

Oracle® Data Integrator

User's Guide

10g Release 3 (10.1.3)

January 2009

Copyright © 2006, Oracle. All rights reserved.

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Table Of Contents

Document Structure	1
Concepts	3
Introduction	3
What is Oracle Data Integrator?	3
Who is Oracle Data Integrator designed for?.....	3
Why Oracle Data Integrator?	3
Interfaces	3
Constructing interfaces	4
Maintaining interfaces	4
Replication and Transformation.....	4
Data replication	4
Data transformation.....	4
Interface Typology	5
Storage Technologies.....	6
Data Quality	7
Data Quality in Oracle Data Integrator.....	7
Oracle Data Profiling and Data Quality for Data Integrator.....	8
Productivity	9
The evolution of RDBMS.....	9
The evolution of technologies and needs	10
The importance of the right tool	10
Knowledge Modules	10
What are Knowledge Modules?	10
Different types of Knowledge Modules	10
How does it work?.....	11
Customization of Knowledge Modules.....	11
Designer	13
Introduction to Designer.....	13
Designer's Interface	13
Handling Models	14
What is a Model?	14
Creating and Reverse-Engineering a Model.....	16
Organizing Models into Model Folders	17
Creating and Organizing Sub-Models.....	18
Creating a Datastore	19
Editing and Viewing a Datastore's Data.....	19
Common Format Designer.....	20
Changed Data Capture	25
Setting Up Data Services.....	33

Handling Projects.....	38
What is a Project?	38
Managing an Oracle Data Integrator Project	40
Creating a New Project	41
Creating a New Folder	41
Creating Packages.....	41
Creating Scenarios.....	49
Creating Interfaces.....	51
Creating Procedures	58
Creating Variables & Sequences	61
Handling Knowledge Modules	66
Working with Oracle Data Quality	68
Using Web Services.....	76
Using User Functions.....	80
Using Markers and Memos	81
Using Cross-References	83
Using Version Management.....	84
Version Comparison Tool.....	88
Handling Concurrent Changes.....	90
Execution	91
Executing a Scenario from Designer or Operator	91
Executing a Scenario from an OS Command.....	92
Managing Executions through Web Services	94
Executing a Scenario from an HTTP URL	98
Working with a Scenario from a Different Repository	104
Scheduling a Scenario with an External Scheduler	105
Restarting a Session	105
Other Operations	106
Connecting to a Work Repository	106
Changing a User's Password.....	106
Setting User Parameters.....	107
Designer: Generating a Report	107
Using the Expression Editor.....	108
Searching for an object	109
Import/Export.....	109
Operator	113
Introduction to Operator.....	113
Operator's Interface.....	113
Working with Operator	115
Sessions.....	115

Scenarios	116
Filtering Sessions in Operator	116
Displaying Scheduling Information	117
Purging the Log	117
Restarting a Session.....	118
Importing Scenarios in Production.....	118
Scenario Folders.....	119
Using Session Folders.....	119
Importing and Exporting the Log	120
Importing Log Data.....	120
Exporting Log Data	120
Topology.....	123
Introduction to Topology Manager.....	123
Topology Manager's Interface.....	123
What is the Topology?.....	124
Physical Architecture.....	125
Contexts	125
Logical Architecture.....	125
Languages	125
Repositories	125
Hosts	126
Defining the Topology.....	126
Creating the Topology.....	126
Creating a Context	126
Creating a Data Server	126
Testing a Data Server Connection.....	128
Creating a Physical Schema.....	128
Creating a Logical Schema.....	129
Creating a Physical Agent.....	129
Creating a Logical Agent.....	130
Automatically Reverse-engineering Datatypes.....	130
Managing Repositories	130
Connecting to the Master Repository.....	130
Creating the Master Repository	131
Creating a Work Repository	131
Exporting the Master Repository.....	132
Importing the Master Repository.....	133
Changing the work repository password.....	134
Managing Hosts & Usages	134
Definitions.....	134

Default behaviour	135
Administrating Hosts & Usages.....	135
Managing Agents.....	136
Launching a Listener Agent	136
Launching a Scheduler Agent.....	138
Displaying Scheduling Information.....	140
Launching a Web Agent.....	140
Stopping an Agent.....	142
Load Balancing	143
Other Operations	144
Encrypting a Password	144
Topology: Generating a report.....	144
Setting User Parameters.....	145
Import/Export.....	145
Security Manager	149
Introduction to Security Manager.....	149
Security Manager's Interface	149
Defining the Security Policy.....	150
Security Concepts.....	150
Security Policies.....	152
Creating Profiles	153
Creating a New Profile	153
Deleting a Profile.....	153
Creating Users.....	153
Creating a New User.....	153
Assigning a Profile to a User.....	153
Removing a Profile from a User.....	153
Deleting a User	154
Managing Rights.....	154
Assigning an Authorization by Profile or User.....	154
Assigning an Authorization by Object Instance.....	154
Deleting an Authorization by Object Instance.....	155
Deleting an Authorization by Profile or User.....	155
Cleaning up Unused Authorizations.....	155
Using Security Manager	156
Defining the Password Policy	156
Exporting an Object.....	156
Importing an Object.....	157
Setting User Parameters.....	157
Use Oracle Data Integrator with	159

Oracle	159
Creating an Oracle Data Server.....	159
Creating an Oracle Physical Schema	160
Creating and Reverse-Engineer an Oracle Model.....	161
Choosing the Right KMs for Oracle	162
Common Errors with Oracle.....	164
DB2 for iSeries.....	166
Creating a DB2/400 Data Server	166
Creating a DB2/400 Physical Schema.....	168
Creating and Reverse-Engineering a DB2/400 Model.....	169
Using CDC on iSeries	170
Choosing the Right KMs for DB2/400	174
Common Errors with DB2/400	176
Installing the Java Agent on iSeries and AS/400.....	178
Excel	182
Creating a Microsoft Excel Data Server	182
Creating a Physical Schema for Microsoft Excel	184
Creating and Reverse-Engineering a Microsoft Excel Model	184
Choosing the Right KMs for Microsoft Excel.....	185
Common Errors with Microsoft Excel.....	186
File	187
Creating a File Data Server.....	187
Creating a Physical Schema for a File.....	188
Creating and Reverse-Engineering a File Model	189
Choosing the Right KMs for Files	192
Reverse-engineering a COBOL Copybook.....	193
Reverse-engineering a Fixed File using the Wizard	193
JMS.....	194
Creating a JMS Data Server	194
Choosing the Right KMs for JMS	195
Creating a Physical Schema for JMS	197
Defining a JMS Model	198
JMS Standard Properties	199
Using JMS Properties	201
JMS XML	202
Choosing the Right KMs for JMS XML	202
Creating a JMS XML Data Server.....	205
Creating a Physical Schema for JMS XML.....	208
Creating and Reverse-Engineering a JMS XML Model	209
XML.....	210

Creating an XML Data Server	210
Creating a Physical Schema for XML	212
Creating and Reverse-Engineering a Model for an XML file	212
Choosing the Right KMs for XML files	214
Common Errors with XML	216
Glossary	219

This guide describes how to work with the graphical components that make up the Oracle Data Integrator graphical user interface. It guides you through common tasks and worked examples of development in Oracle Data Integrator. It includes conceptual and background information on the features and functionalities of Oracle Data Integrator. This guide is aimed at developers and administrators who want to use Oracle Data Integrator as a development tool for their integration processes.

Document Structure

This document is divided up as follows.

- **Chapter 1 - Concepts** provides conceptual information about integration and introduces the general concepts of Oracle Data Integrator.
- **Chapter 2 through 5** describe how to carry out common tasks with Oracle Data Integrator's graphic modules.
- **Chapter 6 - Using Oracle Data Integrator with ...** gives specific advice for using Oracle Data Integrator with certain supported technologies.

Introduction

What is Oracle Data Integrator?

Oracle Data Integrator streamlines the high performance movement and transformation of data between heterogeneous systems in batch, real time, synchronous, and asynchronous modes; it dramatically enhances user productivity with an innovative, modularized design approach and built-in connectivity to all major databases, data warehouse appliances, analytic applications and SOA suites.

Oracle Data Integrator includes graphical modules and software agents that allow you to:

- Reverse engineer application models.
- Check data consistency.
- Design, test, operate and maintain interfaces between applications.
- Check the data flows processed by the interfaces, with error isolation and/or recycling.
- Identify missing data input.

Who is Oracle Data Integrator designed for?

Most of the **Oracle Data Integrator** modules are designed for computer specialists (project managers, programmers, data administrators, etc), who are involved in an application interfacing project. These are people who:

- Have a complete functional knowledge of all source and/or target applications
- Are responsible for operating interfaces

Why Oracle Data Integrator?

IT solutions dedicated to a job function are becoming more frequent in companies.

In this context, data must be synchronized throughout the company Information System.

With **Oracle Data Integrator**, you can:

- Increase the productivity of projects involving data transfer and/or transformation. These projects include migration, installation of packages, construction of secured databases for the Internet, installation of datawarehouse and datamarts, and synchronization of heterogeneous applications.
- Improve the quality of the company's data, in both old IT applications and new ones.
- Improve inter-application dialog by having interfaces executed asynchronously.

Interfaces

An interface consists of a set of rules that define the loading of a target datastore from one or more sources. An integration project is made up of a group of interfaces.

Constructing interfaces

The world of computer technology is constantly evolving and companies must adapt rapidly to developments in their environment. It is vital to implement regularly projects that aim to:

- Migrate applications to an ERP system,
- Keep a link between the ERP and applications that cannot enter the ERP,
- Load an Operational Data Store (ODS) to provide fresh, integrated information,
- Load a datawarehouse to keep the data organized and be capable to answer rapidly to all users' questions,
- Load Datamarts to provide a display that is better adapted to users in terms of rapidity and access tools,
- Load databases for information to be published on the Web,
- Load production databases from data entered over the Internet (by sales forces, agencies, suppliers, customers and third parties), that strictly respect security constraints,
- Load test databases from production databases.

The above projects all different kinds of integration projects. They require different solutions and updating methods, the development of which can generate an unexpected workload.

Maintaining interfaces

The volume of code and the weight of documentation of an interface makes it hard to maintain. This may restrain its development and evolution. However, scalable maintenance is an integral part of an project life cycle.

Replication and Transformation

Data replication

Replication is a relatively simple technological process sometimes integrated in the RDBMS (Relational Data Base Management System) and/or the Software Engineering Workbench.

Replication allows you to manage the data flow between two similar data models, with the same number of tables and the same data validation rules. The data consistency constraints being the same on both source and target models, data errors are not handled. The number of rows or records is the same in both source and target, making it impossible to have, for example, a history level of 18 months on the source and 36 months on the target.

Creation of a replication interface does not require detailed knowledge of the application. The resources most commonly implemented are: file copying, logging on a mirror base, Snapshots (Oracle), etc.,.

Data transformation

The concept of data transformation makes it possible to have a real difference in data modeling between the source and the target. These differences can occur on several levels:

- the type of data model being radically different between the source and the target (star, snowflake, normalized, etc),

- integrity constraints (declared in the RDBMS if necessary), involving an error processing and recycling procedure,
- storage technologies (files, RDBMS, etc),
- logging level (for example, 18 months on the source and 36 months on the target).

This difference in modeling is normal in datawarehouse, datamart and migration projects. Different modeling implies different data validation rules and therefore potential errors on the transferred data.

The types of transformation carried out by an interface are:

- Mapping, which allows target information (a table column) to be mapped using a calculation rule (numeric calculation, chained calculation, concatenation, aggregation, etc), on source information possibly coming from different environments.
- Filters, which allow only functionally useful records to be returned, for example, a filter on the customer file that returns only active customers.
- Incremental or asynchronous loading, allowing procedures to be speeded up by avoiding the regular transfer of unchanged data.
- Artificial id codes (sequences) may need to be created when re-designing a data model. This type of id code is used in both migrations and Data Warehouse projects for star modeling.

Interface Typology

In terms of the frequency of execution, the type of flow handled and the launching mode, several types of interfaces exist. An interface can change its typology during its life cycle. Without an interface tool, such an action requires the complete rewrite of the interface. The terms described below are not exclusive, an interface can belong to several classifications simultaneously.

One Shot interfaces

One shot interfaces are executed once only. These interfaces are used for application migrations, just before putting into production.

Link interfaces

Link interfaces are used frequently, as they enable two IT applications to be linked constantly.

Asynchronous interfaces

These are link interfaces that work in tight flow. This type of interface handles all events in the source application almost instantaneously. This type of interface is based on MOM (Message Oriented Middleware) or requires a polling action on the source application.

Batch interfaces

Batch interfaces work in loose flow on a time-based frequency, usually daily.

Cancel and replace interfaces

This type of batch interface is the simplest to set up, it consists of deleting the target information in order to completely reload it on each execution. This is recommended for One Shot interfaces. For link interfaces, it has the following disadvantages:

- The target database history must be the same as that of the source database, which is not often the case with Data Warehouses.
- Deleting the data before reloading makes it impossible to declare any referential integrity (foreign key) constraints on the target.

- Network traffic is considerable.

Interfaces by increments

These are asynchronous link interfaces that only process the information that has been changed in the source system. These interfaces work from an activity log, a last update date, or any information that isolates a batch that has been changed since the previous procedure.

This type of asynchronous interface is to be preferred in order to minimize the volumes of information to process. It improves execution time on the servers and reduces network traffic.

Interfaces by difference

This type of interface is similar to interfaces by increments, it uses the differences between source data and target data to determine changes. This type of mechanism is useful for reducing the number of updates and/or to date the information on its entry into the Data Warehouse.

Others

As far as interfaces are concerned, there are many needs that are quite specific to an information system's standards. For example, some sites wish to handle logical deletions in target databases in order to retain the date that source information disappears, others wish to set up high level standards for time stamps and logs. It is therefore important to have an interface system capable of integrating specific replication mechanisms.

Storage Technologies

For historical and strategic reasons, most information systems comprise different data storage technologies. This fact poses the following problems:

- Interface development teams must have multiple skills.
- Converting the internal representation of data is costly in terms of execution performance.
- Converting logical data formats (dates, numbers, etc) is costly in terms of productivity, notably in case of error in input data, as often occurs with dates.
- Managing long transactions (with failure recovery) brings several technologies into play.
- Performance deteriorates through the use of Middleware and/or a standard data access language (SQL 92, ODBC, etc).
- The inconsistency of data from several different systems (for example, between the customer file and the invoice file) gives rise to isolation or recycling of errors.

The most widely used storage technologies are files and relational databases. However, upstream from an interface, inherited technologies (database networks, hierarchies, etc) are often used during migration. Downstream, new storage formats are found more and more often on decision systems, mainly multidimensional systems (OLAP) and multiple indexing. To the extent that an interface construction tool supervises all the data flows of a company, it must be capable of adapting rapidly to new technologies.

