching the criteria. Note that all sessions launched from the Load Plan run are purged even if the sessions attached to the Load Plan runs themselves do not match the criteria.
- Select `ALL` to purge both session logs and Load Plan runs matching the criteria.

The `-COUNT` parameter defines the number of sessions and/or Load Plan runs (after filter) to preserve in the log. The `-ARCHIVE` parameter enables automatic archiving of the purged sessions and/or Load Plan runs.

Note: Load Plans and sessions in running, waiting, or queued status are not purged.

Usage

```
OdiPurgeLog
[-PURGE_TYPE=<SESSION|LOAD_PLAN_RUN|ALL>]
[-COUNT=<session_number>] [-FROMDATE=<from_date>] [TODATE=<to_date>]
[-CONTEXT_CODE=<context_code>] [-USER_NAME=<user_name>]
[-AGENT_NAME=<agent_name>] [-PURGE_REPORTS=<Yes|No>] [-STATUS=<D|E|M>]
[-NAME=<session_or_load_plan_name>] [-ARCHIVE=<Yes|No>] [-TODIR=<directory>]
[-ZIPFILE_NAME=<zipfile_name>] [-XML_CHARSET=<charset>] [-JAVA_CHARSET=<charset>]
[-REMOVE_TEMPORARY_OBJECTS=<yes|no>]
```

Parameter

Parameters	Mandatory	Description
`-PURGE_TYPE=<SESSION|LOAD_PLAN_RUN|ALL>`	No	Purges only session logs, Load Plan logs, or both. The default is session.
`-COUNT=<session_number>`	No	Retains the most recent count number of sessions and/or Load Plan runs that match the specified filter criteria and purges the rest. If this parameter is not specified or equals 0, purges all sessions and/or Load Plan runs that match the filter criteria.
`-FROMDATE=<from_date>`	No	Starting date for the purge, using the format yyyy/MM/dd hh:mm:ss. If `-FROMDATE` is omitted, the purge is done starting with the oldest session and/or Load Plan run.

Parameters	Mandatory	Description
`-TODATE=<to_date>`	No	Ending date for the purge, using the format yyyy/MM/dd hh:mm:ss. If `-TODATE` is omitted, the purge is done up to the most recent session and/or Load Plan run.
`-CONTEXT_CODE=<context_code>`	No	Purges only sessions and/or Load Plan runs executed in `<context_code>`. If `-CONTEXT_CODE` is omitted, the purge is done on all contexts.
`-USER_NAME=<user_name>`	No	Purges only sessions and/or Load Plan runs launched by `<user_name>`.
`-AGENT_NAME=<agent_name>`	No	Purges only sessions and/or Load Plan runs executed by `<agent_name>`.
`-PURGE_REPORTS=<0|1>`	No	If set to 1, scenario reports (appearing under the execution node of each scenario) are also purged.
`-STATUS=<D|E|M>`	No	Purges only the sessions and/or Load Plan runs with the specified state: ■ `D`: Done ■ `E`: Error ■ `M`: Warning If this parameter is not specified, sessions and/or Load Plan runs in all of these states are purged.
`-NAME=<session_or_load_plan_name>`	No	Session name or Load Plan name.
`-ARCHIVE=<Yes|No>`	No	If set to Yes, exports the sessions and/or Load Plan runs before they are purged.
`-TODIR=<directory>`	No	Target directory for the export. This parameter is required if `-ARCHIVE` is set to Yes.
`-ZIPFILE_NAME=<zipfile_name>`	No	Name of the compressed file. Target directory for the export. This parameter is required if `-ARCHIVE` is set to Yes.
`-XML_CHARSET=<charset>`	No	XML encoding of the export files. The default value is `ISO-8859-1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
`-JAVA_CHARSET=<charset>`	No	Export file encoding. The default value is `ISO8859_1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
`-REMOVE_TEMPORARY_OBJECTS=<yes|no>`	No	If set to Yes (default), cleanup tasks are performed before sessions are purged so that any temporary objects are removed.

Examples

Purge all sessions executed between 2001/03/25 00:00:00 and 2001/08/31 21:59:00.

```
OdiPurgeLog "-FROMDATE=2001/03/25 00:00:00" "-TODATE=2001/08/31 21:59:00"
```

Purge all Load Plan runs that were executed in the `GLOBAL` context by the `Internal` agent and that are in Error status.

```
OdiPurgeLog "-PURGE_TYPE=LOAD_PLAN_RUN" "-CONTEXT_CODE=GLOBAL"
"-AGENT_NAME=Internal" "-STATUS=E"
```

OdiReadMail

Use this command to read emails and attachments from a POP or IMAP account.

This command connects the mail server `-MAILHOST` using the connection parameters specified by `-USER` and `-PASS`. The execution agent reads messages from the mailbox until `-MAX_MSG` messages are received or the maximum waiting time specified by `-TIMEOUT` is reached. The extracted messages must match the filters such as those specified by the parameters `-SUBJECT` and `-SENDER`. When a message satisfies these criteria, its content and its attachments are extracted in a directory specified by the parameter `-FOLDER`. If the parameter `-KEEP` is set to No, the retrieved message is suppressed from the mailbox.

Usage

```
OdiReadMail -MAILHOST=<mail_host> -USER=<mail_user>
-PASS=<mail_user_password> -FOLDER=<folder_path>
[-PROTOCOL=<pop3|imap>] [-FOLDER_OPT=<none|sender|subject>]
[-KEEP=<no|yes>] [-EXTRACT_MSG=<yes|no>] [-EXTRACT_ATT=<yes|no>]
[-MSG_PRF=<my_prefix>] [-ATT_PRF=<my_prefix>] [-USE_UCASE=<no|yes>]
[-NOMAIL_ERROR=<no|yes>] [-TIMEOUT=<timeout>] [-POLLINT=<pollint>]
[-MAX_MSG=<max_msg>] [-SUBJECT=<subject_filter>] [-SENDER=<sender_filter>]
[-TO=<to_filter>] [-CC=<cc_filter>]
```

Parameters

Parameters	Mandatory	Description
`-MAILHOST=<mail_host>`	Yes	IP address of the POP or IMAP mail server.
`-USER=<mail_user>`	Yes	Valid mail server account.
`-PASS=<mail_user_password>`	Yes	Password of the mail server account.
`-FOLDER=<folder_path>`	Yes	Full path of the storage folder for attachments and messages.
`-PROTOCOL=<pop3|imap>`	No	Type of mail accessed (POP3 or IMAP). The default is POP3.

Parameters	Mandatory	Description
`-FOLDER_OPT=<none|sender|subject>`	No	Allows the creation of a subdirectory in the directory `-FOLDER` according to the following parameters: ■ `none` (default): No action. ■ `sender`: A subdirectory is created with the external name of the sender. ■ `subject`: A subdirectory is created with the subject of the message. For the `sender` and `subject` folder options, the spaces and nonalphanumeric characters (such as `@`) are replaced by underscores in the generated folder's name.
`-KEEP=<no|yes>`	No	If set to Yes, keep the messages that match the filters in the mailbox after reading them. If set to No (default), delete the messages that match the filters of the mailbox after reading them.
`-EXTRACT_MSG=<yes|no>`	No	If set to Yes (default), extract the body of the message into a file. If set to No, do not extract the body of the message into a file.
`-EXTRACT_ATT=<yes|no>`	No	If set to Yes (default), extract the attachments into files. If set to No, do not extract attachments.
`-MSG_PRF=<my_prefix>`	No	Prefix of the file that contains the body of the message. The default is MSG.
`-ATT_PRF=<my_prefix>`	No	Prefix of the files that contain the attachments. The original file names are kept.
`-USE_UCASE=<no|yes>`	No	If set to Yes, force the file names to uppercase. If set to No (default), keep the original letter case.
`-NOMAIL_ERROR=<no|yes>`	No	If set to Yes, generate an error if no mail matches the specified criteria. If set to No (default), do not generate an error when no mail corresponds to the specified criteria.
`-TIMEOUT=<timeout>`	No	Maximum waiting time in milliseconds. If this waiting time is reached, the command ends. The default value is 0, which means an infinite waiting time (as long as needed for the maximum number of messages specified with `-MAX_MSG` to be received).
`-POLLINT=<pollint>`	No	Searching interval in milliseconds to scan for new messages. The default value is 1000 (1 second).

Parameters	Mandatory	Description
`-MAX_MSG=<max_msg>`	No	Maximum number of messages to extract. If this number is reached, the command ends. The default value is 1.
`-SUBJECT=<subject_filter>`	No	Parameter used to filter the messages according to their subjects.
`-SENDER=<sender_filter>`	No	Parameter used to filter messages according to their sender.
`-TO=<to_filter>`	No	Parameter used to filter messages according to their addresses. This option can be repeated to create multiple filters.
`-CC=<cc_filter>`	No	Parameter used to filter messages according to their addresses in copy. This option can be repeated to create multiple filters.

Examples

Automatic reception of the mails of support with attachments detached in the folder `C:\support` on the system of the agent. Wait for all messages with a maximum waiting time of 10 seconds.

```
OdiReadMail -MAILHOST=mail.mymail.com -USER=myaccount -PASS=mypass
-KEEP=no -FOLDER=c:\support -TIMEOUT=0 -MAX_MSG=0
-SENDER=support@mycompany.com -EXTRACT_MSG=yes -MSG_PRF=TXT
-EXTRACT_ATT=yes
```

Wait indefinitely for 10 messages and check for new messages every minute.

```
OdiReadMail -MAILHOST=mail.mymail.com -USER=myaccount -PASS=mypass
-KEEP=no -FOLDER=c:\support -TIMEOUT=0 -MAX_MSG=10 -POLLINT=60000
-SENDER=support@mycompany.com -EXTRACT_MSG=yes -MSG_PRF=TXT
-EXTRACT_ATT=yes
```

OdiRefreshJournalCount

Use this command to refresh for a given journalizing subscriber the number of rows to consume for the given table list or CDC set. This refresh is performed on a logical schema and a given context, and may be limited.

Note: This command is suitable for journalized tables in simple or consistent mode and cannot be executed in a command line with `startcmd`.

Usage

```
OdiRefreshJournalCount -LSCHEMA=<logical_schema>
-SUBSCRIBER_NAME=<subscriber_name>
(-TABLE_NAME=<table_name> | -CDC_SET_NAME=<cdc set name>)
[-CONTEXT=<context>] [-MAX_JRN_DATE=<to_date>]
```

Parameters

Parameters	Mandatory	Description
`-LSCHEMA=<logical_schema>`	Yes	Logical schema containing the journalized tables.
`-TABLE_NAME=<table_name>`	Yes for working with simple CDC	Journalized table name, mask, or list to check. This parameter accepts three formats: ■ Table Name ■ Table Name Mask: This mask selects the tables to poll. The mask is specified using the SQL LIKE syntax: the `%` symbol replaces an unspecified number of characters and the _ symbol acts as a wildcard. ■ Table Names List: List of table names separated by commas. Masks as defined above are **not** allowed. Note that this option works only for tables in a model journalized in simple mode. This parameter cannot be used with `-CDC_SET_NAME`. It is mandatory if `-CDC_SET_NAME` is not set.
`-CDC_SET_NAME=<cdcSetName>`	Yes for working with consistent set CDC	Name of the CDC set to check. Note that this option works only for tables in a model journalized in consistent mode. This parameter cannot be used with `-TABLE_NAME`. It is mandatory if `-TABLE_NAME` is not set.
`-SUBSCRIBER_NAME=<subscriber_name>`	Yes	Name of the subscriber for which the count is refreshed.
`-CONTEXT=<context>`	No	Context in which the logical schema will be resolved. If no context is specified, the execution context is used.
`-MAX_JRN_DATE=<to_date>`	No	Date (and time) until which the journalizing events are taken into account.

Examples

Refresh for the `CUSTOMERS` table in the `SALES_APPLICATION` schema the count of modifications recorded for the `SALES_SYNC` subscriber. This datastore is journalized in simple mode.

```
OdiRefreshJournalCount -LSCHEMA=SALES_APPLICATION
-TABLE_NAME=CUSTOMERS -SUBSCRIBER_NAME=SALES_SYNC
```

Refresh for all tables from the `SALES` CDC set in the `SALES_APPLICATION` schema the count of modifications recorded for the `SALES_SYNC` subscriber. These datastores are journalized with consistent set CDC.

```
OdiRefreshJournalCount -LSCHEMA=SALES_APPLICATION
-SUBSCRIBER_NAME=SALES_SYNC -CDC_SET_NAME=SALES
```

OdiReinitializeSeq

Use this command to reinitialize an Oracle Data Integrator sequence.

Usage

```
OdiReinitializeSeq -SEQ_NAME=<sequence_name> -CONTEXT=<context>
-STD_POS=<position>
```

Parameters

Parameters	Mandatory	Description
`-SEQ_NAME=<sequence_name>`	Yes	Name of the sequence to reinitialize. It must be prefixed with `GLOBAL.` for a global sequence, or by `<project code>.` for a project sequence.
`-CONTEXT=<context>`	Yes	Context in which the sequence must be reinitialized.
`-STD_POS=<position>`	Yes	Position to which the sequence must be reinitialized.

Examples

Reset the global sequence `SEQ_I` to 0 for the `GLOBAL` context.

```
OdiReinitializeSeq -SEQ_NAME=GLOBAL.SEQ_I -CONTEXT=GLOBAL
-STD_POS=0
```

OdiRemoveTemporaryObjects

Use this command to remove temporary objects that could remain between executions. This is performed by executing the cleanup tasks for the sessions identified by the parameters specified in the tool parameters.

Usage

```
OdiRemoveTemporaryObjects [-COUNT=<session_number>] [-FROMDATE=<from_date>]
[-TODATE=<to_date>] [-CONTEXT_CODE=<context_code>]
[-AGENT_NAME=<agent_name>] [-USER_NAME=<user_name>]
[-NAME=<session_name>] [-ERRORS_ALLOWED=<number_of_errors_allowed>]
```

Parameters

Parameters	Mandatory	Description
`-COUNT=<session_number>`	No	Number of sessions for which to skip cleanup. The most recent number of sessions (`<session_number>`) is kept and the rest are cleaned up.
`-FROMDATE=<from_date>`	No	Start date for the cleanup, using the format yyyy/MM/dd hh:mm:ss. All sessions started after this date are cleaned up. If `-FROMDATE` is omitted, the cleanup starts with the oldest session.
`-TODATE=<to_date>`	No	End date for the cleanup, using the format yyyy/MM/dd hh:mm:ss. All sessions started before this date are cleaned up. If `-TODATE` is omitted, the cleanup starts with the most recent session.

Parameters	Mandatory	Description
`-CONTEXT_CODE=<context_code>`	No	Cleans up only those sessions executed in this context (`<context_code>`). If `-CONTEXT_CODE` is omitted, cleanup is performed on all contexts.
`-AGENT_NAME=<agent_name>`	No	Cleans up only those sessions executed by this agent (`<agent_name>`).
`-USER_NAME=<user_name>`	No	Cleans up only those sessions launched by this user (`<user_name>`).
`-NAME=<session_name>`	No	Session name.
`-ERRORS_ALLOWED=<number_of_errors_allowed>`	No	Number of errors allowed before the step ends with OK. If set to 0, the step ends with OK regardless of the number of errors encountered during the cleanup phase.

Examples

Remove the temporary objects by performing the cleanup tasks of all sessions executed between 2013/03/25 00:00:00 and 2013/08/31 21:59:00.

```
OdiRemoveTemporaryObjects "-FROMDATE=2013/03/25 00:00:00" "-TODATE=2013/08/31
21:59:00"
```

Remove the temporary objects by performing the cleanup tasks of all sessions executed in the `GLOBAL` context by the `Internal` agent.

```
OdiRemoveTemporaryObjects "-CONTEXT_CODE=GLOBAL" "-AGENT_NAME=Internal"
```

OdiRetrieveJournalData

Use this command to retrieve the journalized events for a given journalizing subscriber, a given table list or CDC set. The retrieval is performed specifically for the technology containing the tables. This retrieval is performed on a logical schema and a given context.

Note: This tool works for tables journalized using simple or consistent set modes and cannot be executed in a command line with `startcmd`.

Usage

```
OdiRetrieveJournalData -LSCHEMA=<logical_schema>
-SUBSCRIBER_NAME=<subscriber_name>
(-TABLE_NAME=<table_name> | -CDC_SET_NAME=<cdc_set_name>)
[-CONTEXT=<context>] [-MAX_JRN_DATE=<to_date>]
```

Parameters

Parameters	Mandatory	Description
`-LSCHEMA=<logical_schema>`	Yes	Logical schema containing the journalized tables.

Parameters	Mandatory	Description
`-TABLE_NAME=<table_name>`	No	Journalized table name, mask, or list to check. This parameter accepts three formats: ■ Table Name ■ Table Name Mask: This mask selects the tables to poll. The mask is specified using the SQL LIKE syntax: the `%` symbol replaces an unspecified number of characters and the _ symbol acts as a wildcard. ■ Table Names List: List of table names separated by commas. Masks as defined above are **not** allowed. Note that this option works only for tables in a model journalized in simple mode. This parameter cannot be used with `-CDC_SET_NAME`. It is mandatory if `-CDC_SET_NAME` is not set.
`-CDC_SET_NAME=<cdc_set_name>`	No	Name of the CDC set to update. Note that this option works only for tables in a model journalized in consistent mode. This parameter cannot be used with `-TABLE_NAME`. It is mandatory if `-TABLE_NAME` is not set.
`-SUBSCRIBER_NAME=<subscriber_name>`	Yes	Name of the subscriber for which the data is retrieved.
`-CONTEXT=<context>`	No	Context in which the logical schema will be resolved. If no context is specified, the execution context is used.
`-MAX_JRN_DATE=<to_date>`	No	Date (and time) until which the journalizing events are taken into account.

Examples

Retrieve for the `CUSTOMERS` table in the `SALES_APPLICATION` schema the journalizing events for the `SALES_SYNC` subscriber.

```
OdiRetrieveJournalData -LSCHEMA=SALES_APPLICATION
-TABLE_NAME=CUSTOMERS -SUBSCRIBER_NAME=SALES_SYNC
```

OdiReverseGetMetaData

Use this command to reverse-engineer metadata for the given model in the reverse tables using the JDBC driver capabilities. This command is typically preceded by OdiReverseResetTable and followed by OdiReverseSetMetaData.

Notes:

- This command uses the same technique as the standard reverse-engineering, and depends on the capabilities of the JDBC driver used.
- The use of this command is restricted to DEVELOPMENT type Repositories because the metadata is not available on EXECUTION type Repositories.

Usage

```
OdiReverseGetMetaData -MODEL=<model_id>
```

Parameters

Parameters	Mandatory	Description
`-MODEL=<model_id>`	Yes	Model to reverse-engineer.

Examples

Reverse the RKM's current model.

```
OdiReverseGetMetaData -MODEL=<%=odiRef.getModel("ID")%>
```

OdiReverseManageShortcut

Use this command to define how to handle shortcuts when they are reverse-engineered in a model.

Usage

```
OdiReverseManageShortcut "-MODEL=<model_id>" "-MODE=MATERIALIZING_MODE"
```

Parameters

Parameters	Mandatory	Description
`-MODEL=<model_id>`	Yes	Global identifier of the model to be reversed.
`-MODE=ALWAYS_MATERIALIZE|ALWAYS_SKIP|PROMPT`	Yes	This parameter accepts the following 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.

Example

Reverse model `125880` in `ALWAYS_MATERIALIZE` mode.

```
OdiReverseManageShortcut -MODEL=125880 -MODE=ALWAYS_MATERIALIZE
```

OdiReverseResetTable

Use this command to reset the content of reverse tables for a given model. This command is typically used at the beginning of a customized reverse-engineering process.

Usage

```
OdiReverseResetTable -MODEL=<model_id>
```

Parameters

Parameters	Mandatory	Description
`-MODEL=<model_id>`	Yes	Global identifier of the model to be reversed.

Examples

```
OdiReverseResetTable -MODEL=123001
```

OdiReverseSetMetaData

Use this command to integrate metadata from the reverse tables into the Repository for a given data model.

Usage

```
OdiReverseSetMetaData -MODEL=<model_id> [-USE_TABLE_NAME_FOR_UPDATE=<true|false>]
```

Parameters

Parameters	Mandatory	Description
`-MODEL=<model_id>`	Yes	Global identifier of the model to be reversed.
`-USE_TABLE_NAME_FOR_UPDATE=<true|false>`	No	■ If `true`, the `TABLE_NAME` is used as an update key on the target tables. ■ If `false` (default), the `RES_NAME` is used as the update key on the target tables.

Examples

Reverse model `125880`, using the `TABLE_NAME` as an update key on the target tables.

```
OdiReverseSetMetaData -MODEL=123001 -USE_TABLE_NAME_FOR_UPDATE=true
```

OdiSAPALEClient and OdiSAPALEClient3

Use this command to generate SAP Internal Documents (IDoc) from XML source files and transfer these IDocs using ALE (Application Link Enabling) to a remote tRFC server (SAP R/3 server).

Note: The OdiSAPALEClient tool supports SAP Java Connector 2.x. To use the SAP Java Connectors 3.x, use the OdiSAPALEClient3 tool.

Usage for OdiSAPALEClient

```
OdiSAPALEClient -USER=<sap_logon> -ENCODED_PASSWORD=<password>
-GATEWAYHOST=<gateway_host> -SYSTEMNR=<system_number>
-MESSAGESERVERHOST=<message_server> -R3NAME=<system_name>
```

```
-APPLICATIONSERVERSGROUP=<group_name>
[-DIR=<directory>] [-FILE=<file>] [-CASESENS=<yes|no>]
[-MOVEDIR=<target_directory>] [-DELETE=<yes|no>] [-POOL_KEY=<pool_key>]
[-LANGUAGE=<language>] [-CLIENT=<client>] [-MAX_CONNECTIONS=<n>]
[-TRACE=<no|yes>]
```

Usage for OdiSAPALEClient3

```
OdiSAPALEClient3 -USER=<sap_logon> -ENCODED_PASSWORD=<password>
-GATEWAYHOST=<gateway_host> -SYSTEMNR=<system_number>
-MESSAGESERVERHOST=<message_server> -R3NAME=<system_name>
-APPLICATIONSERVERSGROUP=<group_name>
[-DIR=<directory>] [-FILE=<file>] [-CASESENS=<yes|no>]
[-MOVEDIR=<target_directory>] [-DELETE=<yes|no>] [-POOL_KEY=<pool_key>]
[-LANGUAGE=<language>] [-CLIENT=<client>] [-MAX_CONNECTIONS=<n>]
[-TRACE=<no|yes>]
```

Parameters

Parameters	Mandatory	Description
`-USER=<sap_logon>`	Yes	SAP logon. This user may be a system user.
`-PASSWORD=<password>`	Deprecated	SAP logon password. This command is deprecated. Use `-ENCODED_PASSWORD` instead.
`-ENCODED_PASSWORD=<password>`	Yes	SAP logon password, encrypted. The OS command `encode <password>` can be used to encrypt this password.
`-GATEWAYHOST=<gateway_host>`	No	Gateway host, mandatory if `-MESSAGESERVERHOST` is not specified.
`-SYSTEMNR=<system_number>`	No	SAP system number, mandatory if `-GATEWAYHOST` is used. The SAP system number enables the SAP load balancing feature.
`-MESSAGESERVERHOST=<message_server>`	No	Message server host name, mandatory if `-GATEWAYHOST` is not specified. If `-GATEWAYHOST` and `-MESSAGESERVERHOST` are both specified, `-MESSAGESERVERHOST` is used.
`-R3NAME=<system_name>`	No	Name of the SAP system (r3name), mandatory if `-MESSAGESERVERHOST` is used.
`-APPLICATIONSERVERSGROUP=<group_name>`	No	Application servers group name, mandatory if `-MESSAGESERVERHOST` is used.
`-DIR=<directory>`	No	XML source file directory. This parameter is taken into account if `-FILE` is not specified. At least one of the `-DIR` or `-FILE` parameters must be specified.
`-FILE=<file>`	No	Name of the source XML file. If this parameter is omitted, all files in `-DIR` are processed. At least one of the `-DIR` or `-FILE` parameters must be specified.
`-CASESENS=<yes|no>`	No	Indicates if the source file names are case-sensitive. The default value is No.

Parameters	Mandatory	Description
`-MOVEDIR=<target_directory>`	No	If this parameter is specified, the source files are moved to this directory after being processed.
`-DELETE=<yes|no>`	No	Deletes the source files after their processing. The default value is Yes.
`-POOL_KEY=<pool_key>`	No	Name of the connection pool. The default value is `ODI`.
`-LANGUAGE=<language>`	No	Language code used for error messages. The default value is `EN`.
`-CLIENT=<client>`	No	Client identifier. The default value is `001`.
`-MAX_CONNECTIONS=<n>`	No	Maximum number of connections in the pool. The default value is `3`.
`-TRACE=<no|yes>`	No	The generated IDoc files are archived in the source file directory. If the source files are moved (`-MOVEDIR` parameter), the generated IDocs are also moved. The default value is No.

Examples

Process all files in the `/sap` directory and send them as IDocs to the SAP server. The original XML and generated files are stored in the `/log` directory after processing.

```
OdiSAPALEClient -USER=ODI -ENCODED_PASSWORD=xxx -SYSTEMNR=002
-GATEWAYHOST=GW001 -DIR=/sap -MOVEDIR=/log -TRACE=yes
```

OdiSAPALEServer and OdiSAPALEServer3

Use this command to start a tRFC listener to receive SAP IDocs transferred using ALE (Application Link Enabling). This listener transforms incoming IDocs into XML files in a given directory.

Note: The OdiSAPALEServer tool supports SAP Java Connector 2.x. To use the SAP Java Connectors 3.x, use the OdiSAPALEServer3 tool.

Usage of OdiSAPALEServer

```
OdiSAPALEServer -USER=<sap_logon> -ENCODED_PASSWORD=<password>
-GATEWAYHOST=<gateway_host> -SYSTEMNR=<system_number>
-GATEWAYNAME=<gateway_name> -PROGRAMID=<program_id> -DIR=<target_directory>
[-TIMEOUT=<n>] [-POOL_KEY=<pool_key>] [-LANGUAGE=<Language>]
[-CLIENT=<client>] [-MAX_CONNECTIONS=<n>]
[-INTERREQUESTTIMEOUT=<n>] [-MAXREQUEST=<n>] [-TRACE=<no|yes>]
```

Usage of OdiSAPALEServer3

```
OdiSAPALEServer3 -USER=<sap_logon> -ENCODED_PASSWORD=<password>
-GATEWAYHOST=<gateway_host> -SYSTEMNR=<system_number>
-GATEWAYNAME=<gateway_name> -PROGRAMID=<program_id> -DIR=<target_directory>
[-TIMEOUT=<n>] [-POOL_KEY=<pool_key>] [-LANGUAGE=<Language>]
[-CLIENT=<client>] [-MAX_CONNECTIONS=<n>]
[-INTERREQUESTTIMEOUT=<n>] [-MAXREQUEST=<n>] [-TRACE=<no|yes>]
```

Parameters

Parameters	Mandatory	Description
`-USER=<UserName>`	Yes	SAP logon. This user may be a system user.
`-ENCODED_PASSWORD=<password>`	Yes	SAP logon password, encrypted. The system command `encode <password>` can be used to encrypt this password.
`-GATEWAYHOST=<gateway_host>`	Yes	Gateway host.
`-SYSTEMNR=<system_number>`	Yes	SAP system number.
`-GATEWAYNAME=<gateway_name>`	Yes	Gateway name.
`-PROGRAMID=<program_id>`	Yes	The program ID. External name used by the tRFC server.
`-DIR=<target_directory>`	Yes	Directory in which the target XML files are stored. These files are named `<IDOC Number>.xml`, and are located in subdirectories named after the IDoc type. The default is `./FromSAP`.
`-POOL_KEY=<pool_key>`	Yes	Name of the connection pool. The default value is `ODI`.
`-LANG=<language>`	Yes	Language code used for error messages. The default value is `EN`.
`-CLIENT=<client>`	Yes	SAP client identifier. The default value is `001`.
`-TIMEOUT=<n>`	No	Life span in milliseconds for the server. At the end of this period the server stops automatically. If this timeout is set to 0, the server life span is infinite. The default value is 0.
`-MAX_CONNECTIONS=<n>`	Yes	Maximum number of connections allowed for the pool of connections. The default value is 3.
`-INTERREQUESTTIMEOUT=<n>`	No	If no IDOC is received during an interval of `n` milliseconds, the listener stops. If this timeout is set to 0, the timeout is infinite. The default value is 0.
`-MAXREQUEST=<n>`	No	Maximum number of requests after which the listener stops. If this parameter is set to 0, the server expects an infinite number of requests. The default value is 0. Note: If `-TIMEOUT`, `-INTERREQUESTTIMEOUT`, and `-MAXREQUEST` are set to 0 or left empty, then `-MAXREQUEST` automatically takes the value 1.
`-TRACE=<no|yes>`	No	Activate the debug trace. The default value is No.

Examples

Wait for 2 IDoc files and generate the target XML files in the `/temp` directory.

```
OdiSAPALEServer -POOL_KEY=ODI -MAX_CONNECTIONS=3 -CLIENT=001
-USER=ODI -ENCODED_PASSWORD=xxx -LANGUAGE=EN
-GATEWAYHOST=SAP001 -SYSTEMNR=002 -GATEWAYNAME=GW001
-PROGRAMID=ODI01 -DIR=/tmp -MAXREQUEST=2
```

OdiScpGet

Use this command to download a file from an SSH server.