It is not always advisable to access information storage support directly. Thus, many software programs provide communication APIs for integrating or extracting consistent information without worrying about its mode of storage.

The table below details the **Oracle Data Integrator** Access Methods to different storage support

Support	Access Method
---------	---------------

Fixed file / Delimited (ASCII / EBCDIC)	Native
XML file in output	Native
XML file in input	Native
Data server	JDBC/ODBC
Message Server (MOM)	JMS
LDAP Directory	Native
programs' APIs	JCA
Others	Utility program for loading and unloading via an exchange file

Data Quality

Data Quality in Oracle Data Integrator

Checking transferred data has many advantages:

- Productivity gains when using the data. Obviously, data errors slow down not only the development of applications, but also their operation over long periods.
- Validating the modeling. The errors detected do not always arise from insufficient source data quality, but can reveal an incompleteness of the model. Migrating data before rewriting an application makes it possible to validate a new data model and provides a test set close to reality.
- Improved service quality for the end user.

Ensuring data quality in terms of interface construction and operation is not simple. In fact, it calls for the management of the isolation and recycling of erroneous data, implying more complex programming, particularly when the target incorporates an activated mechanism for checking integrity constraints. During operation, a procedure for correcting erroneous data (on the source, the target, or on the recycled flows) should be implemented.

Checking data flows

Data flow control consists of checking the source data of an interface compared to the consistency level required on the target. Data is checked before the data is integrated in the target. Detection of an error can give rise to either non-integration of all the data (even correct), or integration of only the correct data. In any case, it is useful to isolate and/or recycle the erroneous data (after any necessary correction). The recycling and correction of isolated errors only makes sense for increment loadings, as, in this case, the data is not automatically returned on each interface execution.

Simulating this type of procedure can be useful, as this remains the only means of checking a set of source data in comparison with the integrity rules declared on another data model. A simulation consists of launching an interface without updating the data in the target structure.

Static data control

The consistency of data in a model, irrespective of its use in an interface, can be checked to assess the time needed for correcting errors, and also to carry out a **quality audit of an application**.

Of course, data quality control can only apply to the rules not present in the storage technology containing the data. The rules not checked by this source technology should be declared in another metadata dictionary: semantic link, validity limits, validity domain, formats.

The quality of the target data

The quality of the target data can always be called into question, if the target technology does not check its data by an internal mechanism (triggers or declarative integrity). Data consistency will be validated by program or by SQL queries. To avoid these repetitive manual checks, it is important to be able to automate this type of check in the same way as for sources.

Processing erroneous data

Erroneous data can be corrected either on the source or on the target (if the errors have been integrated), or on the recycled flows. The use of a central tool for consulting and, if necessary, correcting the data wherever it is located in the information system, can be very useful in a multi-technology context.

The capacities of data quality control

Most quality control features integrated in **Data Integrator** are only applicable on relational technologies.

These functions are:

- Checking flows according to the target database rules,
- Checking static data (on the source or on the target),
- Recycling and/or isolation of errors,
- Consulting and editing (for manual correction) non-integrated erroneous data (flow or cleanup),
- Consulting and editing (for manual correction) erroneous data in the source and/or target tables,
- Checking joins (referential integrity), validity limits, validity domains, primary and alternate keys and complex rules,
- Recycling and isolation of errors,
- Locating errors.

Oracle Data Profiling and Data Quality for Data Integrator

Oracle Data Profiling and **Oracle Data Quality for Data Integrator** (also referred to as **Oracle Data Quality Products**) extend the inline Data Quality features of Oracle Data Integrator to provide more advanced data governance capabilities.

Oracle Data Profiling

Oracle Data Profiling is a data investigation and quality monitoring tool. It allows business users to assess the quality of their data through metrics, to discover or deduce rules based on this data, and to monitor the evolution of data quality over time.

Oracle Data Quality for Data Integrator

Oracle Data Quality for Data Integrator is a comprehensive award-winning data quality platform that covers even the most complex data quality needs. Its powerful rule-based engine and its robust and scalable architecture places data quality and name and address cleansing at the heart of an enterprise data integration strategy.

More Information

For more information on Oracle Data Profiling and Oracle Data Quality for Data Integrator, please refer to the following documents available with Oracle Data Quality Products:

- *Getting Started with Oracle Data Quality for Data Integrator Guide*
- *Oracle Data Quality Tutorial*
- *Oracle Data Quality Products User's Guide*
- See also the Working with Oracle Data Quality topic

Productivity

Interfaces require a workload varying from a few days to several months. In the past, some data migrations were rapidly implemented between files, but today, with the multiplication of technologies (DBMS, MOM, LDAP, XML, etc), creating an interface is an operation that sometimes takes much longer than expected. Where is the progress? What are the elements that generate this load? By what means can this be remedied?

The evolution of RDBMS

On RDBMSs, the main development concerns the management of erroneous data. Relational databases refuse the input of any data that doesn't conform to the application model. The mechanism put in place is the implicit check of different types of integrity rules: datatype (date, currency, etc), primary keys, alternate keys, foreign keys (link between tables), and complex rules. In this context, it is difficult to consider only one part of the model when constructing an interface. For example, every loading of an entity "Invoices" requires the prior loading of the entities "Customers", "Payment methods" and other parameter tables. Finally, it must be noted that this mechanism has an impact on the organization of projects. In fact, construction and testing each interface should take place according to the order of the data model's functional dependencies.

Moreover, interfaces load more open systems like Data Warehouse, in which the smallest data error generates grossly incorrect results. These errors are difficult to detect by testing the target application, as the data is regularly added to. In addition, the user constructs his information requests himself with powerful access tools. This type of project requires all input data to be systematically checked in relation to a semantic model comprising all the data consistency rules.

The democratization of data access tools to people in the field makes it necessary to load decision databases on an everyday basis. This higher frequency level requires interfaces with faster execution and which don't block on data errors. Programming this type of interface is much more complex than a monthly cancel-and-replace-type loading without data quality control.

The evolution of technologies and needs

Companies need to open up to the outside world and streamline the dialog between their applications, in order to respond more rapidly to external requests (customers, suppliers, etc). To cover these needs, the market has provided companies with new technologies (MOM, XML, LDAP directories) requiring new access methods and new skills.

In this context, interface construction becomes technically complex and requires many different skills.

The importance of the right tool

Interfaces are driven by several types of rules (mapping, transformation, aggregation, filter, etc) used repetitively; the same is true of their programming and their mode of operation and error management. The absence of integrated tools for supporting the design, programming, documentation and maintenance of these rules leads to repetitive interventions on each of them, and at all levels (design, documentation, programming, operation, maintenance). A tool that stores the management rules of an interface, and which is capable of interpreting them, reduces the documentation and programming phases by more than 90%. Moreover, it removes the supplementary load induced by most of the interface characteristics.

With a tool, rule maintenance is a very isolated operation, usually requiring only one person's intervention. Thus, an interface written to do cancel-and-replace (deletion, then re-creation of the target on each execution) must be totally rewritten if a higher logging level is to be added in the target database (for example, 5 years on the target database and 18 months on the source). With a tool that stores these rules, all that needs to be done is to modify the type of interface without changing the transformation rules (mapping, calculations, filters, aggregation, etc).

Knowledge Modules

What are 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.

Combined with a connectivity layer such as JDBC, JMS, JCA or other, KMs define an Open Connector that performs defined tasks against a technology, such as connecting to this technology, extracting data from it, transforming the data, checking it, integrating it, etc.

Open Connectors contain a combination of:

- Connection strategy (JDBC, database utilities for instance).
- Correct syntax or protocol (SQL, JMS, etc.) for the technologies involved.
- Control over the creation and deletion of all the temporary and work tables, views, triggers, etc.
- Data processing and transformation strategies.
- Data movement options (create target table, insert/delete, update etc.).
- Transaction management (commit/rollback), depending on the technology capabilities.

Different types of Knowledge Modules

Oracle Data Integrator' Open Connectors use 6 different types of Knowledge Modules:

- **RKM (Reverse Knowledge Modules)** are used to perform a customized reverse-engineering of data models for a specific technology.
- **LKM (Loading Knowledge Modules)** are used to extract data from the source database tables and other systems (files, middleware, mainframe, etc.).
- **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.
- **IKM (Integration Knowledge Modules)** are used to integrate (load) data to the target tables.
- **CKM (Check Knowledge Modules)** are used to check that constraints on the sources and targets are not violated.
- **SKM (Service Knowledge Modules)** are used to generate the code required for creating data services.

How does it work?

At design time

When designing Interfaces in Oracle Data Integrator, these Interfaces contain several phases, including data loading, data check, data integration, etc. For each of these phases, you will define:

- The functional rules (mappings, constraints, etc.) for this phase
- The Knowledge Module to be used for this phase. You can configure this Knowledge Module for this phase using its options.

At run time

Oracle Data Integrator will use the functional rules, Knowledge Module, Knowledge Module options and the metadata contained in the Repository (topology, models, etc.) to generate automatically a list of tasks to process the job you have defined. Tasks include connection, transaction management and the appropriate code for the job. These tasks will be orchestrated by the Agent via the Open Connectors and executed by the source, target and staging area servers involved.

Customization of Knowledge Modules

Beyond the KMs that are included with Oracle Data Integrator and cover most standard data transfer and integration needs, Knowledge Modules are fully open - their source code is visible to any user authorized by the administrator. This allows clients and partners to easily extend the Oracle Data Integrator Open Connectors to adjust them to a specific strategy, to implement a different approach and integrate other technologies.

Knowledge Modules can be easily exported and imported into the Repository for easy distribution among Oracle Data Integrator installations. At the same time, Oracle Data Integrator allows partners and clients to protect their intellectual property (for example a specific approach, an advanced use of certain technologies) by giving the option of encrypting the code of their KMs.

Designer

Introduction to Designer

Through the **Designer** module, you can handle:

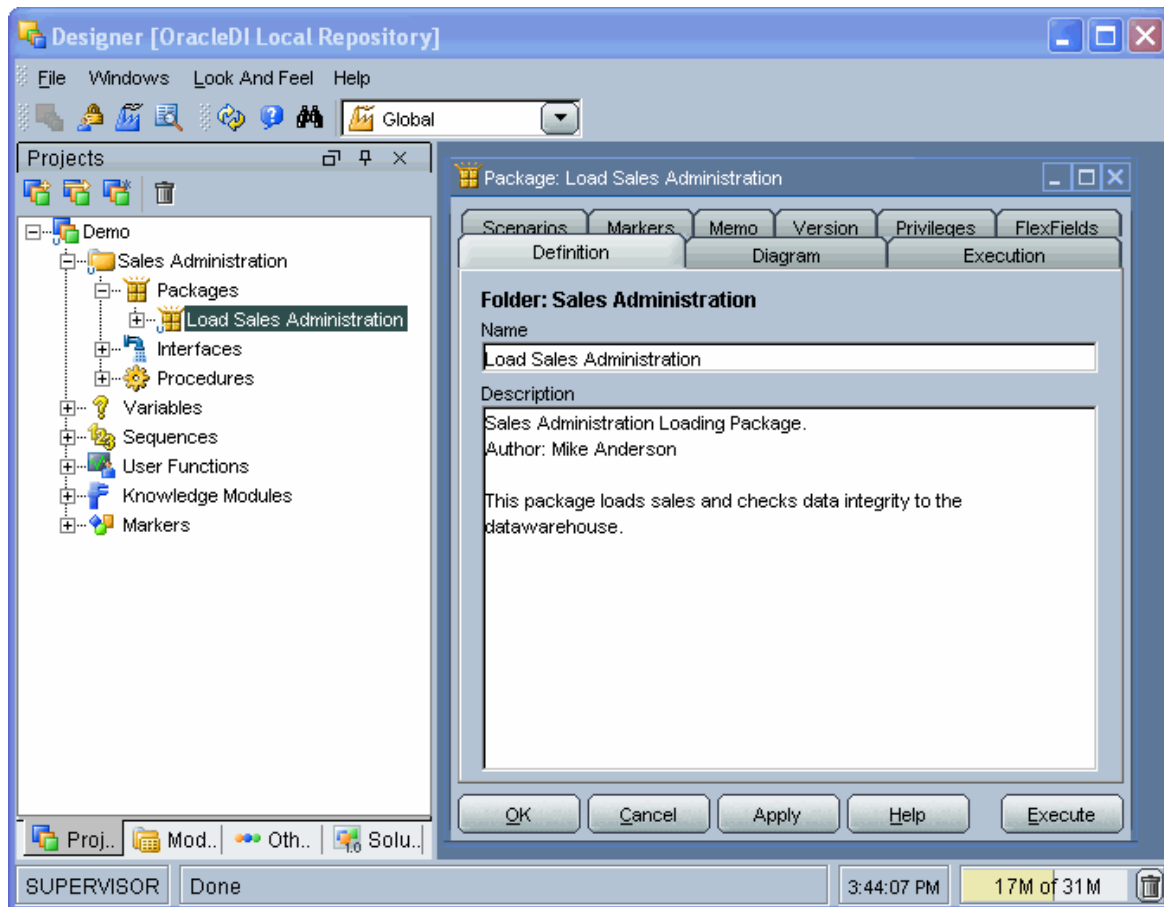
Models: Descriptions of the data and applications structures

Projects: The developments made with Designer.

The Designer module stores this information in a work repository, while using the topology and the security information defined in the master repository.

Designer's Interface

The Designer GUI appears as follow:



The Menu

The **Menu** contains pull-down menus to access the following features:

- Import/Export
- Wizards

- Display options
- Open modules or tree views
- Change the user's password and options

The Toolbar

The **Toolbar** lets you:

- Open other modules
- Refresh the Tree views
- Open the on-online help
- Choose the default **context**

The **context** selected is used as default for all context choices to be made in the application windows. In addition, when data is consulted (right-click on data on a datastore), Designer displays the data in the context defined in the toolbar. For example, if your context is "Development", you consult the data in the "development" context. As a safety precaution, even if you are authorized for all contexts, you are asked for a confirmation password at every change of context, to avoid any inappropriate manipulation. Only the contexts you are granted appear in the menu bar.

The Tree Views

Designer objects available to the current user are organized into the following tree views: **Projects**, **Models**, **Solutions** and **Others** (User Functions, Global Variables and Sequences).

Each tree view appears in a floatable frames that may be docked to the sides of the main window. These frames can be also be stacked up. When several frames are stacked up, tabs appear at the bottom of the frame window to access each frame of the stack.

Tree view frames can be moved, docked and stacked by selecting and dragging the frame title or tab. To lock the position of views, select **Lock views** from the **Windows** menu.

If a tree view frame does not appear in the main window or has been closed, it can be opened using the **Windows > Show View** menu.