Usage

```
OdiScpGet -HOST=<ssh server host name> -USER=<ssh user>
[-PASSWORD=<ssh user password>] -REMOTE_DIR=<remote dir on ssh host>
[-REMOTE_FILE=<file name under the REMOTE_DIR>] -LOCAL_DIR=<local dir>
[-LOCAL_FILE=<file name under the LOCAL_DIR>] [-PASSIVE_MODE=<yes|no>]
[-TIMEOUT=<time in seconds>]
[-IDENTITY_FILE=<full path to the private key file of the user>]
[-KNOWNHOSTS_FILE=<full path to known hosts file>] [COMPRESSION=<yes|no>]
[-STRICT_HOSTKEY_CHECKING=<yes|no>] [-PROXY_HOST=<proxy server host name>]
[-PROXY_PORT=<proxy server port>] [-PROXY_TYPE=<HTTP|SOCKS5>]
```

Parameters

Parameters	Mandatory	Description
`-HOST=<ssh server host name>`	Yes	Host name of the SSH server.
`-USER=<ssh user>`	Yes	User on the SSH server.
`-PASSWORD=<ssh user password>`	No	The password of the SSH user or the passphrase of the password-protected identity file. If the `-IDENTITY_FILE` argument is provided, this value is used as the passphrase for the password-protected private key file. If public key authentication fails, it falls back to the normal user password authentication.
`-REMOTE_DIR=<dir on remote SSH>`	Yes	Directory path on the remote SSH host.
`-REMOTE_FILE=<file name under -REMOTE DIR>`	No	File name under the directory specified in the `-REMOTE_DIR` argument. Note that all subdirectories matching the remote file name will also be transferred to the local folder. If this argument is missing, the file is copied with the `-LOCAL_FILE` file name. If `-LOCAL_FILE` is also missing, the `-LOCAL_DIR` is copied recursively to the `-REMOTE_DIR`.
`-LOCAL_DIR=<local dir path>`	Yes	Directory path on the local machine.

Parameters	Mandatory	Description
`-LOCAL_FILE=<local file>`	No	File name under the directory specified in the `-LOCAL_DIR` argument. If this argument is missing, all files and directories under `-LOCAL_DIR` are copied recursively to the `-REMOTE_DIR`. To filter the files to be copied use `*` to specify the generic characters. Examples: ▪ `*.log` (all files with the `log` extension) ▪ `arch_*.lst` (all files starting with `arch_` and with the extension `lst`)
`-IDENTITY_FILE=<full path to the private key file of the user>`	No	Private key file of the local user. If this argument is specified, public key authentication is performed. The `-PASSWORD` argument is used as the password for the password-protected private key file. If authentication fails, it falls back to normal user password authentication.
`-KNOWNHOSTS_FILE=<full path to the known hosts file on the local machine>`	No	Full path to the known hosts file on the local machine. The known hosts file contains the host keys of all remote machines the user trusts. If this argument is missing, the `<user home dir>/.ssh/known_hosts` file is used as the known hosts file if it exists.
`-COMPRESSION=<yes|no>`	No	If set to Yes, data compression is used. The default value is No.
`-STRICT_HOSTKEY_CHECKING=<yes|no>`	No	If set to Yes (default), strict host key checking is performed and authentication fails if the remote SSH host key is not present in the known hosts file specified in `-KNOWNHOSTS_FILE`.
`-PROXY_HOST=<proxy server host name>`	No	Host name of the proxy server to be used for the connection.
`-PROXY_PORT=<proxy server port>`	No	Port number of the proxy server.
`-PROXY_TYPE=<HTTP|SOCKS5>`	No	Type of proxy server you are connecting to, HTTP or SOCKS5.
`-TIMEOUT=<time in seconds>`	No	Time in seconds after which the socket connection times out.

Examples

Copy the remote directory `/test_copy555` on the SSH server recursively to the local directory `C:\temp\test_copy`.

```
OdiScpGet -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password>
 -LOCAL_DIR=C:\temp\test_copy -REMOTE_DIR=/test_copy555
```

Copy all files matching the `Sales*.txt` pattern under the remote directory `/` on the SSH server to the local directory `C:\temp\`.

```
OdiScpGet -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password>
 -LOCAL_DIR=C:\temp -REMOTE_FILE=Sales*.txt -REMOTE_DIR=/
```

Copy the `Sales1.txt` file under the remote directory `/` on the SSH server to the local directory `C:\temp\` as a `Sample1.txt` file.

```
OdiScpGet -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password>
 -REMOTE_DIR=/ REMOTE_FILE=Sales1.txt -LOCAL_DIR=C:\temp
-LOCAL_FILE=Sample1.txt
```

Copy the `Sales1.txt` file under the remote directory `/` on the SSH server to the local directory `C:\temp\` as a `Sample1.txt` file. Public key authentication is performed by providing the path to the identity file and the path to the known hosts file.

```
OdiScpGet -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password>
-REMOTE_DIR=/ -REMOTE_FILE=Sales1.txt -LOCAL_DIR=C:\temp
-LOCAL_FILE=Sample1.txt -IDENTITY_FILE=C:\Documents and Settings\username\.ssh\id_
dsa -KNOWNHOSTS_FILE=C:\Documents and Settings\username\.ssh\known_hosts
```

Copy the `Sales1.txt` file under the remote directory `/` on the SSH server to the local directory `C:\temp\` as a `Sample1.txt` file. Public key authentication is performed by providing the path to the identity file. All hosts are trusted by passing the No value to the `-STRICT_HOSTKEY_CHECKING` parameter.

```
OdiScpGet -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password>
-REMOTE_DIR=/ -REMOTE_FILE=Sales1.txt -LOCAL_DIR=C:\temp -LOCAL_FILE=Sample1.txt
-IDENTITY_FILE=C:\Documents and Settings\username\.ssh\id_dsa
-STRICT_HOSTKEY_CHECKING=NO
```

OdiScpPut

Use this command to upload a file to an SSH server.

Usage

```
OdiScpPut -HOST=<SSH server host name> -USER=<SSH user>
[-PASSWORD=<SSH user password>] -LOCAL_DIR=<local dir>
[-LOCAL_FILE=<file name under the LOCAL_DIR>] -REMOTE_DIR=<remote dir on ssh host>
[-REMOTE_FILE=<file name under the REMOTE_DIR>] [-PASSIVE_MODE=<yes|no>]
[-TIMEOUT=<time in seconds>]
[-IDENTITY_FILE=<full path to the private key file of the user>]
[-KNOWNHOSTS_FILE=<full path to known hosts file>] [-COMPRESSION=<yes|no>]
[-STRICT_HOSTKEY_CHECKING=<yes|no>] [<-PROXY_HOST=<proxy server host name>]
[-PROXY_PORT=<proxy server port>] [-PROXY_TYPE=<HTTP|SOCKS5>]
```

Parameters

Parameters	Mandatory	Description
`-HOST=<host name of the SSH server>`	Yes	Host name of the SSH server.
`-USER=<host name of the SSH user>`	Yes	User on the SSH server.
`-PASSWORD=<password of the SSH user>`	No	Password of the SSH user or the passphrase of the password-protected identity file. If the `-IDENTITY_FILE` argument is provided, this value is used as the passphrase for the password-protected private key file. If public key authentication fails, it falls back to the normal user password authentication.
`-REMOTE_DIR=<dir on remote SSH`	Yes	Directory path on the remote SSH host.

Parameters	Mandatory	Description
-REMOTE_FILE=<file name under -REMOTE DIR>	No	File name under the directory specified in the -REMOTE_DIR argument. If this argument is missing, the file is copied with the -LOCAL_FILE file name. If the -LOCAL_FILE argument is also missing, the -LOCAL_DIR is copied recursively to the -REMOTE_DIR.
-LOCAL_DIR=<local dir path>	Yes	Directory path on the local machine.
-LOCAL_FILE=<local file>	No	File name under the directory specified in the -LOCAL_DIR argument. If this argument is missing, all files and directories under the -LOCAL_DIR are copied recursively to the -REMOTE_DIR. To filter the files to be copied use * to specify the generic characters. Examples: ■ *.log (all files with the log extension) ■ arch_*.lst (all files starting with arch_ and with the extension lst)
-IDENTITY_FILE=<full path to the private key file of the user>	No	Private key file of the local user. If this argument is specified, public key authentication is performed. The -PASSWORD argument is used as the password for the password-protected private key file. If authentication fails, it falls back to normal user password authentication.
-KNOWNHOSTS_FILE=<full path to the known hosts file on the local machine>	No	Full path to the known hosts file on the local machine. The known hosts file contains the host keys of all remote machines the user trusts. If this argument is missing, the <user home dir>/.ssh/known_hosts file is used as the known hosts file if it exists.
-COMPRESSION=<yes|no>	No	If set to Yes, data compression is used. The default value is No.
-STRICT_HOSTKEY_CHECKING=<yes|no>	No	If set to Yes (default), strict host key checking is performed and authentication fails if the remote SSH host key is not present in the known hosts file specified in -KNOWNHOSTS_FILE.
-PROXY_HOST=<proxy server host name>	No	Host name of the proxy server to be used for the connection.
-PROXY_PORT=<proxy server port>	No	Port number of the proxy server.
-PROXY_TYPE=<HTTP|SOCKS5>	No	Type of proxy server you are connecting to, HTTP or SOCKS5.
-TIMEOUT=<timeout value>	No	Time in seconds after which the socket connection times out.

Examples

Copy the local directory `C:\temp\test_copy` recursively to the remote directory `/test_copy555` on the SSH server.

```
OdiScpPut -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password> -LOCAL_
DIR=C:\temp\test_copy -REMOTE_DIR=/test_copy555
```

Copy all files matching the `Sales*.txt` pattern under the local directory `C:\temp\` to the remote directory `/` on the SSH server.

```
OdiScpPut -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password> -LOCAL_
DIR=C:\temp -LOCAL_FILE=Sales*.txt -REMOTE_DIR=/
```

Copy the `Sales1.txt` file under the local directory `C:\temp\` to the remote directory `/` on the SSH server as a `Sample1.txt` file.

```
OdiScpPut -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password>
-LOCAL_DIR=C:\temp -LOCAL_FILE=Sales1.txt -REMOTE_DIR=/ -REMOTE_FILE=Sample1.txt
```

Copy the `Sales1.txt` file under the local directory `C:\temp\` to the remote directory `/` on the SSH server as a `Sample1.txt` file. Public key authentication is performed by providing the path to the identity file and the path to the known hosts file.

```
OdiScpPut -HOST=machine.oracle.com -USER=test_ftp
-PASSWORD=<password> -LOCAL_DIR=C:\temp -LOCAL_FILE=Sales1.txt
-REMOTE_DIR=/ -REMOTE_FILE=Sample1.txt
-IDENTITY_FILE=C:\Documents and Settings\username\.ssh\id_dsa
-KNOWNHOSTS_FILE=C:\Documents and Settings\username\.ssh\known_hosts
```

Copy the `Sales1.txt` file under the local directory `C:\temp\` to the remote directory `/` on the SSH server as a `Sample1.txt` file. Public key authentication is performed by providing the path to the identity file. All hosts are trusted by passing the No value to the `-STRICT_HOSTKEY_CHECKING` parameter.

```
OdiScpPut -HOST=machine.oracle.com -USER=test_ftp
-PASSWORD=<password> -LOCAL_DIR=C:\temp -LOCAL_FILE=Sales1.txt
-REMOTE_DIR=/ -REMOTE_FILE=Sample1.txt
-IDENTITY_FILE=C:\Documents and Settings\username\.ssh\id_dsa
-STRICT_HOSTKEY_CHECKING=NO
```

OdiSendMail

Use this command to send an email to an SMTP server.

Usage

```
OdiSendMail -MAILHOST=<mail_host> -FROM=<from_user> -TO=<address_list>
[-CC=<address_list>] [-BCC=<address_list>] [-SUBJECT=<subject>]
[-ATTACH=<file_path>]* [-MSGBODY=<message_body> | CR/LF<message_body>]
```

Parameters

Parameters	Mandatory	Description
`-MAILHOST=<mail_host>`	Yes	IP address of the SMTP server.

Parameters	Mandatory	Description
`-FROM=<from_user>`	Yes	Address of the sender of the message. Example: `support@mycompany.com` To send the external name of the sender, the following notation can be used: `"-FROM=Support center <support@mycompany.com>"`
`-TO=<address_list>`	Yes	List of email addresses of the recipients, separated by commas. Example: `"-TO=sales@mycompany.com, support@mycompany.com"`
`-CC=<address_list>`	No	List of e-mail addresses of the CC-ed recipients, separated by commas. Example: `"-CC=info@mycompany.com"`
`-BCC=<address_list>`	No	List of email-addresses of the BCC-ed recipients, separated by commas. Example: `"-BCC=manager@mycompany.com"`
`-SUBJECT=<subject>`	No	Object (subject) of the message.

`-ATTACH=<file_path>`	No	Path of the file to join to the message, relative to the execution agent. To join several files, repeat `-ATTACH`.
		Example: Attach the files `.profile` and `.cshrc` to the mail:
		`-ATTACH=/home/usr/.profile`
		`-ATTACH=/home/usr/.cshrc`

`CR/LF <message_body>` or `-MSGBODY=<message_body>`	No	Message body (text). This text can be typed on the line following the OdiSendMail command (a carriage return - `CR/LF` - indicates the beginning of the mail body), or can be defined with the `-MSGBODY` parameter. The `-MSGBODY` parameter should be used when calling this Oracle Data Integrator command from an OS command line.

Examples

```
OdiSendMail -MAILHOST=mail.mymail.com "-FROM=Application Oracle Data
Integrator<odi@mymail.com>" -TO=admin@mymail.com "-SUBJECT=Execution OK"
-ATTACH=C:\log\job.log -ATTACH=C:\log\job.bad
Hello Administrator !
Your process finished successfully. Attached are your files.
Have a nice day!
Oracle Data Integrator.
```

OdiSftp

Use this command to connect to an SSH server with an enabled SFTP subsystem and perform standard FTP commands on the remote system. Trace from the script is recorded against the task representing the OdiSftp step in Operator Navigator.

Usage

```
OdiSftp -HOST=<ssh server host name> -USER=<ssh user>
[-PASSWORD=<ssh user password>] -LOCAL_DIR=<local dir>
-REMOTE_DIR=<remote dir on ssh host> [-PASSIVE_MODE=<yes|no>]
[-TIMEOUT=<time in seconds>] [-IDENTITY_FILE=<full path to private key file of
user>] [-KNOWNHOSTS_FILE=<full path to known hosts file on local machine>]
[-COMPRESSION=<yes|no>] [-STRICT_HOSTKEY_CHECKING=<yes|no>]
[-PROXY_HOST=<proxy server host name>] [-PROXY_PORT=<proxy server port>]
 [-PROXY_TYPE=<HTTP|SOCKS5>] [STOP_ON_FTP_ERROR=<yes|no>]
-COMMAND=<command>
```

Parameters

Parameters	Mandatory	Description
`-HOST=<ssh server host name>`	Yes	Host name of the SSH server.
`-USER=<ssh user>`	Yes	User on the SSH server.
`-PASSWORD=<ssh user password>`	No	Password of the SSH user.
`-LOCAL_DIR=<local dir>`	Yes	Directory path on the local machine.
`-REMOTE_DIR=<remote dir on ssh host>`	Yes	Directory path on the remote SSH host.
`-TIMEOUT=<time in seconds>`	No	Time in seconds after which the socket connection times out.
`-IDENTITY_FILE=<full path to private key file of user>`	No	Private key file of the local user. If specified, public key authentication is performed. The `-PASSWORD` argument is used as the password for the password-protected private key file. If authentication fails, normal user password authentication is performed.
`-KNOWNHOSTS_FILE=<full path to known hosts file on local machine>`	No	Full path to the known hosts file on the local machine. The known hosts file contains host keys for all remote machines trusted by the user. If this argument is missing, the `<user home dir>/.ssh/known_hosts` file is used as the known hosts file if it exists.
`-COMPRESSION=<yes|no>`	No	If set to Yes, data compression is used. The default value is No.
`-STRICT_HOSTKEY_ CHECKING=<yes|no>`	No	If set to Yes (default), strict host key checking is performed and authentication fails if the remote SSH host key is not present in the known hosts file specified in `-KNOWNHOSTS_ FILE`.
`-PROXY_HOST=<proxy server host name>`	No	Host name of the proxy server to be used for the connection.
`-PROXY_PORT=<proxy server port>`	No	Port number of the proxy server.
`-PROXY_TYPE<HTTP|SOCKS5>`	No	Type of proxy server you are connecting to, HTTP or SOCKS5.

Parameters	Mandatory	Description
`STOP_ON_FTP_ERROR=<yes|no>`	No	If set to Yes (default), the step stops with an Error status if an error occurs rather than running to completion.
`-COMMAND=<command>`	Yes	Raw FTP command to execute. For a multiline command, pass the whole command as raw text after the OdiSftp line without the `-COMMAND` parameter. Supported commands: `APPE`, `CDUP`, `CWD`, `DELE`, `LIST`, `MKD`, `NLST`, `PWD`, `QUIT`, `RETR`, `RMD`, `RNFR`, `RNTO`, `SIZE`, `STOR`

Examples

Execute a script on a remote host that changes directory into a directory, deletes a file from the directory, changes directory into the parent directory, and removes the directory.

```
OdiSftp -HOST=machine.oracle.com -USER=odiftpuser -PASSWORD=<password>
-LOCAL_DIR=/tmp -REMOTE_DIR=/tmp -STOP_ON_FTP_ERROR=No
CWD /tmp/ftpToolDir1
DELE ftpToolFile
CDUP
RMD ftpToolDir1
```

OdiSftpGet

Use this command to download a file from an SSH server with an enabled SFTP subsystem.

Usage

```
OdiSftpGet -HOST=<ssh server host name> -USER=<ssh user>
[-PASSWORD=<ssh user password>] -REMOTE_DIR=<remote dir on ssh host>
[-REMOTE_FILE=<file name under REMOTE_DIR>] -LOCAL_DIR=<local dir>
[-LOCAL_FILE=<file name under LOCAL_DIR>] [-PASSIVE_MODE=<yes|no>]
[-TIMEOUT=<time in seconds>]
[-IDENTITY_FILE=<full path to private key file of user>]
[-KNOWNHOSTS_FILE=<full path to known hosts file on local machine>]
[-COMPRESSION=<yes|no>] [-STRICT_HOSTKEY_CHECKING=<yes|no>]
[-PROXY_HOST=<proxy server host name>] [-PROXY_PORT=<proxy server port>]
[-PROXY_TYPE=<HTTP|SOCKS5>]
```

Parameters

Parameters	Mandatory	Description
`-HOST=<ssh server host name>`	Yes	Host name of the SSH server. You can add the port number to the host name by prefixing it with a colon (`:`). For example: `machine.oracle.com:25` If no port is specified, port 22 is used by default.

Parameters	Mandatory	Description
`-USER=<ssh user>`	Yes	User on the SSH server.
`-PASSWORD=<ssh user password>`	No	Password of the SSH user.
`-REMOTE_DIR=<remote dir on ssh host>`	Yes	Directory path on the remote SSH host.
`-REMOTE_FILE=<file name under -REMOTE DIR>`	No	File name under the directory specified in the `-REMOTE_DIR` argument. If this argument is missing, the file is copied with the `-LOCAL_FILE` file name. If the `-LOCAL_FILE` argument is also missing, the `-LOCAL_DIR` is copied recursively to the `-REMOTE_DIR`.
`-LOCAL_DIR=<local dir>`	Yes	Directory path on the local machine.
`-LOCAL_FILE=<file name under LOCAL_DIR>`	No	File name under the directory specified in the `-LOCAL_DIR` argument. If this argument is missing, all files and directories under the `-LOCAL_DIR` are copied recursively to the `-REMOTE_DIR`. To filter the files to be copied, use `*` to specify the generic characters. Examples: ■ `*.log` (all files with the `log` extension) ■ `arch_*.lst` (all files starting with `arch_` and with the extension `lst`)
`-IDENTITY_FILE=<full path to private key file of user>`	No	Private key file of the local user. If this argument is specified, public key authentication is performed. The `-PASSWORD` argument is used as the password for the password-protected private key file. If authentication fails, it falls back to normal user password authentication.
`-KNOWNHOSTS_FILE=<full path to known hosts file on local machine>`	No	The full path to the known hosts file on the local machine. The known hosts file contains the host keys of all remote machines the user trusts. If this argument is missing, the `<user home dir>/.ssh/known_hosts` file is used as the known hosts file if it exists.
`-COMPRESSION=<yes|no>`	No	If set to Yes, data compression is used. The default value is No.
`-STRICT_HOSTKEY_CHECKING=<yes|no>`	No	If set to Yes (default), strict host key checking is performed and authentication fails if the remote SSH host key is not present in the known hosts file specified in `-KNOWNHOSTS_FILE`.
`-PROXY_HOST=<proxy server host name>`	No	Host name of the proxy server to be used for the connection.
`-PROXY_PORT=<proxy server port>`	No	Port number of the proxy server.
`-PROXY_TYPE=<HTTP|SOCKS5>`	No	Type of proxy server you are connecting to, HTTP or SOCKS5.

Parameters	Mandatory	Description
-TIMEOUT=<time in seconds>	No	Time in seconds after which the socket connection times out.

Examples

Copy the remote directory /test_copy555 on the SSH server recursively to the local directory C:\temp\test_copy.

```
OdiSftpGet -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password> -LOCAL_
DIR=C:\temp\test_copy -REMOTE_DIR=/test_copy555
```

Copy all files matching the Sales*.txt pattern under the remote directory / on the SSH server to the local directory C:\temp\.

```
OdiSftpGet -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password> -LOCAL_
DIR=C:\temp -REMOTE_FILE=Sales*.txt -REMOTE_DIR=/
```

Copy the Sales1.txt file under the remote directory / on the SSH server to the local directory C:\temp\ as a Sample1.txt file.

```
OdiSftpGet -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password> -REMOTE_
DIR=/ -LOCAL_FILE=Sales1.txt -LOCAL_DIR=C:\temp -LOCAL_FILE=Sample1.txt
```

Copy the Sales1.txt file under the remote directory / on the SSH server to the local directory C:\temp\ as a Sample1.txt file. Public key authentication is performed by providing the path to the identity file and the path to the known hosts file.

```
OdiSftpGet -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password>
-REMOTE_DIR=/ -REMOTE_FILE=Sales1.txt -LOCAL_DIR=C:\temp -LOCAL_FILE=Sample1.txt
-IDENTITY_FILE=C:\Documents and Settings\username\.ssh\id_dsa
-KNOWNHOSTS_FILE=C:\Documents and Settings\username\.ssh\known_hosts
```

Copy the Sales1.txt file under the remote directory / on the SSH server to the local directory C:\temp\ as a Sample1.txt file. Public key authentication is performed by providing the path to the identity file. All hosts are trusted by passing the No value to the -STRICT_HOSTKEY_CHECKING parameter.

```
OdiSftpGet -HOST=dev3 -USER=test_ftp -PASSWORD=<password>
-REMOTE_DIR=/ -REMOTE_FILE=Sales1.txt -LOCAL_DIR=C:\temp -LOCAL_FILE=Sample1.txt
-IDENTITY_FILE=C:\Documents and Settings\username\.ssh\id_dsa
-STRICT_HOSTKEY_CHECKING=NO
```

OdiSftpPut

Use this command to upload a file to an SSH server with the SFTP subsystem enabled.

Usage

```
OdiSftpPut -HOST=<ssh server host name> -USER=<ssh user>
[-PASSWORD=<ssh user password>] -LOCAL_DIR=<local dir>
[-LOCAL_FILE=<file name under LOCAL_DIR>] -REMOTE_DIR=<remote dir on ssh host>
[-REMOTE_FILE=<file name under REMOTE_DIR>] [-PASSIVE_MODE=<yes|no>]
[-TIMEOUT=<time in seconds>]
[-IDENTITY_FILE=<full path to private key file of user>]
[-KNOWNHOSTS_FILE=<full path to known hosts file on local machine>]
[-COMPRESSION=<yes|no>] [-STRICT_HOSTKEY_CHECKING=<yes|no>]
[-PROXY_HOST=<proxy server host name>] [-PROXY_PORT=<proxy server port>]
[-PROXY_TYPE=<HTTP|SOCKS5>]
```

Parameters

Parameters	Mandatory	Description
`-HOST=<ssh server host name>`	Yes	Host name of the SSH server. You can add the port number to the host name by prefixing it with a colon (`:`). For example: `machine.oracle.com:25` If no port is specified, port 22 is used by default.
`-USER=<ssh user>`	Yes	User on the SSH server.
`-PASSWORD=<ssh user password>`	No	Password of the SSH user or the passphrase of the password-protected identity file. If the `-IDENTITY_FILE` argument is provided, this value is used as the passphrase for the password-protected private key file. If public key authentication fails, it falls back to normal user password authentication.
`-REMOTE_DIR=<remote dir on ssh host`	Yes	Directory path on the remote SSH host.
`-REMOTE_FILE=<file name under -REMOTE DIR>`	No	File name under the directory specified in the `-REMOTE_DIR` argument. If this argument is missing, the file is copied with the `-LOCAL_FILE` file name. If the `-LOCAL_FILE` argument is also missing, the `-LOCAL_DIR` is copied recursively to the `-REMOTE_DIR`.
`-LOCAL_DIR=<local dir>`	Yes	Directory path on the local machine.
`-LOCAL_FILE=<file name under LOCAL_ DIR>`	No	File name under the directory specified in the `-LOCAL_DIR` argument. If this argument is missing, all files and directories under the `-LOCAL_DIR` is copied recursively to the `-REMOTE_DIR`. To filter the files to be copied, use `*` to specify the generic characters. Examples: ■ `*.log` (all files with the `log` extension) ■ `arch_*.lst` (all files starting with `arch_` and with the extension `lst`)
`-IDENTITY_FILE=<full path to private key file of user>`	No	Private key file of the local user. If this argument is specified, public key authentication is performed. The `-PASSWORD` argument is used as the password for the password-protected private key file. If authentication fails, it falls back to normal user password authentication.
`-KNOWNHOSTS_FILE=<full path to known hosts file on local machine>`	No	Full path to the known hosts file on the local machine. The known hosts file contains the host keys of all remote machines the user trusts. If this argument is missing, the `<user home dir>/.ssh/known_hosts` file is used as the known hosts file if it exists.

Parameters	Mandatory	Description
`-COMPRESSION=<yes|no>`	No	If set to Yes, data compression is used. The default value is No.
`-STRICT_HOSTKEY_CHECKING=<yes|no>`	No	If set to Yes (default), strict host key checking is performed and authentication fails if the remote SSH host key is not present in the known hosts file specified in `-KNOWNHOSTS_FILE`.
`-PROXY_HOST=<proxy server host name>`	No	Host name of the proxy server to be used for the connection.
`-PROXY_PORT=<proxy server port>`	No	Port number of the proxy server.
`-PROXY_TYPE=<HTTP|SOCKS5>`	No	Type of proxy server you are connecting to, HTTP or SOCKS5.
`-TIMEOUT=<time in seconds>`	No	Time in seconds after which the socket connection times out.

Examples

Copy the local directory `C:\temp\test_copy` recursively to the remote directory `/test_copy555` on the SSH server.

```
OdiSftpPut -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password> -LOCAL_
DIR=C:\temp\test_copy -REMOTE_DIR=/test_copy555
```

Copy all files matching the `Sales*.txt` pattern under the local directory `C:\temp\` to the remote directory `/` on the SSH server.

```
OdiSftpPut -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password> -LOCAL_
DIR=C:\temp -LOCAL_FILE=Sales*.txt -REMOTE_DIR=/
```

Copy the `Sales1.txt` file under the local directory `C:\temp\` to the remote directory `/` on the SSH server as a `Sample1.txt` file.

```
OdiSftpPut -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password> -LOCAL_
DIR=C:\temp -LOCAL_FILE=Sales1.txt -REMOTE_DIR=/Sample1.txt
```

Copy the `Sales1.txt` file under the local directory `C:\temp\` to the remote directory `/` on the SSH server as a `Sample1.txt` file. Public key authentication is performed by providing the path to the identity file and the path to the known hosts file.

```
OdiSftpPut -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password>
-LOCAL_DIR=C:\temp -LOCAL_FILE=Sales1.txt -REMOTE_DIR=/Sample1.txt
-IDENTITY_FILE=C:\Documents and Settings\username\.ssh\id_dsa
-KNOWNHOSTS_FILE=C:\Documents and Settings\username\.ssh\known_hosts
```

Copy the `Sales1.txt` file under the local directory `C:\temp\` to the remote directory `/` on the SSH server as a `Sample1.txt` file. Public key authentication is performed by providing the path to the identity file. All hosts are trusted by passing the No value to the `-STRICT_HOSTKEY_CHECKING` parameter.

```
OdiSftpPut -HOST=machine.oracle.com -USER=test_ftp -PASSWORD=<password>
-LOCAL_DIR=C:\temp -LOCAL_FILE=Sales1.txt -REMOTE_DIR=/Sample1.txt
-IDENTITY_FILE=C:\Documents and Settings\username\.ssh\id_dsa
-STRICT_HOSTKEY_CHECKING=NO
```

OdiSleep

Use this command to wait for `<delay>` milliseconds.

Usage

```
OdiSleep -DELAY=<delay>
```

Parameters

Parameters	Mandatory	Description
`-DELAY=<delay>`	Yes	Number of milliseconds to wait

Examples

```
OdiSleep -DELAY=5000
```

OdiSqlUnload

Use this command to write the result of a SQL query to a file.

This command executes the SQL query `<sql_query>` on the data server whose connection parameters are provided by `<driver>`, `<url>`, `<user>`, and `<encoded_pass>`. The resulting resultset is written to `<file_name>`.