It is possible in each tree view to perform the following operations:

- Insert or import root objects to the tree view by clicking the appropriate button in the frame title
- Expand and collapse nodes by clicking on them
- Activate the methods associated to the objects (Edit, Delete, ...) through the popup menus
- Edit objects by double-clicking on them, or by drag and dropping them on the **Workbench**

The Workbench

The windows for object being edited or displayed appear in the **Workbench**.

Handling Models

What is a Model?

A **model** is a set of **datastores** corresponding to data structures contained in a physical schema. Models can be organized into **model folders**.

- See Creating and Reverse-engineering a Model

Model Folders

A **model folder** is an object which groups models together. For example, you can group all models based on a particular technology, located at a particular site or used in a particular project.

Sub-models

A **sub-model** is an object used to organize and classify the datastores of a model into a hierarchical structure. The root of the structure is the model itself.

Reverse-engineering

A model is created without any datastores in it. **Reverse-engineering** a model allows you to automatically retrieve data structures to define the model's datastores in Oracle Data Integrator. There are two different modes:

- **Standard** reverse-engineering uses standard JDBC features to request the metadata.
- **Customized** reverse-engineering uses a technology-specific Reverse Knowledge Module (RKM) to retrieve the metadata, using a method specific to the given technology.

Datastore

A **datastore** describes data as a table structure. Datastores are composed of **columns**.

Datastores are defined in Oracle Data Integrator in a relational model. Therefore, it is possible to attach the following elements to a datastore:

Keys

A **key** is a set of datastore columns that enables each datastore row to be uniquely identified. If it is also an index, it may also allow row access to be optimized. Some drivers retrieve key descriptions during the reverse-engineering process. It is also possible to define keys directly in the repository.

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

A **condition** or a **filter** is a WHERE-type SQL expression attached to a datastore based on any RDBMS that supports SQL. They serve to filter or check data in the datastore.

Journalizing

Journalizing keeps track of changes to data. Journalizing is used in Oracle Data Integrator to eliminate the transfer of unchanged data. This feature has many uses, such as in data synchronization and replication.

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

Creating and Reverse-Engineering a Model

Create a Model

A **Model** is a set of datastores corresponding to data structures contained in a Physical Schema. A model is always based on a **Logical Schema**. In a given **Context**, the **Logical Schema** corresponds to a **Physical Schema**. The Data Schema of this Physical Schema contains the physical structure: tables, files, JMS messages, elements from an XML file, that are represented as datastores.

Important Note : Frequently used technologies have their reverse and model creation methods detailed in the section named Using Oracle Data Integrator with Please refer to this section before proceeding.

To create a Model:

1. Connect to **Designer**
2. Select **Models** in the tree
3. Right-click then select **Insert Model**.
4. In the **Definition** tab, fill in the **Name** field.
5. In the **Technology** field, select the model's technology.
6. In the **Logical Schema** field, select the Logical Schema on which your model will be based.
7. Go to the **Reverse** tab, and select a **Context** which will be used for the model's reverse-engineering. Click **Apply**.

The model is created, but contains no datastore yet.

Reverse-engineer a model

A model is created without any datastores. **Reverse-engineering** a model automatically retrieves data structures to define datastores for the model. There are two different modes:

- **Standard** reverse-engineering uses standard JDBC features to retrieve the metadata.
- **Customized** reverse-engineering uses a Reverse Knowledge Module (RKM) to retrieve the metadata, using a method defined specifically for the given technology.

Note: The Customized reverse-engineering is more complete and entirely customizable, but also more complex to run. RKM are provided only for certain technologies.

Standard Reverse-Engineering

To perform a Standard Reverse- Engineering:

1. Go to the **Reverse** tab of your **Model**.
2. Fill in the following fields:
 - **Standard**

- **Context:** Context used for the reverse-engineering
 - **Types of objects to reverse-engineer:** List of object types that should be taken into account by the reverse-engineering process.
3. Go to the **Selective Reverse** tab.
 - Check the **Selective Reverse**, **New Datastores**, and **Objects to Reverse** boxes
 4. A list of datastores to be reverse-engineered appears. Leave those you wish to reverse-engineer checked.
 5. Click the **Reverse** button, then on **Yes** to validate the changes.
 6. Oracle Data Integrator launches a reverse-engineering process for the selected datastores; A progress bar appears.

The reverse-engineered datastores appear under the model.

It is possible to refine the reverse-engineer using the options in the **Reverse** and **Selective Reverse** tabs. Refer to Model for more information.

Customized Reverse-Engineering

To perform a Customized Reverse-Engineering using a RKM:

Warning ! Using a RKM requires the import of this RKM into a project. Only the imported RKM are available for reverse operations.

1. Go to the **Reverse** tab of the **Model**.
2. Fill in the following fields:
 - **Customized**
 - **Context:** Context used for the reverse-engineering
 - **Object Type:** Type of the objects to reverse-engineer (tables, view...)
 - **Mask:** %
 - **Select the KM:** RKM <technology>.<name of the importation project>
3. Click the **Reverse** button, then **Yes** to validate the changes.
4. Click **OK**
5. The **Session started** window appears. Click **OK**

Follow and validate your model's reverse-engineering operations in the Oracle Data Integrator Log. If it finished correctly, the reversed datastores appear under the reversed module in the tree. It is possible to refine the reverse-engineer using the options of the **Reverse** and **Selective Reverse** tabs. Refer to Model for more information. Also see the RKM description for more information.

Organizing Models into Model Folders

Models can be organized into **Model Folders**. A model folder may also contain other model folders.

To create a model folder:

1. Open the **Models** view.
2. Right-click, then select **Insert Model Folder**.
3. Enter the **Name** of the folder and a detailed **Description**.
4. Click **OK**.

The empty model folder appears.

To move a model into a folder:

1. Open the **Models** view.
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: A model can only be in one folder at a time.

Note: Model Folders can be also moved into other model folders.

Note: It is possible to use also markers to organize models.

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.

The classification is performed :

- During the reverse-engineering, RKM may create sub-models and insert datastores into these sub-models.
- Manually, using datastore drag and drop into the models,
- Automatically, using the classification based on the model's name.

To Create a Sub-Model:

1. Select a **Model** or a **Sub-Model** in the tree view.
2. Right click and select **Insert Sub-Model**.
3. In the **Definition** tab, fill in the **Name** field.
4. Click **OK**.

The new sub-model is created with no datastore.

To manually file a datastore into a sub-model :

1. Select the **Datastore** in the tree view then drag and drop it into the **Sub-Model**.

The datastore disappears from the model and appears in the sub-model.

To set up the automatic distribution of the datastores in a Sub-Model:

1. Go to the **sub-model's distribution** tab.
2. Fill in the following fields:
 - **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.
 - **Automatic Distribution of all Datastores not classified...:** 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.
 - **Automatic Assignment Mask:** Pattern the datastores names must respect to be classified into this sub-model.

- **Mask application order after a reverse:** At the end of a reverse, all rules are applied in the mask application order. 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.
3. Click **OK**.


Creating a Datastore

To create a datastore:

1. Select a **Model** or a **Sub-Model** in the tree view.
2. Right click and select **Insert 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
 - **Resource Name:** Name of the object in the form recognized by the data server which stores it. This may be a table, file, JMS Queue, ... name .
 - **Alias:** This is a short name used in check and filter expressions.
4. If the datastore is created in a **file** model:
 1. Go to the **file tab**.
 2. Fill in the fields in this tab. For more information, refer to Datastore.
 3. If the datastore is a **delimited** file, go to the **columns** tab, then click the **reverse** button to reverse-engineer the list of columns.
 4. If the datastore is a **fixed** file, you can use the Wizard for defining its columns
 5. If you have a Copybook COBOL file describing the file structure, refer to Reverse a COBOL Copybook.
5. Click **OK**.


The datastore is created. Unless you have performed one of the file-specific reverse-engineering, it contains no columns.

To add columns to a datastore :

1. In the **columns** tab of the **datastore**, click 
2. An empty line appears. Fill in the information about the columns. Fields depend on the datastore type. Refer to Datastore for more details on these fields.
3. Repeat steps 1 and 2 for each column you want to add to the datastore.
4. Click **OK**.

Note: For a fixed file datastore, it is possible to use the **Automatic adjustment** option that computes automatically the columns start positions in order to create columns faster.

To delete columns from a datastore :

1. In the **columns** tab of the **datastore**, select the column to delete.
2. Click . The column disappears from the list.

Editing and Viewing a Datastore's Data

To view a datastore's data:

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

The data appear in an editable grid. The **SQL** button enables you to change the datastore data display query and to perform free queries on the datastore.

Note: The data displayed are those stored in the physical schema corresponding to the model's logical schema, in the current context indicated in the Designer's toolbar.

Note: 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.).

Common Format Designer

Introduction

Common Format Designer

Common Format Designer (CFD) is used to quickly design a data model from the Designer user interface. 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 in a data server.

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

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

Designing a Data Model

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, then:
 - assembling in these datastores columns from other datastores,
 - creating new columns in these datastores.

Why assemble datastores and columns from other models?

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

Automatic interface generation does not work to load datastores and columns that are not created from other model's datastores and columns. It is still possible to create the integration interfaces manually, or complete generated interface for columns 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, columns, 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.

To create a new diagram:

1. In the models view, expand the data model, then select the **Diagrams** node.
2. Right-click, then select **Insert Diagram**.
3. Type in the **Diagram Name**, and the **Description**.

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

To insert an existing datastore in a diagram:

1. In the Diagram edition window, select the **Diagram** tab.
2. Select the datastore from a model
3. Drag this datastore to 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.

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

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

To create a new datastore in a diagram:

1. In the Diagram window, select the **Diagram** tab.
2. In the tool bar, click the **Add Datastore**  button.
3. Click the diagram workbench.
4. The new datastore window appears. For other operations on the datastore (creating new columns, adding keys, etc.) please refer to Creating a Datastore.

To add columns from other datastores:

1. In the Diagram edition window, select the **Diagram** tab.
2. Select a column from a datastore
3. Drag this column to a datastore in the diagram.
A datastore window appears with the new column that will be added to this datastore
4. Click **OK**. The new column appear in the datastore.

To create a graphical synonym for a datastore:

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

The new graphical synonym appears in the diagram.

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

To add columns, condition, filters, keys to a datastore:

1. In the Diagram, select the datastore.
2. Right-click then select Add Key, Add Filter, etc,

To add an existing condition, reference or filter to a diagram:

1. Drag and drop an existing condition, reference or filter onto the diagram.


The datastore to which the condition, reference or filter is attached must already exist in the diagram. This is useful if you have created these objects after adding the datastore to the diagram.

To edit a key on a column:

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

1. In the Diagram, select the column.
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 edition window, select the **Diagram** tab.
2. In the toolbar 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 new reference window appears.
5. Set this reference's parameters, then click **OK**.

To move a reference to another graphical synonym:

1. In the Diagram, select the reference.
2. Right-click and select **Display Options**.
A **Display Options** window appears.

3. Select the synonyms to be used as the parent and child of the reference.
4. Click **OK**.
The reference representation appears now on the selected synonyms.

Note: This operation does not changes the reference. It alters only its representation in the diagram.

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 meta-data 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 by generated 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 by on the data server to impact the changes.

Note: The templates for the DDL scripts are defined as Action Groups. Check in Topology Manager that you have the appropriate action group for the technology of the model before starting DDL scripts generation.

To generate the DDL scripts:

1. Select the data model you want to generate the DDL scripts for.
2. Right-click, then select **Generate DDL**.
Oracle Data Integrator retrieves the data structure from the data schema, and compares it to the model definition. The progression is displayed in the status bar. The **Generate DDL** window appears, with the detected differences.
3. Select the **Action Group** to use to generate the DDL script.
4. Click the ... button to select the folder into which the procedure will be created.
5. Filter the type of changes you want to display using the **Filters** check boxes.

Note: These filters may be changed at any time, and apply only to the display. A change already selected may not be visible due to a filtering option.

6. Select the changes you want to apply by checking the **Synchronization** column checkbox. 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).
7. Click **OK** to generate the DDL script.

Oracle Data Integrator generates the DDL scripts, and opens the procedure containing the DDL commands.

Generate Interface IN/OUT

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

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

Example: An AIH assembles bits of information coming from several other applications. It is made up of composite datastores built from several data models, that are assembled in a diagram. The AIH is loaded using the Interfaces IN, and is able to send its data to the original systems using the Interfaces OUT.

To generate the Interfaces IN:

1. Select a data model or datastore.
2. Right-click, then select **Generate Interfaces IN**.
Oracle Data Integrator looks for the original datastores and columns used to build the current model or datastore.
A **Generate Interfaces IN** window appears with a list of datastores for which Interfaces IN may be generated.
3. Select the **Optimization Context** for your interfaces.
4. Click the ... button to select the folder into which the interfaces will be generated.
5. In the **Candidate Datastores**, check the **Generate Interface** checkbox for the datastores to load.
6. Edit the content of the **Interface Name** column to rename the integration interfaces.
7. Click **OK**.
Interface generation starts.

The generated interfaces appear in the specified folder.

Warning! Interfaces automatically generated are built using the available metadata and do not always render the expected rules. These interfaces must be carefully reviewed and modified before execution.

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

To generate the Interface OUT:

1. Select a data model or datastore.
2. Right-click, then select **Generate Interfaces OUT**.
Oracle Data Integrator looks for the existing Interfaces loading these the datastores.
A **Generate Interfaces OUT** window appears with a list of datastores for which Interfaces OUT may be generated.
3. Select the **Optimization Context** for your interfaces.
4. Click the ... button to select the folder into which the interfaces will be generated.
5. In the Candidate Datastores, check the **Generation** and **Generate Interface** checkboxes to select either all or some of the candidate datastore to load from the target datastore of the existing interfaces.

6. Edit the content of the **Interface Name** column to rename the integration interfaces.
7. Click **OK**.
Interface generation starts.

The generated interfaces appear in the specified folder.

Warning! Interfaces automatically generated are built using the available metadata and do not always render the expected rules. These interfaces must be carefully reviewed and modified before execution.

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

Changed Data Capture

Introduction

Changed Data Capture (CDC) allows Oracle Data Integrator to track changes in source data caused by other applications. When running integration interfaces, 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 documentation on journalizing knowledge modules for more information on the capture processes used.
- **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: Say you need 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 integration interfaces 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.

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

Setting up Journalizing

This is the basic process for setting up CDC on an Oracle Data Integrator data model. Each of these steps is described in more detail below.

1. Set the CDC parameters
2. Add the datastores to the CDC
3. For consistent set journalizing, arrange the datastores in order
4. Add subscribers
5. Start the journals

To set the data model CDC parameters:

This includes selecting or changing the journalizing mode and journalizing knowledge module used for the model. 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.

1. Edit the data model you want to journalize, and then select the **Journalizing** tab.
2. Select the journalizing mode you want to set up: **Consistent Set** or **Simple**.
3. Select the **Journalizing KM** 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.