Usage

```
OdiSqlUnload -FILE=<file_name> -DRIVER=<driver> -URL=<url> -USER=<user>
-PASS=<password> [-FILE_FORMAT=<file_format>] [-FIELD_SEP=<field_sep> |
-XFIELD_SEP=<field_sep>] [-ROW_SEP=<row_sep> | -XROW_SEP=<row_sep>]
[-DATE_FORMAT=<date_format>] [-CHARSET_ENCODING=<encoding>]
[-XML_CHARSET_ENCODING=<encoding>] [-FETCH_SIZE=<array_fetch_size>]
( CR/LF <sql_query> | -QUERY=<sql_query> | -QUERY_FILE=<sql_query_file> )
```

Parameters

Parameters	Mandatory	Description
`-FILE=<file_name>`	Yes	Full path to the output file, relative to the execution agent.
`-DRIVER=<driver>`	Yes	Name of the JDBC driver used to connect to the data server.
`-URL=<url>`	Yes	JDBC URL to the data server.
`-USER=<user>`	Yes	Login of the user on the data server that will be used to run the SQL query.
`-PASS=<password>`	Yes	Encrypted password for the login to the data server. This password can be encrypted with the system command `encode <clear_text_ password>`. Note that `agent(.bat` or `.sh)` is located in the `/bin` subdirectory of your Oracle Data Integrator installation directory.

Parameters	Mandatory	Description

`-FILE_FORMAT=<file_format>`	No	Specifies the file format with one of the following three values:
		■ `fixed`: Fixed size recording
		■ `variable`: Variable size recording
		■ `xml`: XML file
		If `<file_format>` is not specified, the format defaults to `variable`.
		If `<file_format>` is `xml`, the XML nodes generated have the following structure:
		`<TABLE>`
		`<ROW>`
		`<column_name>![CDATA[VALUE]]</column_name>`
		`<column_name>![CDATA[VALUE]]</column_name>`
		...
		`</ROW>`
		....
		`</TABLE>`

`-FIELD_SEP=<field_sep>`	No	Field separator character in ASCII format if `-FILE_FORMAT=variable`. The default `<field_sep>` is a tab character.
`-XFIELD_SEP=<field_sep>`	No	Field separator character in hexadecimal format if `-FILE_FORMAT=variable`. The default `<field_sep>` is a tab character.
`-ROW_SEP=<row_sep>`	No	Record separator character in ASCII format. The default `<row_sep>` is a Windows carriage return. For instance, the following values can be used: ■ UNIX: `-ROW_SEP=\n` ■ Windows: `-ROW_SEP=\r\n`
`-XROW_SEP=<row_sep>`	No	Record separator character in hexadecimal format. Example: `0A`.
`-DATE_FORMAT=<date_format>`	No	Output format used for date datatypes. This date format is specified using the Java date and time format patterns. For a list of these patterns, see: http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html.
`-CHARSET_ENCODING=<encoding>`	No	Target file encoding. The default value is `ISO-8859-1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
`-XML_CHARSET_ENCODING=<encoding>`	No	Encoding specified in the XML file, in the tag `<?xml version="1.0" encoding="ISO-8859-1"?>`. The default value is `ISO-8859-1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
`-FETCH_SIZE=<array_fetch_size>`	No	Number of rows (records read) requested by Oracle Data Integrator in each communication with the data server.

Parameters	Mandatory	Description
`-CR/LF=<sql_query> | -QUERY=<sql_query> | -QUERY_FILE=<sql_query_file>`	Yes	SQL query to execute on the data server. The query must be a `SELECT` statement or a call to a stored procedure returning a valid recordset. This query can be entered on the line following the OdiSqlUnload command (a carriage return - `CR/LF` - indicates the beginning of the query). The query can be provided within the `-QUERY` parameter, or stored in a file specified with the `-QUERY_FILE` parameter. The `-QUERY` or `-QUERY_FILE` parameters must be used when calling this command from an OS command line.

Examples

Generate the file `C:\temp\clients.csv` separated by `;` containing the result of the query on the `Customers` table.

```
OdiSqlUnload -FILE=C:\temp\clients.csv -DRIVER=sun.jdbc.odbc.JdbcOdbcDriver
-URL=jdbc:odbc:NORTHWIND_ODBC -USER=sa
-PASS=NFNEKKNGGJHAHBHDHEHJDBGBGFDGGH -FIELD_SEP=;
"-DATE_FORMAT=dd/MM/yyyy hh:mm:ss"

select cust_id, cust_name, cust_creation_date from Northwind.dbo.Customers
```

OdiStartLoadPlan

Use this command to start a Load Plan.

The `-SYNC` parameter starts a load plan in synchronous or asynchronous mode. In synchronous mode, the tool ends with the same status as the completed load plan run.

Usage

```
OdiStartLoadPlan -LOAD_PLAN_NAME=<load_plan_name> [-LOG_LEVEL=<log_level>]
[-CONTEXT=<context_code>] [-AGENT_URL=<agent_url>]
[-AGENT_CODE=<logical_agent_code>] [-ODI_USER=<ODI User>]
[-ODI_PASS=<ODI Password>] [-KEYWORDS=<Keywords>]
[-<PROJECT_CODE>.<VARIABLE>=<var_value> ...] [-SYNC=<yes|no>] [-POLLINT=<msec>]
```

Parameters

Parameters	Mandatory	Description
`-LOAD_PLAN_NAME=<load_plan_name>`	Yes	Name of the load plan to start.
`-LOG_LEVEL=<log_level>`	No	Level of logging information to retain. All sessions with a defined log level lower than or equal to this value are kept in the session log when the session completes. However, if object execution ends abnormally, all tasks are kept, regardless of this setting. Note that log level 6 has the same behavior as log level 5, but with the addition of variable and sequence tracking. See "Tracking Variables and Sequences" on page 13-21 for more information.

Parameters	Mandatory	Description
`[-CONTEXT=<context_code>]`	Yes	Code of the execution context. If this parameter is omitted, the load plan starts in the execution context of the calling session, if any.
`[-AGENT_URL=<agent_url>]`	No	URL of the remote agent that starts the load plan.
`[-AGENT_CODE=<logical_agent_code>]`	No	Code of the logical agent responsible for starting this load plan. If this parameter and `-AGENT_URL` are omitted, the current agent starts this load plan. This parameter is ignored if `-AGENT_URL` is specified.
`[-ODI_USER=<ODI user>]`	No	Oracle Data Integrator user to be used to start the load plan. The privileges of this user are used. If this parameter is omitted, the load plan is started with privileges of the user launching the parent session.
`[-ODI_PASS=<ODI Password>]`	No	Password of the Oracle Data Integrator user. This password must be encoded. This parameter is required if `-ODI_USER` is specified.
`-KEYWORDS=<keywords>`	No	Comma-separated list of keywords attached to this load plan. These keywords make load plan execution identification easier.
`-<VARIABLE>=<value>`	No	List of project or global variables whose value is set as the default for the execution of the load plan. Project variables should be named `<project_code>.<variable_name>` and global variables should be named `GLOBAL.<variable_name>`. This list is of the form `-<variable>=<value>`.
`-SYNC=<yes|no>`	No	Specifies whether the load plan should be executed synchronously or asynchronously. If set to Yes (synchronous mode), the load plan is started and runs to completion with a status of Done or Error before control is returned. If set to No (asynchronous mode), the load plan is started and control is returned before the load plan runs to completion. The default value is No.
`-POLLINT=<msec>`	No	The time in milliseconds to wait between polling the load plan run status for completion state. The `-SYNC` parameter must be set to Yes. The default value is 1000 (1 second). The value must be greater than 0.

Examples

Start load plan `LOAD_DWH` in the `GLOBAL` context on the same agent.

```
OdiStartLoadPlan -LOAD_PLAN_NAME=LOAD_DWH -CONTEXT=GLOBAL
```

OdiStartOwbJob

Use this command to execute Oracle Warehouse Builder (OWB) objects from within Oracle Data Integrator and to retrieve the execution audit data into Oracle Data Integrator.

This command uses an Oracle Warehouse Builder runtime repository data server that can be created in Topology Navigator. This data server must connect as an Oracle Warehouse Builder user who can access an Oracle Warehouse Builder workspace. The physical schemas under this data server represent the Oracle Warehouse Builder workspaces that this user can access. For information about the Oracle Data Integrator topology, see "Setting Up the Topology" on page 4-5.

Usage

```
OdiStartOwbJob -WORKSPACE=<logical_owb_repository> -LOCATION=<owb_location>
-OBJECT_NAME=<owb_object> -OBJECT_TYPE=<owb_object_type>
[-EXEC_PARAMS=<exec_params>] [-CONTEXT=<context_code>] [-LOG_LEVEL=<log_level>]
[-SYNC_MODE=<1|2>] [-POLLINT=<n>] [-SESSION_NAME=<session_name>]
[-KEYWORDS=<keywords>] [<OWB parameters>]
```

Parameters

Parameters	Mandatory	Description
`-WORKSPACE=<logical_owb_repository>`	Yes	Logical schema of the OWB Runtime Repository technology. This resolves to a physical schema that represents the Oracle Warehouse Builder workspace that contains the Oracle Warehouse Builder object to be executed. The Oracle Warehouse Builder workspace was chosen when you added a Physical Schema under the OWB Runtime Repository DataServer in Topology Navigator. The context for this mapping can also be specified using the `-CONTEXT` parameter.
`-LOCATION=<owb_location>`	Yes	Name of the Oracle Warehouse Builder location that contains the Oracle Warehouse Builder object to be executed. This location must exist in the physical workspace that resolves from `-WORKSPACE`.
`-OBJECT_NAME=<owb_object>`	Yes	Name of the Oracle Warehouse Builder object. This object must exist in `-LOCATION`.
`-OBJECT_TYPE=<owb_object_type>`	Yes	Type of Oracle Warehouse Builder object, for example: `PLSQLMAP`, `PROCESSFLOW`, `SQLLOADERCONTROLFILE`, `MAPPING`, `DATAAUDITOR`, `ABAPFILE`
`-EXEC_PARAMS=<exec_params>`	No	Custom and/or system parameters for the Oracle Warehouse Builder execution.

Parameters	Mandatory	Description
`-CONTEXT=<context_code>`	No	Execution context of the Oracle Warehouse Builder object. This is the context in which the logical workspace will be resolved. Studio editors use this value or the Default Context. Execution uses this value or the Parent Session context.
`-LOG_LEVEL=<log_level>`	No	Log level (0-5). The default value is 5, which means that maximum details are captured in the log.
`-SYNC_MODE=<1|2>`	No	Synchronization mode of the Oracle Warehouse Builder job: `1` - Synchronous (default). Execution of the session waits until the Oracle Warehouse Builder job terminates. `2` - Asynchronous. Execution of the session continues without waiting for the Oracle Warehouse Builder job to terminate.
`-POLLINT=<n>`	No	The period of time in milliseconds to wait between each transfer of Oracle Warehouse Builder audit data to Oracle Data Integrator log tables. The default value is 0, which means that audit data is transferred at the end of the execution.
`-SESSION_NAME=<session_name>`	No	Name of the Oracle Warehouse Builder session as it appears in the log.
`-KEYWORDS=<keywords>`	No	Comma-separated list of keywords attached to the session.
`<OWB parameters>`	No	List of values for the Oracle Warehouse Builder parameters relevant to the object. This list is of the form `-PARAM_NAME=value`. Oracle Warehouse Builder system parameters should be prefixed by `OWB_SYSTEM`, for example, `OWB_SYSTEM.AUDIT_LEVEL`.

Examples

Execute the Oracle Warehouse Builder process flow `LOAD_USERS` that has been deployed to the Oracle Workflow `DEV_OWF`.

```
OdiStartOwbJob -WORKSPACE=OWB_WS1 -CONTEXT=QA
-LOCATION=DEV_OWF -OBJECT_NAME=LOAD_USERS -OBJECT_TYPE=PROCESSFLOW
```

Execute the Oracle Warehouse Builder PL/SQL map `STAGE_USERS` that has been deployed to the database location `DEV_STAGE`. Poll and transfer the Oracle Warehouse Builder audit data every 5 seconds. Pass the input parameter `AGE_LIMIT` whose value is obtained from an Oracle Data Integrator variable, and specify an Oracle Warehouse Builder system parameter relevant to a PL/SQL map.

```
OdiStartOwbJob -WORKSPACE=OWB_WS1 -CONTEXT=QA
-LOCATION=DEV_STAGE -OBJECT_NAME=STAGE_USERS -OBJECT_TYPE=PLSQLMAP
-POLLINT=5000 -OWB_SYSTEM.MAX_NO_OF_ERRORS=25 -AGE_LIMIT=#VAR_MINAGE
```

OdiStartScen

Use this command to start a scenario.

The optional parameter `-AGENT_CODE` is used to dedicate this scenario to another agent other than the current agent.

The parameter `-SYNC_MODE` starts a scenario in synchronous or asynchronous mode.

Note: The scenario that is started should be present in the repository into which the command is launched. If you go to production with a scenario, make sure to also take all scenarios called by your scenario using this command. The Solutions can help you group scenarios for this purpose.

Usage

```
OdiStartScen -SCEN_NAME=<scenario> -SCEN_VERSION=<version>
[-CONTEXT=<context>] [-ODI_USER=<odi user> -ODI_PASS=<odi password>]
[-SESSION_NAME=<session_name>] [-LOG_LEVEL=<log_level>]
[-AGENT_CODE=<logical_agent_name>] [-SYNC_MODE=<1|2>]
[-KEYWORDS=<keywords>] [-<VARIABLE>=<value>]*
```

Parameters

Parameters	Mandatory	Description
`-SCEN_NAME=<scenario>`	Yes	Name of the scenario to start.
`-SCEN_VERSION=<version>`	Yes	Version of the scenario to start. If the version specified is -1, the last version of the scenario is executed.
`-CONTEXT=<context>`	No	Code of the execution context. If this parameter is omitted, the scenario is executed in the execution context of the calling session.
`-ODI_USER=<odi user>`	No	Oracle Data Integrator user to be used to run the scenario. The privileges of this user are used. If this parameter is omitted, the scenario is executed with privileges of the user launching the parent session.
`-ODI_PASS=<odi password>`	No	Password of the Oracle Data Integrator user. This password should be encoded. This parameter is required if the user is specified.
`-SESSION_NAME=<session_name>`	No	Name of the session that will appear in the execution log.
`-LOG_LEVEL=<log_level>`	No	Trace level (0 .. 5) to keep in the execution log. The default value is 5.
`-AGENT_CODE=<logical_agent_name>`	No	Name of the logical agent responsible for executing this scenario. If this parameter is omitted, the current agent executes this scenario.
`-SYNC_MODE=<1|2>`	No	Synchronization mode of the scenario: `1` - Synchronous mode (default). The execution of the calling session is blocked until the scenario finishes its execution. `2` - Asynchronous mode. The execution of the calling session continues independently from the return of the called scenario.

Parameters	Mandatory	Description
`-KEYWORDS=<keywords>`	No	Comma-separated list of keywords attached to this session. These keywords make session identification easier.
`-<VARIABLE>=<value>`	No	List of variables whose value is set for the execution of the scenario. This list is of the form `PROJECT.VARIABLE=value` or `GLOBAL.VARIABLE=value`.

Examples

Start the scenario `LOAD_DWH` in version `2` in the production context (synchronous mode).

```
OdiStartScen -SCEN_NAME=LOAD_DWH -SCEN_VERSION=2
-CONTEXT=CTX_PRODUCTION
```

Start the scenario `LOAD_DWH` in version `2` in the current context in asynchronous mode on the agent `UNIX Agent` while passing the values of the variables `START_DATE` (local) and `COMPANY_CODE` (global).

```
OdiStartScen -SCEN_NAME=LOAD_DWH -SCEN_VERSION=2 -SYNC_MODE=2
"-AGENT_CODE=UNIX Agent" -MY_PROJECT.START_DATE=10-APR-2002
-GLOBAL.COMPANY_CODE=SP4356
```

OdiUnZip

Use this command to extract an archive file to a directory.

Usage

```
OdiUnZip -FILE=<file> -TODIR=<target_directory> [-OVERWRITE=<yes|no>]
[-ENCODING=<file_name_encoding>]
```

Parameters

Parameters	Mandatory	Description
`-FILE=<file>`	Yes	Full path to the ZIP file to extract.
`-TODIR=<target_file>`	Yes	Destination directory or folder.
`-OVERWRITE=<yes|no>`	No	Indicates if the files that already exist in the target directory must be overwritten. The default value is No.
`-ENCODING=<file_name_encoding>`	No	Character encoding used for file names inside the archive file. For a list of possible values, see: `http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html` Defaults to the platform's default character encoding.

Examples

Extract the file `archive_001.zip` from directory `C:\archive\` into directory `C:\TEMP`.

```
OdiUnZip "-FILE=C:\archive\archive_001.zip" -TODIR=C:\TEMP\
```

OdiUpdateAgentSchedule

Use this command to force an agent to recalculate its schedule of tasks.

Usage

```
OdiUpdateAgentSchedule -AGENT_NAME=<physical_agent_name>
```

Parameters

Parameters	Mandatory	Description
`-AGENT_NAME=<physical_agent_name>`	Yes	Name of the physical agent to update.

Examples

Cause the physical agent `agt_s1` to update its schedule.

```
OdiUpdateAgentSchedule -AGENT_NAME=agt_s1
```

OdiWaitForChildSession

Use this command to wait for the child session (started using the OdiStartScen tool) of the current session to complete.

This command checks every `<polling_interval>` to determine if the sessions launched from `<parent_sess_number>` are finished. If all child sessions (possibly filtered by their name and keywords) are finished (status of Done, Warning, or Error), this command terminates.

Usage

```
OdiWaitForChildSession [-PARENT_SESS_NO=<parent_sess_number>]
[-POLL_INT=<polling_interval>]
[-SESSION_NAME_FILTER=<session_name_filter>]
[-SESSION_KEYWORDS=<session_keywords>]
[-MAX_CHILD_ERROR=ALL|<error_number>]
```

Parameters

Parameters	Mandatory	Description
`-PARENT_SESS_NO=<parent_sess_number>`	No	ID of the parent session. If this parameter is not specified, the current session ID is used.
`-POLL_INT=<polling_interval>`	No	Interval in seconds between each sequence of termination tests for the child sessions. The default value is 1.
`-SESSION_NAME_FILTER=<session_name_filter>`	No	Only child sessions whose names match this filter are tested. This filter can be a SQL LIKE-formatted pattern.
`-SESSION_KEYWORDS=<session_keywords>`	No	Only child sessions for which ALL keywords have a match in this comma-separated list are tested. Each element of the list can be a SQL LIKE-formatted pattern.

Parameters	Mandatory	Description
`-MAX_CHILD_ERROR= ALL|<error_number>`	No	This parameter enables OdiWaitForChildSession to terminate in error if a number of child sessions have terminated in error: ■ `ALL`: Error if all child sessions terminate in error. ■ `<error_number>`: Error if `<error_number>` or more child sessions terminate in error. If this parameter is equal to 0, negative, or not specified, OdiWaitForChildSession never terminates in an error status, regardless of the number of failing child sessions.

Examples

Wait and poll every 5 seconds for all child sessions of the current session with a name filter of `LOAD%` and keywords `MANDATORY` and `CRITICAL` to finish.

```
OdiWaitForChildSession -PARENT_SESS_NO=<%=odiRef.getSession("SESS_NO")%>
-POLL_INT=5 -SESSION_NAME_FILTER=LOAD%
-SESSION_KEYWORDS=MANDATORY,CRITICAL
```

OdiWaitForData

Use this command to wait for a number of rows in a table or set of tables. This can also be applied to a number of objects containing data, such as views.

The OdiWaitForData command tests that a table, or a set of tables, has been populated with a number of records. This test is repeated at regular intervals (`-POLLINT`) until one of the following conditions is met: the desired number of rows for one of the tables has been detected (`-UNIT_ROWCOUNT`), the desired, cumulated number of rows for all of the tables has been detected (`-GLOBAL_ROWCOUNT`), or a timeout (`-TIMEOUT`) has been reached.

Filters may be applied to the set of counted rows. They are specified by an explicit SQL where clause (`-SQLFILTER`) and/or the `-RESUME_KEY_xxx` parameters to determine field-value-operator clause. These two methods are cumulative (AND).

The row count may be considered either in absolute terms (with respect to the total number of rows in the table) or in differential terms (the difference between a stored reference value and the current row count value).

When dealing with multiple tables:

- The `-SQLFILTER` and `-RESUME_KEY_xxx` parameters apply to **ALL** tables concerned.
- The `-UNIT_ROWCOUNT` parameter determines the row count to be expected for each table. The `-GLOBAL_ROWCOUNT` parameter determines the SUM of the row count number cumulated over the set of tables. When only one table is concerned, the `-UNIT_ROWCOUNT` and `-GLOBAL_ROWCOUNT` parameters are equivalent.

Usage

```
OdiWaitForData -LSCHEMA=<logical_schema> -TABLE_NAME=<table_name>
[-OBJECT_TYPE=<list of object types>] [-CONTEXT=<context>]
[-RESUME_KEY_VARIABLE=<resumeKeyVariable>
-RESUME_KEY_COL=<resumeKeyCol>
[-RESUME_KEY_OPERATOR=<resumeKeyOperator>]|-SQLFILTER=<SQLFilter>]
[-TIMEOUT=<timeout>] [-POLLINT=<pollInt>]
[-GLOBAL_ROWCOUNT=<globalRowCount>]
[-UNIT_ROWCOUNT=<unitRowCount>] [-TIMEOUT_WITH_ROWS_OK=<yes|no>]
[-INCREMENT_DETECTION=<no|yes> [-INCREMENT_MODE=<M|P|I>]
[-INCREMENT_SEQUENCE_NAME=<incrementSequenceName>]]
```

Parameters

Parameters	Mandatory	Description
`-LSCHEMA=<logical_schema>`	Yes	Logical schema containing the tables.
`-TABLE_NAME=<table_name>`	Yes	Table name, mask, or list of table names to check. This parameter accepts three formats: ■ Table Name ■ Table Name Mask: This mask selects the tables to poll. The mask is specified using the SQL LIKE syntax: the `%` symbol replaces an unspecified number of characters and the _ symbol is a single character wildcard. ■ Table Names List: Comma-separated list of table names. Masks as defined above are allowed.
`-OBJECT_TYPE=<list of object types>`	No	Type of objects to check. By default, only tables are checked. To take into account other objects, specify a comma-separated list of object types. Supported object types are: ■ `T`: Table ■ `V`: View
`-CONTEXT=<context>`	No	Context in which the logical schema will be resolved. If no context is specified, the execution context is used.
`-SQLFILTER=<SQLFilter>`	No	Explicit SQL filter to be applied to the table(s). This statement must be valid for the technology containing the checked tables. Note that this statement must not include the `WHERE` keyword.

Parameters	Mandatory	Description
-RESUME_KEY_ VARIABLE=<resumeKeyVariable> -RESUME_KEY_COL=<resumeKeyCol> [-RESUME_KEY_ OPERATOR=<resumeKeyOperator>]	No	The RESUME_KEY_xxx parameters enable filtering of the set of counted rows in the polled tables. ■ <key_column>: Name of a column in the checked table. ■ <operator>: Valid comparison operator for the technology containing the checked tables. If this parameter is omitted, the value > is used by default. ■ <variable_name>: Variable name whose value has been previously set. The variable name must be prefixed with : (bind) or # (substitution). The variable scope should be explicitly stated in the Oracle Data Integrator syntax; GLOBAL.<variable name> for global variables or <project code>.<variable name> for project variables.
-TIMEOUT=<timeout>	No	Maximum period of time in milliseconds over which data is polled. If this value is equal to 0, the timeout is infinite. The default value is 0.
-POLLINT=<pollInt>	No	The period of time in milliseconds to wait between data polls. The default value is 1000.
-UNIT_ROWCOUNT=<unitRowCount>	No	Number of rows expected in a polled table to terminate the command. The default value is 1.
-GLOBAL_ROWCOUNT=<globalRowCount>	No	Total number of rows expected cumulatively, over the set of tables, to terminate the command. If not specified, the default value 1 is used.
-INCREMENT_DETECTION=<no|yes>	No	Defines the mode in which the command considers row count: either in absolute terms (with respect to the total number of rows in the table) or in differential terms (the difference between a stored reference value and the current row count value). ■ If set to Yes, the row count is performed in differential mode. The number of additional rows in the table is compared to a stored **reference value**. The reference value depends on the -INCREMENT_MODE parameter. ■ If set to No, the count is performed in absolute row count mode. The default value is No.

Parameters	Mandatory	Description
`-INCREMENT_MODE=<M|P|I>`	No	This parameter specifies the persistence mode of the **reference value** between successive OdiWaitForData calls. Possible values are: ■ `M`: Memory. The **reference value** is nonpersistent. When OdiWaitForData is called, the **reference value** takes a value equal to the number of rows in the polled table. When OdiWaitForData ends the value is lost. A following call in this mode sets a new reference value. ■ `P`: Persistent. The **reference value** is persistent. It is read from the **increment sequence** when OdiWaitForData starts and saved in the increment sequence when OdiWaitForData ends. If the **increment sequence** is not set (at initial call time) the current table row count is used. ■ `I`: Initial. The **reference value** is initialized and is persistent. When OdiWaitForData starts, the **reference value** takes a value equal to the number of rows in the polled table. When OdiWaitForData ends, it is saved in the increment sequence as for the persistent mode. The default value is `M`. Note that using the Persistent or Initial modes is not supported when a mask or list of tables is polled.
`-INCREMENT_SEQUENCE_ NAME=<incrementSequenceName>`	No	This parameter specifies the name of an automatically allocated storage space used for reference value persistence. This increment sequence is stored in the Repository. If this name is not specified, it takes the name of the table. Note that this Increment Sequence is not an Oracle Data Integrator Sequence and cannot be used as such outside a call to OdiWaitForData.
`-TIMEOUT_WITH_ROWS_OK=<yes|no>`	No	If this value is set to Yes, at least one row was detected, and the timeout occurs before the expected number of rows has been inserted, the API exits with a return code of 0. Otherwise, it signals an error. The default value is Yes.

Examples

Wait for the `DE1P1` table in the `ORA_WAITFORDATA` schema to contain 200 records matching the filter.

```
OdiWaitForData -LSCHEMA=ORA_WAITFORDATA -TABLE_NAME=DE1P1
-GLOBAL_ROWCOUNT=200 "-SQLFILTER=DATMAJ >
to_date('#MAX_DE1_DATMAJ_ORACLE_CHAR', 'DD/MM/YYYY HH24:MI:SS')"
```

Wait for a maximum of 4 hours for new data to appear in either the `CITY_SRC` or the `CITY_TRG` table in the logical schema `SQLSRV_SALES`.

```
OdiWaitForData -LSCHEMA=SQLSRV_SALES -TABLE_NAME=CITY%
-TIMEOUT=14400000 -INCREMENT_DETECTION=yes
```

OdiWaitForLoadPlans

Use this command to wait for load plan runs to complete.

Usage

```
OdiWaitForLoadPlans [-PARENT_SESS_NO=<parent_sess_guid>]
[-LP_NAME_FILTER=<load_plan_name_filter>] [-LP_KEYWORDS=<load_plan_keywords>]
[-MAX_LP_ERROR=ALL|<number_of_lp_errors>] [-POLLINT=<polling_interval_msec>]
```

Parameters

Parameters	Mandatory	Description
`-PARENT_SESS_NO=<parent_sess_guid>`	No	Global ID of the parent session that started the load plan. If this parameter is not specified, the global ID of the current session is used.
`-LP_NAME_FILTER=<load_plan_name_filter>`	No	Only load plan runs whose name matches this filter are tested for completion status. This filter can be a SQL LIKE-formatted pattern.
`-LP_KEYWORDS=<load_plan_keywords>`	No	Only load plan runs whose keywords contain all entries in this comma-separated list are tested for completion status. Each element in the list can be a SQL LIKE-formatted pattern.
`-MAX_LP_ERROR=ALL|<number_of_lp_errors>`	No	OdiWaitForLoadPlans terminates in error if a number of load plan runs are in Error status: ■ `ALL`: Error if all load plan runs complete in Error status. ■ `<number_of_lp_errors>`: Error if the number of load plan runs in Error status is at or above this value `<number_of_lp_errors>` when all load plan runs are complete. If this parameter is not specified or its value is less than 1, OdiWaitForLoadPlans never terminates in error, regardless of the number of load plan runs in Error status.
`-POLLINT=<polling_interval_msec>`	No	The time in milliseconds to wait between polling load plan run status for completion state. The default value is 1000 (1 second). The value must be greater than 0.

Examples

Wait and poll every 5 seconds for all load plan runs started by the current session with a name filter of `POPULATE%` and keywords `MANDATORY` and `CRITICAL` to finish in a Done or Error status. If 2 or more load plan runs are in Error status when execution is complete for all selected load plan runs, OdiWaitForLoadPlans ends in error.

```
OdiWaitForLoadPlans -PARENT_SESS_NO=<%=odiRef.getSession("SESS_GUID")%>
```

```
-LP_NAME_FILTER=POPULATE% -LP_KEYWORDS=MANDATORY,CRITICAL
-POLLINT=5000 -MAX_LP_ERROR=2
```

OdiWaitForLogData

Use this command to wait for a number of modifications to occur on a journalized table or a list of journalized tables.

The OdiWaitForLogData command determines whether rows have been modified on a table or a group of tables. These changes are detected using the Oracle Data Integrator changed data capture (CDC) in simple mode (using the `-TABLE_NAME` parameter) or in consistent mode (using the `-CDC_SET_NAME` parameter). The test is repeated every `-POLLINT` milliseconds until one of the following conditions is met: the desired number of row modifications for one of the tables has been detected (`-UNIT_ROWCOUNT`), the desired cumulative number of row modifications for all of the tables has been detected (`-GLOBAL_ROWCOUNT`), or a timeout (`-TIMEOUT`) has been reached.