4. Set the **Options** for this KM. Refer to the knowledge module's description for more information on the options.
5. Click **OK** to save the changes.

To add or remove datastores to or from the CDC:

You should now flag the datastores that you want to journalize. 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.

1. Select the datastore, the model or the sub-model you want to add/remove to/from CDC.
2. Right-click then select **Changed Data Capture > Add to CDC** to add the datastore, model or sub-model to CDC, or select **Changed Data Capture > Remove from CDC** to remove it.
3. Refresh the tree view. 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.

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

Note: 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).

To arrange the datastores in 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 into 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 ORDER table), its order should be lower still.

1. Edit the data model you want to journalize, then select the **Journalized Tables** tab.
2. 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 data models' foreign keys. Note that this automatic reorganization is not error-free, so you should review the suggested order afterwards.
3. 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.
4. Repeat step 3 until the datastores are ordered correctly, then click **OK** to save the changes.

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

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

Note: From this tab, you can remove datastores from CDC using the **Remove from CDC** button

To add or remove subscribers:

This adds or removes a list of entities that will use the captured changes.

1. Select the journalized data model if using Consistent Set Journalizing or select a data model or 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 into the field, then click the **Add Subscriber** button.
4. Repeat the operation for each subscriber you want to add. Then click **OK**.

A session to add the subscribers to the CDC is launched. You can track this session from the Operator.

To remove a subscriber is very similar. 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.

To start/stop 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.

Note: Stopping the journals deletes the entire the journalizing infrastructure and all captured changes are lost. Restarting a journal does not remove or alter any changed data that has already been captured.

1. Select the data model or datastore you want to journalize.
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.

A session begins to start or drops the journals. You can track this session from the Operator.

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.

Typical situation: the developer manually configures CDC in the Development context. After this is working well, CDC is automatically deployed in the Test context by using a package. Eventually the same package is used to deploy CDC in the Production context.

To automate journalizing setup:

1. Create a new package in Designer
2. Drag and drop the model or datastore you want to journalize. A new step appears.
3. Double-Click the step icon in the diagram. The properties panel 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. Click **OK** to save.
8. Generate a scenario for this package.

When this scenario is executed in a context, it starts the journals according to the model configuration and creates the specified subscribers using 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. See the *Packages* section for more information.

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 **Data 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$`. The Database-specific programs are installed separately (see the KM documentation for more information).
- A CDC common infrastructure for the data server is installed in the **Work Schema** for the Oracle Data Integrator physical schema that is flagged as **Default** for this data server. This common infrastructure contains information about subscribers, consistent sets, etc for all the journalized schemas of this data server. This common infrastructure consists of tables whose names are prefixed with `SNP_CDC_`.
- Journal tables and journalizing views are installed in the **Work Schema** for the Oracle Data Integrator physical schema containing the journalized tables. The journal table and journalizing view names are prefixed with the prefixes defined in the **Journalizing Elements Prefixes** for the physical schema. The default value is `J$` for journal tables and `JV$` for journalizing views

Note: 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 Note: 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

Datstores in models or interfaces 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 view the changes captured.

To view the changed data:

1. Select the journalized datastore
2. Right-click, then select **Changed Data Capture > Journal Data**.

A window containing the changed data appears.

Note: This window selects data using the journalizing view.

The changed data displays three extra columns for the changes 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 integration interfaces. The way it is used depends on the journalizing mode.

Using Changed Data: Simple Journalizing

Using changed data from simple journalizing consists of designing interfaces using journalized datastores as sources.

Designing Interfaces

Journalizing Filter

When a journalized datastore is inserted into an interface diagram, a **Journalized Data Only** check box appears in this datastore's property panel.

When this box is checked:

- the **journalizing columns** (**JRN_FLAG**, **JRN_DATE** and **JRN_SUBSCRIBER**) become available in the datastore.
- A **journalizing filter** is also automatically generated on this datastore. This filter will reduce the amount of source data retrieved. 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). A typical filter for retrieving all changes for a given subscriber is: `JRN_SUBSCRIBER = '<subscriber_name>'`.

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

Knowledge Module Options

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.

Using Changed Data: Consistent Set Journalizing

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

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 interface using changes captured for the subscriber(s).

Designing Interfaces

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

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

- The changes taken into account by the interface (that is filtered with `JRN_FLAG`, `JRN_DATE` and `JRN_SUBSCRIBER`) are not automatically purged at the end of the interface. They can be reused by subsequent interfaces. The unlock subscriber and purge journal operations (see below) are required to commit consumption of these changes, and remove useless entries from the journal respectively.
- In consistent mode, the `JRN_DATE` column should not be used in the journalizing filter. Using this timestamp to filter the changes consumed does not entirely ensure consistency in these changes.

Operations after Using the Changed Data

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

- **Unlock Subscribers:** This operation commits the use of the changes that were locked during the **Lock Subscribers** operations for the subscribers. It should be processed only after all the changes for the subscribers have been processed. This operation is implemented using a package step with the **Journalizing Model Type**. It should be always performed after the last interface 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.

To create an Extend Window, Lock/Unlock Subscriber or Purge Journal step in a package:

1. Open the package where the operations will be performed.
2. Drag and drop the model for which you want to perform the operation.
3. In the **Type** list, select **Journalizing Model**.
4. Check the option boxes corresponding to the operations you want to perform.

5. Enter the list of subscribers into the **Subscribers** group if performing lock/unlock subscribers operations.
6. Click **OK**.

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

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:

- **SnpsWaitForData** waits for a number of rows in a table or a set of tables.
- **SnpsWaitForLogData** waits for a certain number of modifications to occur on a journalized table or a list of journalized tables. This tool calls SnpsRefreshJournalCount to perform the count of new changes captured.
- **SnpsWaitForTable** waits for a table to be created and populated with a pre-determined number of rows.
- **SnpsRetrieveJournalData** 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.
- **SnpsRefreshJournalCount** refreshes the number of rows to consume for a given table list or CDC set for a specified journalizing subscriber.

See the *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.

Template 1: One Simple Package (Consistent Set)

- Step 1: Extend Window + Lock Subscribers
- Step 2 ... n-1: Interfaces 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 ... n: Interfaces 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 SnpsWaitForLogData (Consistent Set or Simple)

- Step 1: SnpsWaitForLogData. 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 SnpsStartScen

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 interfaces 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: SnpsWaitForLogData. 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 ... n-1: Interfaces 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 ... n-1: Interfaces 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.

Setting Up Data Services

Data Services are specialized Web Services that provide access to data in datastores, and to captured changes in these datastores. These Web Services are automatically generated by Oracle Data Integrator and deployed to a Web Services container - normally a Java application server.

In the examples below, Apache Tomcat 5.5 or Oracle Container for Java (OC4J) are used as the application server, with Apache Axis2 as the Web Services container. These examples may need to be adapted if using other Web Services containers.

Setting up Data Services involves the following stages :

1. Setting up datastores in a Web Services container.

2. Configuring the topology to declare the Web Services container.
3. Setting up the model, to declare datastores and columns for Data Services.
4. Generating, deploying and testing Data Services

Setting up datasources

Before getting started, you should already have configured your Web Services container and the application server. Refer to the respective documentation for these components for details.

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

In addition, all necessary drivers for the specified datasources must be installed on the application server.

To set up a datasource:

1. Install the driver files (.jar or .zip) for your datasource in the appropriate directory for your application server. Use `/common/lib` for Tomcat. Use `ORACLE_HOME/j2ee/home/applib` for OC4J.
2. Create the JDBC Datasource pointing to the data server you want to access:
 - For Tomcat, Define the datasource and connection properties for your data in the file `WEB-INF/context.xml` on your Web Services container. Note the name of the resource (marked in green below), as you will need it later. Some connection properties can be found in the topology; these are marked in blue. One example is given below.

```
<Context >
  <Resource
    name="jdbc/Oracle_SRV1/Oracle/dataServices"
    type="javax.sql.DataSource"
    driverClassName="oracle.jdbc.OracleDriver"
    url="jdbc:oracle:thin:@SRV1:1521:ORA10"
    username="ODI"
    password="ODI"
    maxIdle="2"
    maxWait="-1"
    maxActive="4" />
</Context>
```

- For OC4J, you must define the datasource as follows:
 1. Connect to OC4J administration interface.
 2. In the **Administration** tab, in **Services | JDBC Resources** click **Go to task**.
 3. Click the **Create** button in the **Connection Pools** section.
 4. Select the Axis2 application, select **New Connection Pool** , then **Continue**.
 5. Fill in the fields for the JDBC datasource then click **Finish**.
 6. Click the **Create** button in the **Data Sources** section.
 7. Select the Axis2 application, select **Managed Datasource** , then **Continue**.

8. Fill in the fields for the JDBC datasource then click **Finish**.
For more information on configuring datasources for OC4J, refer to the application server documentation.
4. Make a reference to the datasource in your Web Services container's `WEB-INF/web.xml` file. Note that you must specify the name of the datasource in the `res-ref-name` tag. One example of referencing the datasources defined above is given below:

```

...
<resource-ref>
    <description>Data Integrator Data Services on
    Oracle_SRV1</description>
    <res-ref-name>jdbc/Oracle_SRV1/Oracle/dataServices</res-
    ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>

```

4. Restart your Axis2 application to take the new datasources into account.

Configuring the topology

Once the Web Services container is ready, you must declare it as a data server in the topology, in order to let Oracle Data Integrator deploy Data Services on it.

Note : Be careful not to confuse the Web Services containers and the servers containing deployed data. While both are declared as data servers in Oracle Data Integrator, the former do not contain any data, and only serve to host the published Data Services.

Web Services 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 supported on some servers, such as Axis2.

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

To configure the Web Services container:

1. In **Topology Manager's Physical Architecture** view, select the **Axis2** technology. Right-click and select **Insert Dataserver**. If you are using a different Web Services container, then choose the appropriate technology instead.
2. Fill in the following fields on the Definition tab:
 - **Name**: Name of the dataserver as it will appear in Oracle Data Integrator.
 - **Base URL for published services**: `http://<Host>:<HTTP port>/axis2/services`
 - Select the option corresponding to the chosen method of deployment:
 - **File copy**: Specify the directory in the application server which will receive Web Services.

- **Web Service Upload:** Specify the root URL for the Axis2 application, typically `http://<Host>:<HTTP port>/axis2/axis2-admin/`, as well as the user name and password of the Axis2 administrator.
 - **FTP Upload:** Specify the FTP URL of the deployment directory as well as the user name and password of a user with rights to write in this directory.
3. Click **OK**.
A window will open to allow you to create a physical schema.
 4. Go to the **Context** tab, and define one logical schema for each context in which you will deploy the Data Services.
 5. Click **OK**.

Note : You only need to define one physical schema per Web Services container.

For further information about setting up the topology, refer to [Creating the Topology](#).

Setting up the model

To configure Data Services for a model, make sure that the datastores which you wish to deploy are already correctly defined (such as by reverse-engineering) and accessible. For further information, see [Creating and Reverse-engineering a Model](#).

Make sure that you 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 further information on importing KMs, see [Importing a KM](#).

You may also wish to consult the section [Overview of Generated Services](#) for further information about how the properties below will affect the generated Data Services.

To set up a model to use Data Services:

1. Open the model and go to the **Services** tab.
2. On the **Application Server** tab, select the Web Services container that you set up earlier.
3. Set the **Namespace**, which will be used in the generated WSDL.
4. Specify the **Package name**, used to name the generated Java package that contains your Web Service. Generally, this is of the form `com.<company name>.<project name>`
5. In the **Datasource name** field, copy/paste the name of the datasource that you defined for the server when setting up the datasources. This name should be prefixed by `java:/comp/env/`.
6. Define the **Data Service Name**. Note that this name is only used for the Data Service operating at the model level. You can also define a Data Service name for each datastore - see below.
7. Select a service knowledge module (SKM) from the list, and set its options. Refer to the description of the SKM for more information. Note that only those SKMs imported into projects appear here.
8. Go to the **Deployed Datastores** tab.
9. Select every datastore that you wish to expose as a Web Service. For each one, specify a **Data Service Name** and the name of the **Published Entity**.
10. Click OK to save your changes.

Although not required at first, you can also fine-tune the configuration of your generated Data Services at the datastore and column level.

To configure options for Data Services at the datastore level:

1. Open the datastore and select the **Services** tab.
2. Check Deploy as Data Service if you want the datastore to be deployed.
3. Enter the **Data Service Name** and the name of the **Published Entity** for the datastore.
4. Click **OK** to save your changes.

To configure Data Service options at the column level:

You can specify the operations that will be permitted for each column. One important use of this is to lock a column against being written to via Data Services.

1. Open the column and select the **Services** tab.
2. Check the operations that you wish to allow: **INSERT, UPDATE, SELECT**.
3. Click **OK** to save your changes.

Generating and Deploying Data Services

Generating Data Services for a model also has the effect of generating the Data Services for the individual datastores.

To generate Data Services for a datastore or model:

1. Select the model or datastore for which you wish to generate the Data Services.
2. Right-click, and select **Generate Service**.
The generation configuration window opens.
3. Specify the path to store the generated Data Service 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.
4. Specify the **Context**. Note that the choice of context has three effects:
 - Determining the JDBC/Java datatype bindings at generation time.
 - Determining which physical schemas are used to provide data.
 - Determining which physical Web Services container is deployed to.
5. Choose 1 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.

It is also possible to use the **Generate and deploy...** button on the **Services** tab in either the **Datastore** or **Model** window.

Note: Generating Data Services for a model generates one Data Service for the model, and an additional Data Service for each of the datastores which are enabled for publishing as such.

Phase	Description
Generation	<ul style="list-style-type: none"> • Deletes everything in the directory. • Generates .java files using SKM.
Compilation	<ul style="list-style-type: none"> • Extracts Web Service framework. • Compiles .java files into .class files.
Deployment	<ul style="list-style-type: none"> • Generates a Java package (.aar) from compiled files. • Copies package to deployment target, using chosen deployment method.

Overview of Generated Services

Model-level services

Data Services are only generated at the model-level when the model is enabled for consistent set CDC. The following services are generated:

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

See Change Data Capture 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 currently supplied by 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" will be used.

- 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 a 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 `SnpsInvokeWebService` Oracle Data Integrator tool. See also Invoking a Web Service for more information.

Handling Projects

What is a Project?

A **project** is a group of objects developed using Oracle Data Integrator.

- Managing a Project

Oracle Data Integrator Project Components

The following components are stored below the level of the project in the tree view:

Folder

Certain objects in a project are organized into **folders** and **sub-folders**.

Packages

The **package** is the largest unit of execution in Oracle Data Integrator. A package is made up of a sequence of **steps** organized into an execution diagram.

- More information about Packages
- Create a Package

Interface

An interface consists of a set of rules that define the loading of a Datastore or a temporary target structure from one or more source Datastores.