Note: This command takes into account all journalized operations (inserts, updates and deletes).

The command is suitable for journalized tables only in simple or consistent mode.

Usage

```
OdiWaitForLogData -LSCHEMA=<logical_schema>  -SUBSCRIBER_NAME=<subscriber_name>
(-TABLE_NAME=<table_name> | -CDC_SET_NAME=<cdcSetName>)
[-CONTEXT=<context>] [-TIMEOUT=<timeout>] [-POLLINT=<pollInt>]
[-GLOBAL_ROWCOUNT=<globalRowCount>]
[-UNIT_ROWCOUNT=<unitRowCount> [-OPTIMIZED_WAIT=<yes|no|AUTO>]
[-TIMEOUT_WITH_ROWS_OK=<yes|no>]
```

Parameters

Parameters	Mandatory	Description
`-CONTEXT=<context>`	No	Context in which the logical schema will be resolved. If no context is specified, the execution context is used.
`-GLOBAL_ROWCOUNT=<globalRowCount>`	No	Total number of changes expected in the tables or the CDC set to end the command. The default value is 1.
`-LSCHEMA=<logical_schema>`	Yes	Logical schema containing the journalized tables.

Parameters	Mandatory	Description
`-OPTIMIZED_WAIT=<yes|no|AUTO>`	No	Method used to access the journals. ■ `yes`: Optimized method. This method works for later versions of journalizing. It runs faster than the nonoptimized mode. ■ `no`: Nonoptimized method. A count is performed on the journalizing table. This method is of lower performance but compatible with earlier versions of the journalizing feature. ■ `AUTO`: If more than one table is checked, the optimized method is used. Otherwise, the nonoptimized method is used. The default value is `AUTO`.
`-POLLINT=<pollInt>`	No	The period of time in milliseconds to wait between polls. The default value is 2000.
`-SUBSCRIBER_NAME=<subscriber_name>`	Yes	Name of the subscriber used to get the journalizing information.
`-TABLE_NAME=<table_name>`	Yes	Journalized table name, mask, or list to check. This parameter accepts three formats: ■ Table Name ■ Table Name Mask: This mask selects the tables to poll. The mask is specified using the SQL LIKE syntax: the `%` symbol replaces an unspecified number of characters and the _ symbol acts as a wildcard. ■ Table Names List: List of table names separated by commas. Masks as defined above are **not** allowed. Note that this option works only for tables in a model journalized in simple mode. This parameter cannot be used with `-CDC_SET_NAME`. It is mandatory if `-CDC_SET_NAME`. is not set.
`-CDC_SET_NAME=<cdcSetName>`	Yes	Name of the CDC set to check. This CDC set name is the fully qualified model code, typically `PHYSICAL_SCHEMA_NAME.MODEL_CODE`. It can be obtained in the current context using a substitution method API call, as shown below: `<%=odiRef.getObjectName("L", "model_code", "logical_schema", "D")%>`. Note that this option works only for tables in a model journalized in consistent mode. This parameter cannot be used with `-TABLE_NAME`. It is mandatory if `-TABLE_NAME` is not set.
`-TIMEOUT=<timeout>`	No	Maximum period of time in milliseconds over which changes are polled. If this value is equal to 0, the timeout is infinite. The default value is 0.

Parameters	Mandatory	Description
-TIMEOUT_WITH_ROWS_OK=<yes|no>	No	If this parameter is set to Yes, at least one row was detected, and the timeout occurs before the predefined number of rows has been polled, the API exits with a return code of 0. Otherwise, it signals an error. The default value is Yes.
-UNIT_ROWCOUNT=<unitRowCount>	No	Number of changes expected in one of the polled tables to end the command. The default value is 1. Note that -UNIT_ROWCOUNT is not taken into account with -CDC_SET_NAME.

Examples

Wait for the CUSTOMERS table in the SALES_APPLICATION schema to have 200 row modifications recorded for the SALES_SYNC subscriber.

```
OdiWaitForLogData -LSCHEMA=SALES_APPLICATION
-TABLE_NAME=CUSTOMERS -GLOBAL_ROWCOUNT=200
-SUBSCRIBER_NAME=SALES_SYNC
```

OdiWaitForTable

Use this command to wait for a table to be created and populated with a predefined number of rows.

The OdiWaitForTable command regularly tests whether the specified table has been created and has been populated with a number of records. The test is repeated every -POLLINT milliseconds until the table exists and contains the desired number of rows (-GLOBAL_ROWCOUNT), or until a timeout (-TIMEOUT) is reached.

Usage

```
OdiWaitForTable -CONTEXT=<context> -LSCHEMA=<logical_schema>
-TABLE_NAME=<table_name> [-TIMEOUT=<timeout>] [-POLLINT=<pollInt>]
[-GLOBAL_ROWCOUNT=<globalRowCount>] [-TIMEOUT_WITH_ROWS_OK=<yes|no>]
```

Parameters

Parameters	Mandatory	Description
-CONTEXT=<context>	No	Context in which the logical schema will be resolved. If no context is specified, the execution context is used.
-GLOBAL_ROWCOUNT=<globalRowCount>	No	Total number of rows expected in the table to terminate the command. The default value is 1. If not specified, the command finishes when a new row is inserted into the table.
-LSCHEMA=<logical_schema>	Yes	Logical schema in which the table is searched for.
-POLLINT=<pollInt>	No	Period of time in milliseconds to wait between each test. The default value is 1000.

Parameters	Mandatory	Description
`-TABLE_NAME=<table_name>`	Yes	Name of table to search for.
`-TIMEOUT=<timeout>`	No	Maximum time in milliseconds the table is searched for. If this value is equal to 0, the timeout is infinite. The default value is 0.
`-TIMEOUT_WITH_ROWS_OK=<yes|no>`	No	If this parameter is set to Yes, at least one row is detected, and the timeout occurs before the expected number of records is detected, the API exits with a return code of 0. Otherwise, it signals an error. The default value is Yes.

Examples

Wait for the `DE1P1` table in the `ORA_WAITFORDATA` schema to exist, and to contain at least 1 record.

```
OdiWaitForTable -LSCHEMA=ORA_WAITFORDATA -TABLE_NAME=DE1P1
-GLOBAL_ROWCOUNT=1
```

OdiXMLConcat

Use this command to concatenate elements from multiple XML files into a single file.

This tool extracts all instances of a given element from a set of source XML files and concatenates them into one target XML file. The tool parses and generates well formed XML. It does not modify or generate a DTD for the generated files. A reference to an existing DTD can be specified in the `-HEADER` parameter or preserved from the original files using `-KEEP_XML_PROLOGUE`.

Note: XML namespaces are not supported by this tool. Provide the local part of the element name (without the namespace or prefix value) in the `-ELEMENT_NAME` parameter.

Usage

```
OdiXMLConcat -FILE=<file_filter> -TOFILE=<target_file>
-XML_ELEMENT=<element_name> [-CHARSET_ENCODING=<encoding>]
[-IF_FILE_EXISTS=<overwrite|skip|error>]
[-KEEP_XML_PROLOGUE=<all|xml|doctype|none>] [-HEADER=<header>]
[-FOOTER=<footer>]
```

Parameters

Parameters	Mandatory	Description
`-FILE=<file_filter>`	Yes	Filter for the source XML files. This filter uses standard file wildcards (`?`,`*`). It includes both file names and directory names. Source files can be taken from the same folder or from different folders. The following file filters are valid: ■ `/tmp/files_*/customer.xml` ■ `/tmp/files_*/*.*` ■ `/tmp/files_??/customer.xml` ■ `/tmp/files/customer_*.xml` ■ `/tmp/files/customer_??.xml`
`-TOFILE=<target_file>`	Yes	Target file into which the elements are concatenated.
`-XML_ELEMENT=<element_name>`	Yes	Local name of the XML element (without enclosing <> characters, prefix, or namespace information) to be extracted with its content and child elements from the source files. Note that this element detection is not recursive. If a given instance of `<element_name>` contains other instances of `<element_name>`, only the element of higher level is taken into account and child elements are only extracted as a part of the top element's content.
`-CHARSET_ENCODING=<encoding>`	No	Target files encoding. The default value is `ISO-8859-1`. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
`-IF_FILE_EXISTS=<overwrite|skip|error>`	No	Define behavior when the target file exists. ■ `overwrite`: Overwrite the target file if it exists. ■ `skip`: Do nothing for this file. ■ `error`: Raise an error.
`-KEEP_XML_PROLOGUE=<all|xml|doctype|none>`	No	Copies the source file XML prologue in the target file. Depending on this parameter's value, the following parts of the XML prologue are preserved: ■ `all`: Copies all of the prologue (XML and document type declaration). ■ `xml`: Copies only the XML declaration `<?xml...?>` and not the document type declaration. ■ `doctype`: Copies only the document type declaration and not the XML declaration. ■ `none`: Does not copy the prologue from the source file. **Note**: If all or part of the prologue is not preserved, it should be specified in the `-HEADER` parameter.

Parameters	Mandatory	Description
`-HEADER=<header>`	No	String that is appended after the prologue (if any) in each target file. You can use this parameter to create a customized XML prologue or root element.
`-FOOTER=<footer>`	No	String that is appended at the end of each target file. You can use this parameter to close a root element added in the header.

Examples

Concatenate the content of the IDOC elements in the files `ord1.xml`, `ord2.xml`, and so on in the `ord_i` subfolder into the file `MDSLS.TXT.XML`, with the root element `<WMMBID02>` added to the target.

```
OdiXMLConcat "-FILE=./ord_i/ord*.xml" "-TOFILE=./MDSLS.TXT.XML" -XML_ELEMENT=IDOC
"-CHARSET_ENCODING=UTF-8" -IF_FILE_EXISTS=overwrite -KEEP_XML_PROLOGUE=xml
"-HEADER=<WMMBID02>" "-FOOTER=</WMMBID02>"

OdiXMLConcat "-FILE=./o?d_*/ord*.xml" "-TOFILE=./MDSLS.TXT.XML" -XML_ELEMENT=IDOC
"-CHARSET_ENCODING=UTF-8" -IF_FILE_EXISTS=overwrite -KEEP_XML_PROLOGUE=none
"-HEADER=<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<WMMBID02>"
"-FOOTER=</WMMBID02>"
```

Concatenate the EDI elements of the files `ord1.xml`, `ord2.xml`, and so on in the `ord_i` subfolder into the file `MDSLS2.XML`. This file will have the new root element `EDI_BATCH` above all `<EDI>` elements.

```
OdiXMLConcat "-FILE=./o?d_?/ord*.xml" "-TOFILE=./MDSLS2.XML" -XML_ELEMENT=EDI
"-CHARSET_ENCODING=UTF-8" -IF_FILE_EXISTS=overwrite -KEEP_XML_PROLOGUE=xml
"-HEADER= <EDI_BATCH>" "-FOOTER=</EDI_BATCH>"
```

OdiXMLSplit

Use this command to split elements from an XML file into several files.

This tool extracts all instances of a given element stored in a source XML file and splits it over several target XML files. This tool parses and generates well formed XML. It does not modify or generate a DTD for the generated files. A reference to an existing DTD can be specified in the `-HEADER` parameter or preserved from the original files using `-KEEP_XML_PROLOGUE`.

Note: XML namespaces are not supported by this tool. Provide the local part of the element name (without the namespace or prefix value) in the `-ELEMENT_NAME` parameter.

Usage

```
OdiXMLSplit -FILE=<file> -TOFILE=<file_pattern> -XML_ELEMENT=<element_name>
[-CHARSET_ENCODING=<encoding>] [-IF_FILE_EXISTS=<overwrite|skip|error>]
[-KEEP_XML_PROLOGUE=<all|xml|doctype|none>] [-HEADER=<header>]
[-FOOTER=<footer>]
```

Parameters

Parameters	Mandatory	Description
-FILE=<file>	Yes	Source XML file to split.
-TOFILE=<file_pattern>	Yes	File pattern for the target files. Each file is named after a pattern containing a mask representing a generated number sequence or the value of an attribute of the XML element used to perform the split: ■ **Number Sequence Mask**: Use the * (star) value to indicate the place of the file number value. For example, if the <file_ pattern> is equal to target_*.xml, the files created are named target_1.xml, target_2.xml, and so on. ■ **Attribute Value Mask**: Between square brackets, specify the name of the attribute of <element_name> whose value should be pushed to create the file name. For example, customer_[CUSTID].xml creates files named customer_041.xml, customer_123.xml, and so on, depending on the value of the attribute CUSTID of the element used to split. Note that if a value repeats over several successive elements, target files may be overwritten according to the value of the -OVERWRITE parameter. Note that the pattern can be used for creating different files within a directory or files in different directories. The following patterns are valid: ■ /tmp/files_*/customer.xml ■ /tmp/files_[CUSTID]/customer.xml ■ /tmp/files/customer_*.xml ■ /tmp/files/customer_[CUSTID].xml
-XML_ELEMENT=<element_name>	Yes	Local name of the XML element (without enclosing <> characters, prefix, or namespace information) to be extracted with its content and child elements from the source files. Note that this element detection is not recursive. If a given instance of <element_name> contains other instances of <element_name>, only the element of higher level is taken into account and child elements are only extracted as a part of the top element's content.
-CHARSET_ENCODING=<encoding>	No	Target files encoding. The default value is ISO-8859-1. For the list of supported encodings, see: http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html
-IF_FILE_ EXISTS=<overwrite|skip|error>	No	Define behavior when the target file exists. ■ overwrite: Overwrite the target file if it exists. ■ skip: Do nothing for this file. ■ error: Raise an error.

Parameters	Mandatory	Description
`-KEEP_XML_ PROLOGUE=<all|xml|doctype|none>`	No	Copies the source file XML prologue in the target file. Depending on this parameter's value, the following parts of the XML prologue are preserved: ■ `all`: Copies all of the prologue (XML and document type declaration). ■ `xml`: Copies only the XML declaration `<?xml...?>` and not the document type declaration. ■ `doctype`: Copies only the document type declaration and not the XML declaration. ■ `none`: Does not copy the prologue from the source file. **Note**: If all or part of the prologue is not preserved, it should be specified in the `-HEADER` parameter.
`-HEADER=<header>`	No	String that is appended after the prologue (if any) in each target file. You can use this parameter to create a customized XML prologue or root element.
`-FOOTER=<footer>`	No	String that is appended at the end of each target file. You can use this parameter to close a root element added in the header.