- More information about Interfaces
- Create an Interface

Procedure

A Specific Procedure is a reusable component that groups operations that do not fit in the Interface framework - loading a target datastore from one or more sources.

Examples of procedures:

- wait and unzip a file
- send a batch of files via FTP
- receive emails
- purge a database

A procedure can launch commands on logical schemas defined in the topology, but may also use OS commands or Oracle Data Integrator tools.

Variable

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

The value:

- has a default defined at creation time
- can be passed as a parameter when running a scenario using the variable,
- may change with the refresh, set and increment variable steps ,
- may be evaluated to create conditions in the packages,
- may be used in the interfaces, procedures, steps, ...

A variable can be defined outside a project (global scope), in order to be used in all projects.

Sequence

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

The sequences are usable like variable in interfaces, procedures, steps, ...

A sequence can also be defined outside a project (global scope), in order to be used in all projects.

User Functions

User functions enable to define customized functions or "functions aliases", for which you will define technology-dependant implementations. They are usable in the interfaces and procedures.

Knowledge Modules

Oracle Data Integrator uses knowledge modules to define methods related to a given technology. These modules enable processes generation for the technology, dedicated to a precise function.

Note: Default knowledge modules are provided with Oracle Data Integrator, and must be imported in the project before use.

Marker

Elements of a project may be flagged in order to reflect the methodology or organization of the developments.

Flags are defined using the markers. These markers are organized into groups, and can be applied to most objects in a project.

Scenario

When a package, interface, procedure or variable component is finished, it is compiled in a **scenario**. A scenario is the execution unit for production, that can be scheduled.

Managing an Oracle Data Integrator Project

Managing an Oracle Data Integrator project generally involves the following steps:

1. Creating and reverse-engineering models.
2. Creating a project.
3. Using markers (optional).
4. Creating and organizing folders.
5. Importing KMs.
6. Creating and modifying reusable objects:
 - Variables.
 - Sequences.
 - Interfaces.
 - Procedures.
 - User functions.

7. Unit testing interfaces and procedures (back to step 6).
8. Building packages from elements created in step 6.
9. Integration testing the packages.
10. Generating scenarios.
11. Scheduling scenarios.
12. Managing the scenarios in production.
13. Maintenance, bug fixing and further modifications (back to step 6).

Creating a New Project

To create a project:

1. Click the **Insert Project** button in the **Projects** view.
2. Enter the **Name** of the project.
3. Keep or change the automatically-generated **Code**.
4. Click **OK**.

The empty project appears with an empty folder for interfaces, procedures and packages.

Creating a New Folder

Interfaces, procedures and packages are organized into **folders** and **sub-folders**.

To create a folder:

1. Click the **project** or the **folder** where you want to create the folder.
2. Right click and select **Insert Folder**.
3. Enter the **Name** of the folder.
4. Click **OK**.

The empty folder appears.

Creating Packages

What is a Package ?

The **package** is the biggest execution unit in Oracle Data Integrator. A package is made of a sequence of **steps** organized in an execution diagram.

- Create a Package.

Steps

There are different type of steps. They can be grouped in families of steps:

- **Flow (Interface):** Executes an interface.
- **Procedure:** Executes a procedure.
- **Variable:** Declares, sets, refreshes or evaluates the value of a variable.
- **Oracle Data Integrator Tools:** These tools, available in the Toolbox, enable to access all Oracle Data Integrator API commands, or perform operating system calls.

- **Models, Sub-models and Datastores:** Performs journalizing, static control or reverse-engineering operations on these objects.

For example, the "Populate Sales Datamart" Package could be made up of the following jobs:

1. Procedure "System Backup"
2. Interface "Customer Group"
3. Interface "Customer"
4. Interface "Product"
5. Refresh variable "Last Invoice ID"
6. Interface "Invoice Header"
7. Interface "Invoice Lines"

Creating a Package

Package creation can be performed thanks to the following processes:

1. Create a New Package.
2. Handle Steps (insert, duplicate, delete, ...).
3. Define Step Sequences.
4. Run the Package.

1. Creating a New Package

To create a new package:

1. Click the **packages** node in the **folder** where you want to create the package.
2. Right click and select **Insert Package**.
3. Type in the **Name** of the package.
4. Click **OK**.


An new package is created.

2. Working with Steps

To insert a step:


Adding a step depends on the nature of the steps being inserted. For more details, see Adding a Step.

To delete a step:

1. In the **package's diagram** tab, click .
2. Right click the icon of the step and select **Delete Step**, then click **OK**.


The step disappears from the diagram.

To duplicate a step:

1. In the **package's diagram** tab, click .
2. Right click the icon of the step and select **Duplicate Step**.


A new step appears in the diagram.

To run a step:

1. In the **package's diagram** tab, click .
2. Right click the icon of the step and select **Execute Step**.
3. Set the execution options, then click **OK**.


To edit a step's linked object:

The step's linked object is the interface, procedure,... from which the step is created.

1. In the **package's diagram** tab, click .
2. Double-click on the step's icon. The linked object's editing window appears.

To arrange the steps in the diagram:

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

1. In the **package's diagram** tab, click .
2. Select one of more **steps**:
 - Keep the CTRL key pressed and click on each step
 - Drag the cursor on the diagram with the left mouse button pressed. A selection box enclosing the steps will appear.
3. To arrange the selected steps, you may either:
 - Drag them to arrange their position
 - Right-click, then select a vertical or horizontal alignment option in the contextual menu.

It is also possible to use the **reorganize** button from the toolbar to automatically reorganize the steps.

3. Defining the Sequence of Steps

Once the steps are created, you must reorder 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, 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	<ul style="list-style-type: none"> • Error in an interface command • Maximum number or percentage of errors allowed reached.
Procedure	<ul style="list-style-type: none"> • Error in a procedure command

Refresh Variable	<ul style="list-style-type: none">• Error in the refresh order
Set Variable	<ul style="list-style-type: none">• Error when setting the variable (invalid value)
Evaluate Variable	<ul style="list-style-type: none">• The condition defined in the step is not valid
Declare Variable	<ul style="list-style-type: none">• N/A
Execute OS Command	<ul style="list-style-type: none">• OS Command return code is different from zero
Oracle Data Integrator Tool	<ul style="list-style-type: none">• Oracle Data Integrator Command return code is different from zero
Journalize Datastore, Model or Sub-Model	<ul style="list-style-type: none">• Error in a journalizing command
Check Datastore, Model or Sub-Model	<ul style="list-style-type: none">• Error in a the check process
Reverse Model	<ul style="list-style-type: none">• Error in the reverse process


Define the sequences

To define the first step of the package:

1. In the **diagram** tab, click .
2. Click the icon of the step you want to define as the **first step**.
3. Right-click and select **First Step** box.


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

To define the next step upon failure:


1. In the **diagram** tab, click .
2. Click the icon of the step that may **fail**.
3. Keep the mouse button pressed and move the cursor to the icon of the step that must follow in case of a **failure**, then Release the mouse button.

A red arrow appears between the two steps (In the case of an Evaluate variable step, it is green).


To define the end of the package upon failure:

1. In the **diagram** tab, click .
2. Click the icon of the step that may **fail**.
3. In the properties pane, on the **Advanced** tab, select **End** in the **Processing after failure** zone.

To define retries before going in error


1. In the **diagram** tab, click .
2. Click the icon of the step that may **fail**.
3. In the properties pane, go to the **Advanced** tab.
4. Fill in the **Number of attempts** and the **Time between attempts**.

To define the next step upon success:

1. In the **diagram** tab, click .
2. Click the icon of the step that may be a **success**.
3. Keep the mouse button pressed and move the cursor to the icon of the step that must follow in case of a **success**, then Release the mouse button.

A green arrow appears between the two steps.

To define the end of the package upon success:

1. In the **diagram** tab, click .
2. Click the icon of the step that may be a **success**.
3. In the properties pane, on the **Advanced** tab, select **End** in the **Processing after success** zone.

4. Running the Package

To run a package:

1. In the **Package's Execution** tab, click **Execute**.
2. If you have not saved your package yet, click **Yes** to apply the changes.
3. Go to the Execution window and click **OK**.

Adding a Step

The procedure to add a step in a package depends on the step type.

Flow (Interface)

To insert a Flow step into a package:

1. Open the **package** and go to the **Diagram** tab.
2. Select the **interface** to add to the package from the tree view.
3. Drag and drop the interface into the diagram. The **Flow Step** appears.
4. Click the step's icon in the diagram. The properties panel opens.
5. In the **General** tab, fill in the **Name** field.
6. In the **Advanced** tab, define the processing after success/failure options (See Diagram).
7. Click **Apply** to save.

Procedure

To insert a Procedure step in a package:

1. Open the **package** and go to the **Diagram** tab.
2. Select the **procedure** to add to the package from the tree view.
3. Drag and drop the procedure into the diagram. The **procedure step** appears.
4. Click the step's icon in the diagram. The properties panel opens.
5. In the **General** tab, fill in the **Name** field.
6. In the **Options** tab, set the procedure options.

7. In the **Advanced** tab, define the processing after success/failure options (See Diagram).
8. Click **Apply** to save.

Variable

To insert a Declare Variable step in a package:

1. Open the **package** and go to the **Diagram** tab.
2. Select the **variable** to add to the package from the tree view.
3. Drag and drop the variable into the diagram. The **Variable Step** will appear.
4. Click on the step icon in the diagram. The properties panel opens.
5. In the **General** tab, fill in the **Name** field. Select **Declare Variable** in the Step **Type**.
6. In the **Advanced** tab, define the processing after success/failure options (See Diagram).
7. Click **Apply** to save.

To insert a Refresh Variable step in a package:

1. Open the **package** and go to the **Diagram** tab.
2. Select the **variable** to add to the package from the tree view.
3. Drag and drop the variable into the diagram. The **Variable Step** will appear.
4. Click on the step icon in the diagram. The properties panel opens.
5. In the **General** tab, fill in the **Name** field. Select **Refresh Variable** in the Step **Type**.
6. In the **Advanced** tab, define the processing after success/failure options (See Diagram).
7. Click **Apply** to save.

To insert a Set Variable step in a package:

1. Open the **package** and go to the **Diagram** tab.
2. Select the **variable** to add to the package from the tree view.
3. Drag and drop the variable into the diagram. The **Variable Step** will appear.
4. Click on the step icon in the diagram. The properties panel opens.
5. In the **General** tab, fill in the **Name** field. Select **Set Variable** in the Step **Type**.
6. Select **Assign** or **Increment** depending on the operation you want to perform.
7. Type in the **value** to assign or the increment amount.
8. In the **Advanced** tab, define the processing after success/failure options (See Diagram).
9. Click **Apply** to save.



To insert an Evaluate Variable step in a package:

1. Open the **package** and go to the **Diagram** tab.
2. Select the **variable** to add to the package from the tree view.
3. Drag and drop the variable into the diagram. The **Variable Step** will appear.
4. Click on the step icon in the diagram. The properties panel opens.
5. In the **General** tab, fill in the **Name** field. Select **Evaluate Variable** in the Step **Type**.
6. Select the evaluation **Operator** and the **value** to compare. This value may be another variable.

7. In the **Advanced** tab, define the processing after success/failure options (See Diagram). Success of an evaluate variable step indicates that the evaluation gives a positive result.
8. Click **Apply** to save.

Oracle Data Integrator Tools




To insert an Oracle Data Integrator Tool step:

1. Open the **package** and go to the **Diagram** tab.
2. In the **Toolbox**, select the tool you want to use.
3. Click on the diagram. An **step** corresponding to your tool appears.
4. Click .
5. Double-click the step icon in the diagram. The properties panel opens.
6. In the **General** tab, fill in the **Name** of the step.
7. Enter or select the appropriate values for the parameters of the tool.
8. In the **Command** tab, the generated Oracle Data Integrator command appears. You can use the expression editor by clicking .
9. In the **Advanced** tab, define the processing after success/failure options (See Diagram).
10. Click **Apply** to save.

Note: For invoking a web service, a customized GUI is provided. See Invoking Web Services for more informations.



Tip: To create SnpsStartScen (start an Oracle Data Integrator scenario) step, you can directly drag and drop the scenario from the project tree view to the diagram. You can also use the **Additional Variables** to enter the scenarios variables.


To insert an OS command step in a package:

1. Open the **package** and go to the **Diagram** tab.
2. In the **Toolbox**, select **OS Command**  (in the **Utilities** group).
3. Click on the diagram. An **OS Command step** will appear.
4. Click .
5. Double-click the step icon in the diagram. The properties panel opens.
6. In the **General** tab, fill in the **Name** of the step.
7. Type in the **Text of the command**. You may call the expression editor by clicking .
8. In the **Advanced** tab, define the processing after success/failure options (See Diagram).
9. Click **Apply** to save.

Warning: Using an OS command may make your package platform-dependant.

To insert an Oracle Data Integrator Command step in a package:

1. Open the **package** and go to the **Diagram** tab.
2. In the **Toolbox**, select Oracle Data Integrator **Command**  (in the **Utilities** group).
3. Click the diagram. An Oracle Data Integrator **Command step** appears.
4. Click .

5. Double-click on the step icon in the diagram. The properties panel opens.
6. In the **General** tab, fill in the **Name** of the step.
7. Go to the **Command** tab, and type in the Oracle Data Integrator command. You can use the expression editor by clicking  to edit the code. If you go back to the **General** tab, the Oracle Data Integrator command is recognized and the properties list appears. You can use this tab to enter parameters values.
8. In the **Advanced** tab, define the processing after success/failure options (See Diagram).
9. Click **Apply** to save.

See the Tools Reference for the details of each Oracle Data Integrator Tool.

To insert an Oracle Data Integrator Open Tool step:

Open Tools are listed under the **Plugins** group, and are used the same way as other tools. See Using Open Tools in the Oracle Data Integrator Tools Reference for more information on installing and using Open Tools.

Model and Datastore

To insert a Check step in a package:

1. Open the **package** and go to the **Diagram** tab.
2. Select the **model** or **datastore** to add to the package from the tree view.
3. Drag and drop the model or datastore into the diagram. A **model** or **datastore Step** will appear.
4. Double-Click on the step icon in the diagram. The properties panel opens.
5. In the **General** tab, fill in the **Name** field. Select **Check Datastore/Model** in the Step **Type**.
6. If you want to remove errors from the checked tables, check **Delete ... from checked Tables**.
7. In the **Options** tab, set the specific KM options.
8. In the **Advanced** tab, define the processing after success/failure options (See Diagram).
9. Click **Apply** to save.

Note: It is necessary to define the CKM in the model to perform the check.

To insert a Journalizing step in a package:

1. Open the **package** and go to the **Diagram** tab.
2. Select the **model** or **datastore** to add to the package from the tree view.
3. Drag and drop the model or datastore into the diagram. A **model** or **datastore Step** will appear.
4. Click on the step icon in the diagram. The properties panel opens.
5. In the **General** tab, fill in the **Name** field. Select **Journalize Datastore/Model** in the Step **Type**.
6. Set the parameters corresponding to the journalizing operations you want to perform. See Changed Data Capture for more information.
7. In the **Options** tab, set the specific KM options.
8. In the **Advanced** tab, define the processing after success/failure options (See Diagram).
9. Click **Apply** to save.

Note: It is necessary to define the JKM in the model to perform the journalizing.

To insert a Reverse-engineering step in a package:

1. Open the **package** and go to the **Diagram** tab.
2. Select the **model** to add to the package from the tree view.
3. Drag and drop the model into the diagram. A **model Step** will appear.
4. Click on the step icon in the diagram. The properties panel opens.
5. In the **General** tab, fill in the **Name** field. Select **Reverse Model** in the **Step Type**.
6. In the **Options** tab, set the specific KM options.
7. In the **Advanced** tab, define the processing after success/failure options (See Diagram) .
8. Click **Apply** to save.

Note: It is necessary to define the reverse-engineering options in the model.

Creating Scenarios

Generating a Scenario


When a component is finished and tested, you can operate it by generating the scenario corresponding its actual state. This operation takes place in the **Designer** module.

It is possible to generate scenarios for packages, procedures, interfaces or variables. These three later types are single-step scenarios.

Scenario variables are variables used in the scenario that should be set when starting the scenario to parameterize its behavior.

Note: Scenarios generated for variables contain a single step performing a refresh operation for the variable.

To generate a scenario:

1. Double-click the **Package** , **Interface**, **Procedure** or **Variable** in the Projects view. The edition window opens.
2. Select the **Scenarios** tab.
3. Click the  **Generate Scenario** button. A **New Scenario** window appears.
4. Enter the **Name** of the scenario. Avoid special characters and blanks, as this name could later be used in an operating system command.
5. Enter the **version** of your scenario (same remark as for the scenario name concerning characters).
6. Click **OK**.
7. If you use variables in the scenario, you can define in the **Scenario Variables** window the variables that will be considered as parameters for the scenario. Select **Use All** if you want all variables to be parameters, or **Selective Use**, then check the parameter variables.

The scenario appears under the **scenarios** node of the source object in the Projects view.

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.

To regenerate a scenario:

1. Select the **Scenario** in the tree

2. Right click and select **Regenerate**.
3. Click **OK**

Warning: 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, interfaces, procedures and variables** grouped under a **project** or **folder** is finished and tested, you can prepare them for operation by generating the scenarios. This operation takes place in the **Designer** module.

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** window, select the scenario **Generation Mode**:
 - **Replace:** causes the last scenario generated to be replaced by the new one generated.
 - **Creation:** creates a new scenario named the same as the last scenario generated, with the version number automatically increased. If no scenario is created yet, the new scenario will have the source component's name.

Note: If the **version** of the last scenario is an integer, it will be automatically incremented by 1 when selecting a **Creation Generation Mode**. If not, the version will be automatically set to the current date.

4. Select the types of objects for which you want to generate scenarios.
5. 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** window the variables that will be considered as parameters for the scenario. Select **Use All** if you want all variables to be parameters, or **Selective Use**, then 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 unique scenarios or groups of scenarios.

To export one scenario, see Export an Object.

To export an a group of scenarios:

1. Select the **Project** or **Folder** containing the group of scenarios.
2. Right click and select **Export all scenarios**.
3. Fill in the Export parameters.
If the **Export Directory** is not specified, then the export file is created in the **Default Export Directory**.
4. Select the type of objects whose scenarios must be exported.
5. Click **OK**

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

Encrypting a Scenario

Encrypting a scenario allows you to protect valuable code. With an encrypted scenario the commands that appear in the log are executed while being unreadable for the user.

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 scenario 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.

To Encrypt a Scenario:

1. Right click the scenario you want to encrypt.
2. Select **Encrypt**.
3. In the **Encryption Options** window, either:
 - select the **Encrypt with a personal key** option , then select an existing Encryption Key file,
 - select the **Encrypt with a personal key** option, then type in (or paste) the string corresponding to your **personal key**,
 - or let Oracle Data Integrator generate a key using the **Get a new encryption key** option.
4. The **Encryption Key** window appears when the encryption is finished. From this window, you can save the key.

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** window, 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.

Creating Interfaces

What is an Interface?

An interface consists of a set of rules that define the loading of a datastore or a temporary target structure from one or more source datastores.

- Create an Interface

Components of an Interface

Target Datastore

The **target datastore** is the element that will be loaded by the interface. This datastore may be permanent (defined in a model) or temporary (created by the interface in the staging area).

Source Datastores

The **source datastores** contain data used to load the target datastore. Two types of datastores can be used as a source of an interface: The datastores from the models, and temporary datastores target of an interface.

The source datastores of an interface can be filtered during the loading process, and must be put in relation through joins. **Joins** and **filters** can be recovered from the models definitions, and can also be defined for the interface.

Mapping

The **mapping** defines the transformation rules on the sources enabling to generate data to load the target.

Flow

The **flow** is the set of loading and integration strategies for mapped data, based on **knowledge modules**.

Control Strategy

The **flow control strategy** lets you define the method used to check the flow before the insertion in the target. The control strategy is defined by a **Check Knowledge Module (CKM)**.

The interfaces use the following components:

- **Datastores** defined in the models as sources and target or the loading process.
- **Knowledge Modules** to generate the appropriate processes.
- **Variables** and **Sequence** to store values or counters in the expressions.
- **Users functions** to ease the transformation rule coding.

Creating an Interface

The following operations must be performed to create an interface:

1. Create a New Interface.
2. Define the Target Datastore.
3. Define the source datastores, filters and joins on these sources.
4. Define the mapping between source and target data.
5. Define the interface flow.
6. Define the flow control.
7. Execute the interface for testing.

1. Create a New Interface

To create a new interface:

1. In the Designer tree, select the **Interfaces** node in the **folder** under the **project** where you want to create the interface. Right click and select **insert interface**. A blank interface appears in the right panel of the Designer window.
2. Fill in the interface **Name**.
3. Check the **Staging Area different from Target** box if necessary and select a logical schema that will be the staging area.

Note: The staging area defaults to the target. It may be necessary to put it on a different logical schema if the target does not have the required transformation capabilities for the interface. This is the case for File, JMS, ... logical schemas. It is also necessary to define a staging area for interfaces with a temporary target datastore.

4. Go to the **Diagram tab** to proceed.

2. Define the Target Datastore

The target datastore is the element that will be loaded by the interface. This datastore may be permanent (defined in a model) or temporary (created by the interface in the staging area).

2.1 Permanent Target Datastore

To insert the permanent target datastore in an interface:

1. Expand the tree to show the **datastore** to be inserted as the target.
2. Select the **datastore**, then drag and drop it as the target of the interface (in the right part of the diagram tab). The target datastore appears in the diagram.

To display the data of the permanent target datastore of an interface:

1. Right click the title of the target datastore in the **diagram** tab of the **interface** window.
2. Select **data**.

A window containing the data of the target datastore appears. Data in a temporary target datastore cannot be displayed since this datastore is created by the interface.

2.2 Temporary Target Datastore

To add a temporary target datastore:

1. In the **Diagram** tab, double-click the title of the target datastore: **Untitled**.
2. In the property pane, type in a **name** for this datastore. Specify if you wish to create the temporary datastore in the **Work Schema** or **Data Schema** of the staging area.

Note: You can define a temporary target datastore only if the staging area is different from the target.

Note: The temporary target datastore will be created only if you activate the KM option `CREATE_TARGET_TABLE` when defining the flow.

The temporary target datastore is created without columns. They must be added to define its structure.

To add a column to a temporary target datastore:

1. Right click the title of the target datastore.

2. Select **Add a column**.
3. A new empty column appears. Double click this new column in the target and fill in the **target column** fields (See Mapping).

To delete a column from a temporary target datastore:

1. Right click the **column** to be deleted from the target datastore.
2. Select **Delete**.

To add all of the columns from a source datastore to a temporary target datastore:

1. Add the source **datastore**.
2. Select the title of the **entity** representing the source datastore.
3. Right click and select **Add to the target**.

2.3 Define the update key

If you want to use update or flow control features in your interface, it may be necessary to define an update key on the target datastore. This key identifies each record to update or check before insertion into the target. This key can be a unique key defined on the target datastore in its model, or a group of columns specified as a key for the interface.

To define the update key from a unique key:

1. In the **Diagram** tab, double-click the title of the target datastore.
2. In the property panel, select the **Update key** from the list.

Note: Only keys defined in the model appear in this list.

To define the update key from the columns:

1. In the **Diagram** tab, double-click one of the target datastore's columns that is part of the update key.
2. In the properties panel, check the **key** box. A key symbol appears in front of the column in the mapping.
3. Repeat the operation for each column that making up the update key.

3. Define the Source Datastores

The source datastores contain data used to load the target datastore. Two types of datastores can be used as an interface source: The datastores from the models and temporary datastores target of an interface.

The source datastores of an interface can be filtered during the loading process and must be put in relation through joins. Joins and filters can be recovered from the model definitions and can also be defined for the interface.

3.1 Define the Source Datastores

To add a permanent-type source datastore to an interface:

1. Select the **datastore** then drag and drop it onto the composition panel (left part of the diagram tab). The datastore appears in the diagram model.

Warning: If there are any filters on the datastore, or references between this datastore and existing datastores in the diagram, they appear along with the datastore. These references and filters are copied as joins and filters in the interface. They are not links to existing references and filters from the model. Therefore, modifying a reference or a filter in a model does not affect the join or filter in the interface, and vice versa.

Using journalized datastores: If the datastore is journalized, it is possible to use only the journalized data in the interface flow. Check the **Journalized Data only** box in the source datastore properties. A Journalizing filter is automatically created in the diagram.

To add a temporary-type source datastore to an interface:

1. Select the **interface**, then drag and drop it onto the composition panel (left part of the diagram tab). The target datastore of this interface appears in the diagram model.

To delete a source datastore from an interface:

1. Right click the title of the **entity** (table) representing the source datastore.
2. Select **Delete**. A confirmation window appears. Select **Yes**.

The source datastore disappears, along with the associated filters and joins.




To display the data or the number for rows of a source datastore of an interface:

1. Right click the title of the **entity** (table) that represents the source datastore.
2. Select **Number of rows** or **data**.

A window containing the number or rows or the data of the source datastore appears.

3.2 Define Filters on the sources

To define a filter on a source datastore:

1. Select a column in the entity representing the first table to join, and, holding it selected, drag and drop it onto the composition panel. A **filter** appears in the model.
2. Modify the **implementation** to create the required filter. You may call the expression editor by clicking .
3. Select the execution location: **source** or **staging area**.
4. Click  to validate the expression, then click  to save.
5. Check the **Active Filter** box to define whether or not you want this filter to be used for the execution of the interface.

To delete a filter on a source datastore of an interface:

1. Select the **filter** to remove.
2. Right click and select **Delete**.




To display the data or the number of rows resulting from a filter:

1. Right click the **filter**.
2. Select **data** or **number of rows**.

A window containing the data or the number of rows after the filter appears.

3.3 Define Joins between sources

To create a join between the source datastores of an interface:

1. Select a column in the entity representing the first table to join, and, holding it selected, drag and drop it on a column in the entity of the second table to join.
2. A **relation** linking the two tables appears in the model. In the **implementation** field, an equality between the two columns concerned also appears.
3. Modify the **implementation** to create the required relation. The columns of all the tables in the model can be drag-and-dropped into the join text to create multi-table joins. You may call the expression editor by clicking .
4. Click  to validate the expression, then click  to save.
5. Select the execution location: **source** or **staging area**.
6. Select the type of join (right/left, inner/outer, ISO), and the order number.
7. Check the **Active Clause** box to define whether or not you want this join be used for the execution of the interface.

To delete a join between source datastores of an interface:

1. Right click the **relation** that represents the join to remove.
2. Select **Delete**.

To display the data or the number of rows resulting from a join:

1. Right click the **relation** representing the join.
2. Select **data** or **number of rows**.

A window containing the data or the number of rows resulting from the join appears.

4. Define the mappings

The mapping defines the transformation rules on the sources, allowing the generation of data to load the target.




The mapping is filled automatically at each new source or target insertion using column names matching. The user-defined mapping always takes precedence over name matching.

To regenerate the automatic mapping by name matching:

1. Right click the **target datastore**.
2. Select **Redo Automatic Mapping**.

The target datastore columns are automatically mapped on the source datastores' columns with the closest name.

To define the mapping of a target column:

1. Double click the column of the target datastore.
2. Modify the **implementation** in the composition panel to perform the required transformation. The columns of all the tables in the model can be drag-and-dropped into the text. You may call the expression editor by clicking .
3. Click  to validate the expression, then click  to save.
4. Select the execution location: **source**, **target** or **staging area**. See Mappings in the Reference Manual for restrictions on the possible execution locations.

5. Check the boxes from the **update** zone if you want the mapping to be executed in **Insert** or **Update** operations or in KM-specific operations (**UD1** to **UD5**)
6. Check the **Active Mapping** box to define whether or not you want this mapping to be used for the execution of the interface.

5. Define the interface flow

In the **flow** tab, you define the loading and integration strategies for mapped data. Oracle Data Integrator automatically computes a group of source-sets depending on the mapping rules, joins and filters of the interface's **diagram**. It proposes default KMs for the data flow. The flow tab enables you to view the source-sets and the KMs used to load and integrate data. If you want to define the strategies for the flow, you must change the KMs in use.

To change the KMs in use:

1. In the interface's flow tab, select a **source-set** by clicking its title.
2. In the properties panel, select a **LKM** or **IKM** depending on the strategy to set. For more information on the KM choices, refer to the KM documentation.
3. Set the **KM Options**.
4. Repeat the operation for all source-sets, then click on **Apply**.

Note: Only KMs that are relevant for the technologies in use and that are imported into the project appear in the list of KMs.

Note: Some IKM require the use of an update key.

Note: When changing from one KM to another, any options of the same type and the same name are preserved.

6. Set up flow control

The flow control strategy enables you to define the method used to check the flow before insertion into the target. The control strategy is defined by a CKM.

To define the CKM used in an interface:

1. In the **Control** tab of the interface, select a **CKM**. For more information on the KM Choice, refer to the KM documentation.
2. Set the **KM Options**.
3. Select the **constraints** to be checked.
4. Fill in the **Maximum number of errors allowed** and check the **%** box if you want the interface to fail when a number (or percentage) of errors is reached.
5. Click **Apply**.

Note: Only CKM that are relevant for the technologies in use and that are imported into the project appear in the CKM lists.

Note: Some CKM require the use of an update key.

Note: Only the constraints declared on the target datastore in its model appear in the constraints list.