Examples

Split the file `MDSLS.TXT.XML` into several files. The files `ord1.xml`, `ord2.xml`, and so on are created and contain each instance of the IDOC element contained in the source file.

```
OdiXMLSplit "-FILE=./MDSLS.TXT.XML" "-TOFILE=./ord_i/ord*.xml" -XML_ELEMENT=IDOC
"-CHARSET_ENCODING=UTF-8" -IF_FILE_EXISTS=overwrite -KEEP_XML_PROLOGUE=xml
"-HEADER= <WMMBID02>" "-FOOTER= </WMMBID02>"
```

Split the file `MDSLS.TXT.XML` the same way as in the previous example except name the files using the value of the `BEGIN` attribute of the IDOC element that is being split. The XML prologue is not preserved in this example but entirely generated in the header.

```
OdiXMLSplit "-FILE= ./MDSLS.TXT.XML" "-TOFILE=./ord_i/ord[BEGIN].xml"
-XML_ELEMENT=IDOC "-CHARSET_ENCODING=UTF-8" -IF_FILE_EXISTS=overwrite -KEEP_XML
PROLOGUE=none "-HEADER= <?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<WMMBID02>"
"-FOOTER=</WMMBID02>"
```

OdiZip

Use this command to create a ZIP file from a directory or several files.

Usage

```
OdiZip -DIR=<directory> -FILE=<file> -TOFILE=<target_file> [-OVERWRITE=<yes|no>]
[-RECURSE=<yes|no>] [-CASESENS=<yes|no>]
[-ENCODING=<file_name_encoding>]
```

Parameters

Parameters	Mandatory	Description
`-DIR=<directory>`	Yes if `-FILE` is omitted	Base directory (or folder) that will be the future root in the ZIP file to generate. If only `-DIR` and not `-FILE` is specified, all files under this directory are archived.
`-FILE=<file>`	Yes if `-DIR` is omitted	Path from the base directory of the file(s) to archive. If only `-FILE` and not `-DIR` is specified, the default directory is the current work directory if the `-FILE` path is relative. Use `*` to specify the generic characters. Examples: `/var/tmp/*.log` (all files with the `log` extension of the directory `/var/tmp`) `arch_*.lst` (all files starting with `arch_` and with the extension `lst`)
`-TOFILE=<target_file>`	Yes	Target ZIP file.
`-OVERWRITE=<yes|no>`	No	Indicates whether the target ZIP file must be overwritten (Yes) or simply updated if it already exists (No). By default, the ZIP file is updated if it already exists.
`-RECURSE=<yes|no>`	No	Indicates if the archiving is recursive in the case of a directory that contains other directories. The value No indicates that only the files contained in the directory to copy (without the subfolders) are archived.
`-CASESENS=<yes|no>`	No	Indicates if file search is case-sensitive. By default, Oracle Data Integrator searches files in uppercase (set to No).
`-ENCODING=<file_name_encoding>`	No	Character encoding to use for file names inside the archive file. For the list of supported encodings, see: `http://java.sun.com/j2se/1.4.2/docs/guide/intl/encoding.doc.html` This defaults to the platform's default character encoding.

Examples

Create an archive of the directory `C:\Program files\odi`.

```
OdiZip "-DIR=C:\Program Files\odi" -FILE=*.* -TOFILE=C:\TEMP\odi_archive.zip
```

Create an archive of the directory `C:\Program files\odi` while preserving the `odi` directory in the archive.

```
OdiZip "-DIR=C:\Program Files" -FILE=odi\*.* -TOFILE=C:\TEMP\odi_archive.zip
```

B

Using Groovy Scripting with Oracle Data Integrator

This appendix 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

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. Groovy language expressions in ADF Business Components differs from the Java code that you might use in a Business Components custom Java class because Groovy expressions are executed at runtime, while the strongly typed language of Java is executed at compile-time. Additionally, because Groovy expressions are dynamically compiled, they are stored in the XML definition files of the business components where you use it. ADF Business Components supports the use of the Groovy scripting language in places where access to entity object and view object attributes is useful, including attribute validators (for entity objects), attribute default values (for either entity objects or view objects), transient attribute value calculations (for either entity objects or view objects), bind variable default values (in view object query statements and view criteria filters), and placeholders for error messages (in entity object validation rules). Additionally, ADF Business Components provides a limited set of built-in keywords that can be used in Groovy expressions.

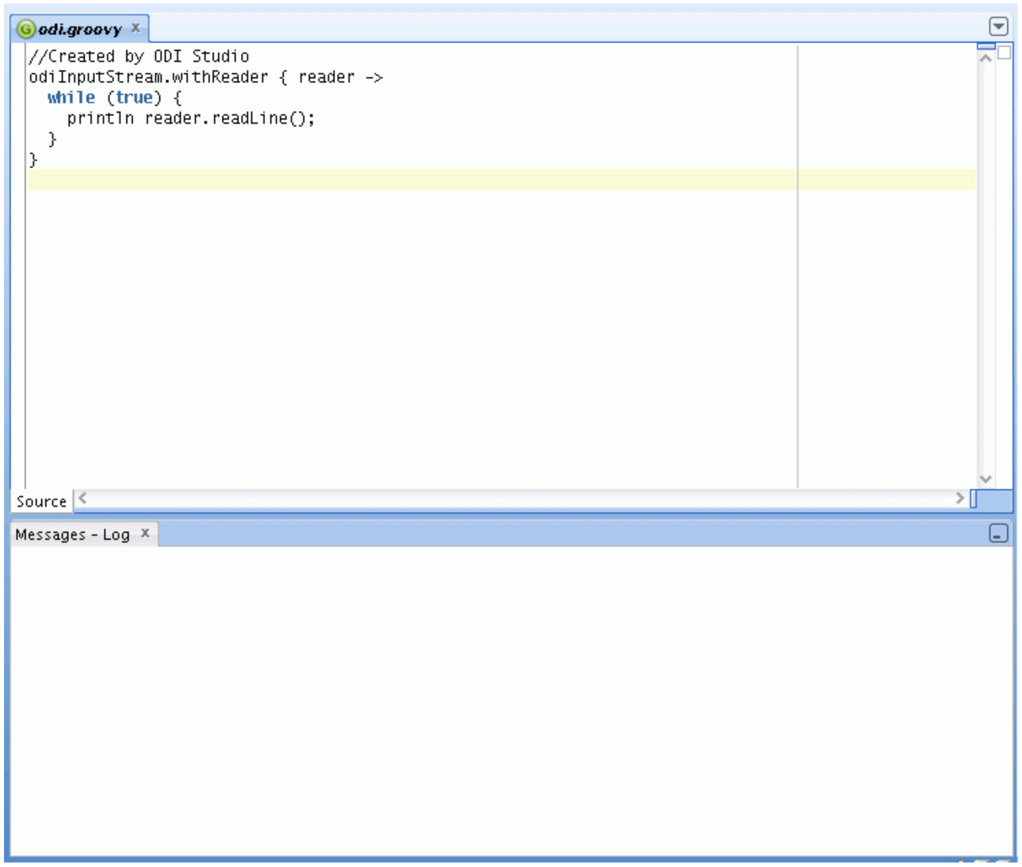
For more information about the Groovy language, see the following web site:

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

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 B–1 gives an overview of the Groovy editor.

Figure B–1 Groovy Editor



```
//Created by ODI Studio
odiInputStream.withReader { reader ->
  while (true) {
    println reader.readLine();
  }
}
```

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

Using the Groovy Editor

You can perform the following actions with the Groovy editor:

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.

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.

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.

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.

Note that each script execution launches its own Log window. The Log window is named according to the following format: *Running <script_name>*.

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.

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 B–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 B–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 B–2, "Creating a Project" shows how you can use the *odiInstance* variable.

Automating Development Tasks - Examples

Oracle Data Integrator provides support for the use of Groovy to automate development tasks. These tasks include for example:

- Example B–2, "Creating a Project"
- Example B–3, "External Groovy File"
- Example B–4, "Class from External File"
- Example B–5, "For Studio UI Automation"

Example B–2 shows how to create an ODI Project with a Groovy script.

Example B–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.getTransactionalEntityManager().persist(myProject)
tm.commit(txnStatus)
```

Example B–3 shows how to import an external Groovy script.

Example B–3 External Groovy File

```
//Created by ODI Studio
import gd.Test1;
println "Hello World"
Test1 t1 = new Test1()
println t1.getName()
```

Example B–4 shows how to call a class from a different Groovy script.

Example B–4 Class from External File

```
import gd.GroovyTestClass

GroovyTestClass tc = new GroovyTestClass()
println tc.getClassLoaderName()
```

Example B–5 shows how to implement Studio UI automation.

Example B–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()
```

C

Oracle Warehouse Builder to Oracle Data Integrator Migration Utility Patch

This appendix provides information about the new and enhanced features that are provided in ODI 12.1.2 patch number 17053768 "Oracle Warehouse Builder to Oracle Data Integrator Migration Utility Patch".

The ODI 12.1.2 patch number 17053768 is available for search and download through My Oracle Support. To access My Oracle Support, click the following URL:

`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info`

This appendix includes the following sections:

- New Features
- Enhanced Features

New Features

This section provides information about the new features that are available in ODI 12.1.2 patch number 17053768. This section includes the following topics:

- Pivot Component
- Unpivot Component
- Table Function Component
- Subquery Filter Component

Pivot Component

A pivot component is a projector component (see: "Projector Components" on page 11-10) 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.

Example: Pivoting Sales Data

SALES 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 C–1 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

PIVOTED DATA depicts data from the relational table SALES after unpivoting 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 C–2 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

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 unpivot 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.

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 unpivot component are displayed in the Property Inspector.
5. Enter a name and description for the pivot component.
6. 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.
7. 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.
8. 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.
9. 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.
10. Drag and drop the target datastore into the logical diagram.
11. 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.
12. 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.
13. Go to the physical diagram and assign new KMs if you want to.

    Save and execute the mapping to perform the pivot operation.

Unpivot Component

An unpivot component is a projector component (see: "Projector Components" on page 11-10) 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.

Example: Unpivoting Sales Data

The external table, QUARTERLY_SALES_DATA, shown in Table C–3, contains data from a flat file. There is a row for each year and separate attributes for sales in each quarter.

Table C–3 QUARTERLY_SALES_DATA

Year	Q1_Sales	Q2_Sales	Q3_Sales	Q4_Sales
2010	10.5	11.4	9.5	8.7

Table C–3 (Cont.) QUARTERLY_SALES_DATA

Year	Q1_Sales	Q2_Sales	Q3_Sales	Q4_Sales
2011	9.5	10.5	10.3	7.6

Table C–4 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 C–4 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

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.

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.

Table Function Component

A table function component is a projector component (see: "Projector Components" on page 11-10) 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.

Subquery Filter Component

A subquery filter component is a projector component (see: "Projector Components" on page 11-10) 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.

Enhanced Features

This section provides information about the existing features that have been enhanced in ODI 12.1.2 patch number 17053768. This section includes the following topics:

- Lookup Component Enhancements
- Sequence Enhancements

Lookup Component Enhancements

This section provides information about the enhancements done to the Lookup component. Two new properties, **Multiple Match Rows** and **No-Match Rows** are added to the Lookup properties. These properties appear under **Match Row Rules** section of the Lookup properties page.

Multiple Match Rows

The **Lookup Type** property has been replaced with **Multiple Match Rows**.

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 cause mapping to fail**

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

- **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.

- **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.

Lookup Attributes Order: Use the **Lookup Attributes Default Value & Order By** table to specify how the result set that contains multiple rows should be ordered. 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 arrows to change the position of the attributes to specify the order.

No-Match Rows

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 specify the action 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.

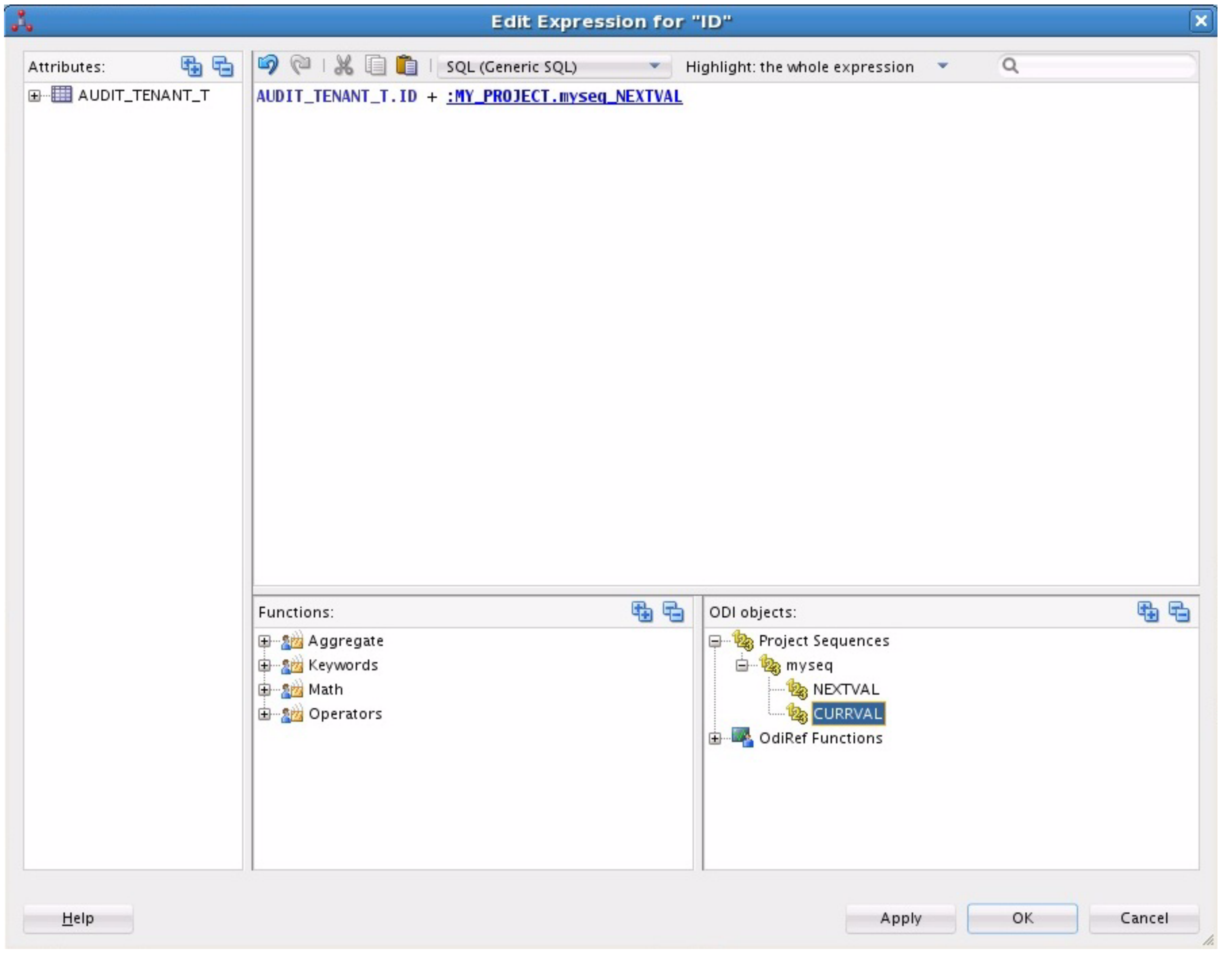
- **Return a row with the following default values**

  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.

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 C–1.

Figure C–1 Expression editor enhancement: SEQUENCE



```
AUDIT_TENANT_T.ID + :MY_PROJECT.myseq_NEXTVAL
```