Warning: Flow control will be executed only if the `FLOW_CONTROL` option is activated in the IKM (**Flow** tab).

7. Execute the interface

Once the interface is created, it is possible to test it.

To run an interface:

1. In the interface's **Execution** tab, click **Execute**.
2. If you have not saved your interface yet, then click **Yes** when asked to save it.
3. Define the **Execution options**, then click **OK**.

Creating Procedures

Creating a Procedure

A Procedure is a reusable component that allows you to group actions that do not fit in the Interface framework. (That is load a target datastore from one or more sources).

A **Procedure** is a sequence of **commands** launched on logical schemas. It has a group of associated options. These **options** parameterize whether or not a command should be executed as well as the code of the commands.

Creating a procedure is made through the following steps:

1. Create a new procedure.
2. Define the procedure's options.
3. Create and Manage the procedure's commands.
4. Execute the procedure.

1. Creating a new procedure

To create a new procedure:

1. In the Designer tree view, click the **procedures** node in the **folder** where you want to create the **procedure**.
2. Right click and select **Insert Procedure**.
3. Fill in the procedure's **name**.
4. Check the **Multi-Connections** box if your procedure will 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 connection (the source connection) in a command sent to another connection (the target connection). This data will pass through the execution agent. 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.

5. Select the **target technology** and if the **Multi-Connections** box is checked also select the **source technology**. If you wish to make the procedure generic, leave these fields empty.
6. Click **Apply**.

A new procedure is created, and appears in the list of procedures in the tree view.

2. Defining the procedure's options

To create procedure's options:



1. In the Designer tree view, click on your **procedures's** node
2. Right click and select **Insert Option**. An option window will appear.
3. Fill in the option's **Name, type** and **default value**.
4. Click **OK**.
5. Repeat these operations for each **option** that is required for the procedure.

Note: Only **checkbox** options (yes | no) enable you to parameterize whether or not a command should be executed. The values of the other options (**value** and **text**) can only be recovered in the code of the procedure's **commands**, using the `getOption()` substitution method.

Options appear in the tree view under the **procedure** node.

3. Creating and managing the procedure's commands

To create a procedures's command line:

1. Go to the **Details** tab of the **Procedure**.
2. Click . A **command** window appears.
3. Fill in the following fields:
 - **Name** of the command.
 - **Ignore Errors** must be checked if you do not want the procedure to stop if this command returns an error.
 - **Log Counter** Shows which counter (Insert, Update, Delete or Errors) will record the number of rows processed by this command.
 - **Log level:** Level of importance of the command, for consulting the execution log according to this level.
4. For the **Command on Target**, fill in the following fields:
 - **Technology:** Technology used for this command. If it is not set, the technology specified in the procedure window is used.
 - **Schema:** Logical schema for execution of the command.
 - **Context:** Forced Context for the execution. If it is left undefined the execution context will be used.
 - **Transaction:** Transaction where the command will be executed.
 - **Commit:** Indicates the commit mode of the command in the transaction.
 - **Transaction Isolation:** The transaction isolation level for the command.
 - **Command:** Text of the command to execute. You may call the expression editor by clicking .

Note: You can use commands in the appropriate language for the technology. To use OS commands, you must use the **Operating System Technology**, and to use Oracle Data Integrator Tools, you must use the **Oracle Data Integrator API** technology.

Note: It is advised to use the substitution methods to make the code generic and dependent on the elements of the topology.

Warning: The **transaction**, **commit** and **transaction isolation** options work only for technologies supporting transactions.


5. Repeat step 4 for the **command on the source**.
6. Go to the **Options** tab.
7. Check the **always execute** box if you want this command to be executed all the time regardless of the option values. Otherwise, check the **options** that will cause this command to be executed.
8. Click **OK**

To duplicate a command:

1. Go to the **Details** tab of the **Procedure**.
2. Click the command to duplicate.
3. Right-click then select **Duplicate**.
A **command** window appears. It is a copy of the selected command.
4. Change it, then click **OK**.

The new command appears.



To delete a command line:

1. Go to the **Details** tab of the **Procedure**.
2. Click the **command line** to delete.
3. Click .

The command line will disappear from the list..

To order the command lines:

The **command** lines are executed in the order displayed in the **details** tab of the **procedure** window. It may be necessary to reorder them..

1. Go to the **Details** tab of the **Procedure**.
2. Click on the **command line** you wish to move.
3. Click  and  to move the command line to the appropriate position.

4. Executing the procedure

When the procedure is ready, it is possible to test it.

To run a procedure:

1. In the **Execution** tab of the **procedure**, click **Execute**.
2. If you have not saved your procedure yet, then click **Yes** when asked to save it.
3. Set the **Execution options**, then click **OK**.

Encrypting a KM or Procedure

Encrypting a Knowledge Module (KM) or a 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 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.

To Encrypt a KM or a Procedure:

1. Right click the KM or procedure you wish to encrypt.
2. Select **Encrypt**.
3. In the **Encryption Options** window, either:
 - select the **Encrypt with a personal key** option , then select an existing Encryption Key file,
 - select the **Encrypt with a personal key** option, then type in (or paste) the string corresponding to your **personal key**,
 - or let Oracle Data Integrator generate a key using the **Get a new encryption key** option.
4. The **Encryption Key** window appears when the encryption is finished. From this window, you can save the key.

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.

See also:

Decrypting a KM or a Procedure

Decrypting a KM or Procedure

To decrypt a KM or a procedure:

1. Right click the KM or procedure that you wish to decrypt.
2. Select **Decrypt**.
3. In the **KM Decryption** window, either
 - select an existing encryption key file;
 - or type in (or paste) the string corresponding to your **personal key**.

A message appears when the decryption has finished.

See also:

Encrypting a KM or a Procedure

Creating Variables & Sequences

Creating and Using Variables



A variable is an object that stores a single value. This value can be a string, a number or a date. The value is stored in Oracle Data Integrator, and can be updated at run-time.

The value of a variable can be updated from the result of a query executed on a logical schema. For example, it can retrieve the current date and time from a database.

A variable can be created as a **global** variable or in a **project**. Global variables can be used in all projects, while project variables can only be used within the project in which they are defined.

Creating Variables

To create a variable:

1. Click the **Variables** node in a project, or the **Global Variables** node in the **Others** view.
2. Right-click and select **Insert Variable**.
3. Specify a **Name**, **Action**, **Datatype** and **Default Value**. See Variable for more information.
4. If you want the variable's value to be set by a query:
 1. Select the **Refreshing** tab.
 2. 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 by clicking .
 3. Click  to check the syntax of your expression.
 4. The **Refresh** button allows you to test the variable by executing the query immediately.
5. Click **OK**.

The variable appears in the tree view.

To delete a variable:

1. Click the variable in the appropriate tree view.
2. Right click and select **Delete**.
3. Click **OK**.

Using variables

To use a variable in a package:

Variables can be used in packages in several different ways, as follows:

- **Declaration:** When a variable is used in a package (or in certain elements of the topology which 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.
- **Refreshing:** A **Refresh Variable** step allows you to re-execute the command or query that computes the variable value.
- **Assigning:** A **Set Variable** step of type **Assign** sets the current value of a variable.
- **Incrementing:** A **Set Variable** step of type **Increment** increases or decreases a numeric value by the specified amount.
- **Conditional evaluation:** An **Evaluate Variable** step tests the current value of a variable and branches depending on the result of the comparison.
- In expressions in other steps, such as interfaces, procedures, OS Commands and so forth.

Variable scope

A variable can and should be used by explicitly specifying its scope, using the syntax `#GLOBAL.<variable name>` for **global** variables or `#<project code>.<variable name>` for **project** variables.

Using the value of the variable

Variables can be used in all Oracle Data Integrator expressions:

- Mapping,
- Filters,
- Joins,
- Constraints,
- ...

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 */
```

Note: 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`

Note: You can drag-and-drop a variable into most expressions with the Expression Editor. 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 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)

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 [Launching a scenario from an OS command](#).

Creating and Using Sequences

A sequence is a value that increments automatically when used.

Oracle Data Integrator sequences are intended primarily to compensate for the lack of native sequences on certain RDBMS. The value 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 two 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.

Note: 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.

Note: Oracle Data Integrator sequences were developed to compensate for their absence on some RDBMS. If RDBMS sequences exist, they should be used. This may prove to be faster because it reduces the dialog between the agent and the database.



Creating Sequences

To create a standard sequence:

1. Click the **Sequences** node in the project or on the node **Global Sequences**.
2. Right click and select **Insert Sequence**.
3. Enter the sequence **Name**, then select **Standard Sequence**.
4. Enter the initial **Position** and the **Increment**.
5. Click **OK**.

The sequence appears in the tree view.

To create a specific sequence:

1. Click the **Sequences** node in the project or on the node **Global Sequences**.
2. Right click and select **Insert Sequence**.
3. Enter the sequence **Name**, then select **Specific Sequence**.
4. Enter the **Increment** value.
5. Enter the names of the **Schema**, **Table** and **Column** that contain the sequence value.
6. Type in a **Filter** which will allow Oracle Data Integrator to locate a specific row in the **table**. You can use the Expression Editor by clicking . Click  to validate the expression.
7. Click **OK**.

The sequence appears in the tree view.

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`.

To delete a sequence:

1. Click the **sequence** in the tree view.
2. Right click and select **Delete**
3. Click **OK**

The sequence disappears from the tree view.

Using sequences and identity columns

Sequence Scope

Unlike for variables, you do not need to state the scope of sequences explicitly in code.

Using Sequences

The sequences can be used in all Oracle Data Integrator expressions, such as in:

- Mappings,
- Filters,
- Joins,
- Constraints,
- ...

A sequence can be used in all statements with the following syntax:

- #<SEQUENCE_NAME>_NEXTVAL - the sequence value is incremented then substituted directly into the text of the expression.

In SQL statements only, sequences can be used in binding mode with the following syntax :

- :<SEQUENCE_NAME>_NEXTVAL - the sequence value is incremented, then passed to the RDBMS as a parameter of the query.

Note: A sequence is resolved by an agent, it can therefore be incremented only when the agent itself processes one record.

For example:

- In the SQL statement `insert into fac select :NO_FAC_NEXTVAL, date_fac, mnt_fac` the sequence value will be incremented only once, even if the SQL statement processes 10,000 rows, because the agent does not process each records, but just sends the command to the database engine.

- To increment every row, the data must transit via the agent. To do this, use the **SELECT/INSERT** syntax specific to Oracle Data Integrator:

```
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 */
```

Tips for Using 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 interface.
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.

Note: Please note the following limitations:

- A column mapped with a sequence should not be checked for not null.
- Similarly, static control and flow control are cannot be performed on a primary or alternate key that references the sequence.

Identity Columns

Sequences in Oracle Data Integrator exist as a solution to emulate native sequences. 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 interface.

Note: Please note 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.

Handling Knowledge Modules

Decrypting a KM or Procedure

To decrypt a KM or a procedure:

1. Right click the KM or procedure that you wish to decrypt.
2. Select **Decrypt**.
3. In the **KM Decryption** window, either
 - select an existing encryption key file;
 - or type in (or paste) the string corresponding to your **personal key**.

A message appears when the decryption has finished.

See also:

Encrypting a KM or a Procedure

Encrypting a KM or Procedure

Encrypting a Knowledge Module (KM) or a 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 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.

To Encrypt a KM or a Procedure:

1. Right click the KM or procedure you wish to encrypt.
2. Select **Encrypt**.
3. In the **Encryption Options** window, either:

- select the **Encrypt with a personal key** option , then select an existing Encryption Key file,
 - select the **Encrypt with a personal key** option, then type in (or paste) the string corresponding to your **personal key**,
 - or let Oracle Data Integrator generate a key using the **Get a new encryption key** option.
4. The **Encryption Key** window appears when the encryption is finished. From this window, you can save the key.

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.

See also:

Decrypting a KM or a Procedure

Importing a KM

To import a Knowledge Module into Oracle Data Integrator:

1. Select the **project** into which you want to import the KM
2. Right-click on the project, and select **Import > Import Knowledge Module**
3. Specify the **Import Directory**, and select one or more **Knowledge Module Files** to Import from this directory
4. Select **Duplication** as the **Import Mode**, then click **OK**

The Knowledge Modules are imported into the work repository, and become usable within the project.

See also:

Exporting an Object

Import Modes

Importing a KM in Replace Mode

Importing a KM in Replace Mode

Knowledge modules are usually imported into new **projects** in duplication mode.

When you want to replace a KM in a project by another one and have all interfaces 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 on the **Knowledge Module**, and select **Import Replace**
3. Specify the **Knowledge Module Export File**
4. Click **OK**

The Knowledge Module is now replaced by the new one.

Warning: When replacing a Knowledge module by another one, 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 interfaces.

Warning: Replacing a KM by another one may lead to issues if the KMs are radically different. It is advised to check the interfaces' design and execution with the new KM.

See also:

Exporting an Object

Import Modes

Importing a KM

Working with Oracle Data Quality

Introduction Data Quality

A complete Data Quality system includes data profiling, integrity and quality.

- **Profiling** makes possible data investigation and quality assessment. It allows business users to get a clear picture of their data quality challenges, to monitor and track the quality of their data over time. Profiling is handled by **Oracle Data Profiling**. It allows business users to assess the quality of their data through metrics, to discover or infer rules based on this data, and finally to monitor over time the evolution of the data quality.
- **Integrity** 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. **Oracle Data Integrator** provides built-in working environment to detect these constraint violation and store them for recycling or reporting purposes. Static and Flow checks in Oracle Data Integrator are integrity checks.
- **Quality** includes integrity and extends to more complex quality processing. A rule-based engine apply data quality standards as part of an integration process to cleanse, standardize, enrich, match and de-duplicate any type of data, including names and addresses. **Oracle Data Quality for Data Integrator** places data quality and name & address cleansing at the heart of the enterprise integration strategy.

Quality process Overview

1. Create a Quality input file from **Oracle Data Integrator**, containing the data to cleanse.
2. Create an Entity in **Oracle Data Quality**, based on this file.
3. Create a **Oracle Data Quality** project cleansing this Entity.
4. Export this project from **Oracle Data Quality** for run-time.
5. Reverse-engineer the entities using the **RKM Oracle Data Quality**.
6. Use Oracle Data Quality Input and Output Files in Interfaces.
7. Run this project from Oracle Data Integrator using the **OdiDataQuality tool**.
8. Sequence the process in a package.

Create a Quality Input File

Oracle Data Quality uses as a source for the Quality project a flat file which contains the data to cleanse. This Quality input file can be created from Data Integrator and loaded from any source datastore using interfaces. This file should be a FILE datastore with the following parameters defined on the **Files** tab:

Parameter	Value
File Format	Delimited
Heading (Number of Lines)	1
Record Separator	MS-DOS
Field Separator	Other
[Field Separator] Other	, (comma sign - Hexadecimal 2C)
Text Delimiter	" (double quotation marks)
Decimal Separator	empty, not specified

For more information creating a FILE datastore, refer to the Datastores - File topic. You will find more information on loading flat files in the Using Oracle Data Integrator with Files topic.

Create an Entity

To import a data source into Oracle Data Quality for Data Integrator means to create an entity based on a delimited source file.

Step 1: Validate Loader Connections

Your administrator must set up at least one Loader Connection when he or she installs Oracle Data Quality for Data Integrator. This Loader Connection is used to access the Oracle Data Quality input file. As the input file is a delimited file, this Loader Connection should be a Delimited Loader Connection. Step 1 requires you validate this Delimited Loader Connection set up. Also verify that all the data and schema files you need are copied to the directory defined by the Loader Connection.

If you do not have access to the Metabase Manager, ask your Metabase administrator to verify the Loader Connection for you.

If you are a Metabase User and have access to the Metabase Manager, follow this procedure:

To validate a Loader Connection

1. Open the **Metabase Manager (Start > All Programs > Oracle > Oracle Data Profiling and Quality > Metabase Manager)**.
2. Verify you are in **Admin Mode**.
3. Expand the **Control Admin** node.
4. Double-click **Loader Connections**.
5. On the right, the **Loader Connections list view** displays each Loader Connection, showing its name, type, data file, and parameters. Review the information to verify that the **Loader Connection** created by your administrator is a **Delimited Loader Connection** and that the data and schema directories are pointing to the correct location.

Note: If you are a Metabase User with full Metabase privileges, you can create a new Loader Connection.

Step 2: Create Entity and Import Data

Use the **Create Entity wizard** to create an Entity. The Wizard takes you through each step, helps you to select data to load, and provides an interface for specifying connection and schema settings. It also gives you options for customizing how the data appears in an Entity.

To import a delimited source file into Oracle Data Quality for Data Integrator:

1. Copy the flat file that you want to import into **Oracle Data Quality for Data Integrator** into the data directory that you specified when you defined the Loader Connection.
2. Click on the Windows Start Menu and selecting **Programs > Oracle > Oracle Data Profiling and Quality > Oracle Data Profiling and Quality**.
3. Log in the user interface with your metabase user.
The Oracle Data Profiling and Quality user interface opens
4. From the **Main Menu**, select **Analysis > Create Entity...**
4. The **Create Entity wizard** opens in the upper right pane.
5. On the **Connection Page of the Create Entity wizard**, select the Loader Connection given to you by the administrator that you have checked in step 1.
6. Leave the default settings for the filter and the connection and click **Next**.
7. **Oracle Data Quality** connects to the data source using the Loader Connection you selected in Step 4. If the connection fails, contact your Metabase Administrator
8. In the **Entity Selection dialog**, select the data source file name you want to import in the list and click **Next**.
9. Select the schema settings for the selected data file corresponding to the parameters of the file described in the section Create a Quality Input File.
 - **Delimiter** : , (comma)
 - **Quote** : " (double quotation marks)
 - **Attribute information** : Names on first line
 - Select **Records are CR/LF terminated**
 - **Character encoding** : ascii

For more information on configuring Entities for delimited files, please refer to the *Oracle Data Quality for Data Integrator Help*.

Note : If the file is generated using Oracle Data Integrator These file format parameters should correspond to the file format specified in the Files tab of the datastore definition.

10. After you select the schema settings, click **Preview**. The **Preview mode** shows how the data will appear in the Entity, based on your selected schema settings. The data displays below in a list view. Use the Preview mode to customize how the data will appear in the new Entity.
11. When you are ready to continue, click **Close**.
12. Click **Next**. The **Load Parameters dialog** opens. Specify the parameters as follows:
 - Select the **All Rows** option
 - Leave the default **Job name**
13. Click **Next** to continue.

14. In **Confirm Settings**, review the list of settings and click **Finish** to schedule the Entity creation job. The **Schedule Job window** opens.
15. Click **Run Now**.

Step 3: Verify Entity

During the data import process, **Oracle Data Quality for Data Integrator** translates your data files into three basic components (Metabase objects): **Entities**, **Attributes**, and **Rows**.

Perform the following list of verification tasks to ensure that the data you expected has been successfully imported to a Metabase and are correctly represented in the Metabase Explorer.

1. Make sure that for every data file imported you have one corresponding Entity.
2. Make sure that the column names do not contain any special characters with the exception of underscore (`_`) or minus sign (`-`) characters. Minus signs and underscores will be translated into spaces during the data load process.
3. Make sure that for every field imported you have one corresponding Attribute.
4. Make sure that you have one Entity Row for every data row imported.

Create a Profiling Project

You can now run a **Data Profiling Project** with **Oracle Data Profiling** to find quality problems. Profiling discovers and analyzes the quality of your enterprise data. It analyzes data at the most detailed levels to identify data anomalies, broken filters and data rules, misaligned data relationships, and other concerns that allow data to undermine your business objectives.

For more information on Data Profiling see the chapter *Working with Oracle Data Profiling* in the *Oracle Data Quality for Data Integrator Help*.

Create a Quality Project

You can now create an **Oracle Data Quality Project** to validate and transform your data, and resolve data issues such as mismatching and redundancy.

Oracle Data Quality for Data Integrator is a powerful tool for repairing and correcting fields, values and records across multiple business contexts and applications, including data with country-specific origins. **Oracle Data Quality for Data Integrator** enables data processing for standardization, cleansing and enrichment, tuning capabilities for customization, and the ability to view your results in real-time.

A Quality Project cleanses input files and loads cleansed data into output files. At the end of your Oracle Data Quality project this input file may be split into several output files, depending on the data Quality project.

Important Note: A Data Quality project contains many temporary entities, some of them not useful in the integration process. To limit the Entities reversed-engineered for usage by Oracle Integrator, a filter based on entities name can be used. To use this filter efficiently, it is recommended that you rename in your quality project the entities that you want to use in **Oracle Data Integrator** in a consistent way. For example rename the entities `ODI_IN_XXX` and the output (and no-match) files `ODI_OUT_XXX`, where XXX is the name of the entity.

For more information on Data Quality projects see the chapter *Working with Oracle Data Quality* in the *Oracle Data Quality for Data Integrator Help*.

Export a Data Quality Project

Oracle Data Integrator is able to run projects exported from Oracle Data Quality. Once the Data Quality project is complete, you need to export it for Oracle Data Integrator. The exported project contains the data files, Data Dictionary Language (DDL) files, settings files, output and statistics files, user-defined tables and scripts for each process module you in the project. An exported project can be run on UNIX or Windows platforms without the user interface, and only requires the Oracle Data Quality Server.

To create a batch script

1. From the **Explorer** or **Project Workflow**, right-click the **Oracle Data Quality project** and select **Export... > ODQ Batch Project > No data**.
2. In **Browse for Folder**, select or make a folder where you want the project to be exported.
3. Click **OK**.
A message window appears indicating that the files are being copied.

This export process creates a folder named after the metabase (<metabase_name>) at the location that you specified. This folder contains a `projectN` sub-folder (where N is the project identifier in Oracle Data Quality). This project folder contains the following folders among others:

- **data**: This folder is used to contain input and output data as well as temporary data files. These files have a .DAT extension. As you specified No data for the export, this folder is empty.
 - **ddl**: This folder contains the entities metadata files (.DDX and .XML). These files describe the data files' fields. They are prefixed with `eNN_`, where NN is the Entity number. Each entity is described in two metadata files. `eNN_<name of the entity>.ddx` is the description of the entity with possible duplicated columns (suitable for fixed files). `eNN_<name of the entity>.csv.ddx` is the description of the entity with non-duplicated columns (suitable for fixed and delimited files). It recommended to use these files for the reverse-engineering process.
 - **scripts**: This folder contains the configuration file `config.txt`, and the batch script `runprojectN`. This script runs the quality process and is the one that will be triggered by Oracle Data Integrator.
 - **settings**: This folder contains settings files (.ddt, .sto, .stt, .stx).
4. After the message window has disappeared, examine the folder you have specified and check that all folders and files are correctly created.
 5. Move the exported project to a folder on the run-time machine. This machine must have the Oracle Data Quality Server installed at it will run the quality project.
 6. Open with a text editor the batch script (`runprojectN`) and the configuration file (`config.txt`) in the `/scripts` sub-folder of your `projectN` folder.
 7. Perform the following changes to configure the run-time directory in the project.
 - In `config.txt`, specify the location (absolute path) of the directory containing the `projectN` folder for the `DATABASE` parameter.
 - In `runprojectN`, specify the location (absolute path) of the `ProjectN` directory for the `TS_PROJECT` parameter.

For example, if you have the `config.txt` and `runproject2.*` files located in `C:/oracle/oracledq/metabase_data/metabase/oracledq/project2/scripts /`, you should specify

- in `config.txt`:
`DATABASE = "C:/oracle/oracledq/metabase_data/metabase/oracledq"`
- In `runprojectN.*`:
`set`


```
TS_PROJECT=C:\oracle\oracledq\metabase_data\metabase\oracledq\project2
```

8. Save and close the `config.txt` file.
9. In `runprojectN` uncomment the very last line of the file (remove the `::` character at the beginning of the last line).
10. Save and close the `runprojectN` file.
11. Oracle Data Integrator uses CSV formatted files (typically, comma-delimited with one header line) to provide the data quality project with input data, and expects output data to be in the same format.

In the `/settings` directory, open with an editor the settings file corresponding to the first process of your project. This file is typically named `eN_transfmr_p1.stx` (where N is the internal ID of the entity corresponding to the quality input file) if the first process is a transformer.

12. Change the following input parameters in the settings file:

- In `DATA_FILE_NAME`, specify the name and location (absolute path) of your quality input file in run-time.
- In `FILE_DELIMITER`, specify the delimiter used in the quality input file.
- In `START_RECORD`, specify the line number where data starts. For example, if there is a 1 line header, the value should be 2.

For example, if you have the `customer_master.csv` quality input file (comma-separated with one header line) located in

`C:/oracle/oracledq/metabase_data/metabase/oracledq/Data/`, you should edit the following section

```
<CATEGORY><INPUT><PARAMETER><INPUT_SETTINGS>
  <ARGUMENTS>
  <ENTRY>
    <ENTRY_ID>1</ENTRY_ID>
    <FILE_QUALIFIER>Customer_Master(1)</FILE_QUALIFIER>
    <DATA_FILE_NAME>$(INPUT)/e1_customer_master.dat</DATA
_FILE_NAME>
    <DDL_FILE_NAME>$(DDL)/e1_customer_master.ddx</DDL_FIL
E_NAME>
    <FILE_DELIMITER/>
    <USE_QUOTES_AS_QUALIFIER/>
    <START_RECORD/>
```

as shows below

```
<CATEGORY><INPUT><PARAMETER><INPUT_SETTINGS>
  <ENTRY>
    <ENTRY_ID>1</ENTRY_ID>
    <FILE_QUALIFIER>Customer_Master(1)</FILE_QUALIFIER>
    <DATA_FILE_NAME>C:\oracle\oracledq\metabase_data\meta
base\oracledq\Data\customer_master.csv</DATA_FILE_NAME>
    <DDL_FILE_NAME>$(DDL)/e1_customer_master.ddx</DDL_FIL
E_NAME>
    <FILE_DELIMITER>,</FILE_DELIMITER>
    <USE_QUOTES_AS_QUALIFIER/>
```

```
<START_RECORD>2</START_RECORD>
```

12. Save and close the settings file.
13. Also in the `/settings` directory, open the file that corresponds to the settings of the process generating the output (cleansed) data. Typically, for a cleansing project which finishes with a Data Reconstructor process, it is named with `eNN_datarec_pXX.stx`. Change the following value in the settings file to give the full path of the generated output file.

```
<CATEGORY><OUTPUT><PARAMETER>  
  <OUTPUT_SETTINGS>  
    <ARGUMENTS>  
      <FILE_QUALIFIER>OUTPUT</FILE_QUALIFIER>  
      <DATA_FILE_NAME>C:\oracle\oracledq\metabase_data\metabase\or  
acledq\Data\customer_master_cleansed.csv</DATA_FILE_NAME>  
      <DDL_FILE_NAME>$(DDL)/e36_us_datarec_p11.ddx</DDL_FILE_NAME>
```

14. Save and close the settings file.
15. If you have several data quality processes that generate useful output files (for example, one data reconstructor per country). Repeat the two previous steps for each of these processes.

Reverse-engineer Entities

In order to provide the Quality process with input data and use its output data in data integrator's integration processes, it is necessary to reverse-engineer these Entities. This operation is performed using a customized reverse-engineering method based on the Oracle Data Quality RKM. The RKM reads metadata from the `.ddx` files located in the `/ddl` folder of your data quality project.

To reverse-engineer the Entities of a data Quality project:

1. Import the **RKM Oracle Data Quality** into your Oracle Data Integrator project.
2. Insert a physical schema for the File technology in Topology Manager. Specifying for both, the Directory (**Schema**) and the Directory (**Work Schema**), the absolute path of your data folder.
For example
`C:\oracle\oracledq\metabase_data\metabase\oracledq\projectN\data`
This directory must be accessible to the agent that will be used to run the transformations. **Oracle Data Integrator** will look in the schema for the source and target data structures for the interfaces. The RKM will access the output data files and reverse-engineer them.
3. Create a File model and reverse the `/ddl` folder.
 1. Connect to **Designer**.
 2. Select **Models** in the tree.
 3. Right-click then select **Insert Model**.
 4. In the **Definition tab**, fill in the **Name field**.
 5. In the **Technology field**, select **File**.
 6. In the **Logical Schema field**, select the **Logical Schema** on which your model will be based.
 7. Go to the **Reverse tab** and select:
 1. **Customized Reverse**
 2. Reverse-engineering Context
 3. **Type of objects to reverse-engineer**: Table
 4. **KM** : Select the RKM Oracle Data Quality

8. Set the RKM options:

Parameter	Default Value	Description
DDX_FILE_NAME	*.ddx	Mask for DDX Files to process. If you have used a naming convention in the Quality project for the Entities that you want to use, enter a mask that will return only these Entities. For example, specify the <code>ODI*_csv.ddx</code> mask if you have used the <code>ODI_IN_XX</code> and <code>ODI_OUT_XX</code> naming convention for your input and output entities.
FILE_FORMAT	D	Format used for the generated file datastore: <ul style="list-style-type: none"> • D (Delimited): The fields of a line are separated by a record separator. • F (Fixed): The fields of a line are not separated, but their length is fixed.
RECORD_SEPARATOR	MS-DOS	One or several characters separating lines (or records) in the file: <ul style="list-style-type: none"> • MS-DOS: DOS carriage return. • UNIX: UNIX carriage return. Any other value corresponds to an empty record separator.
FIELD_SEPARATOR	, (comma)	Character(s) used to separate the fields in a record.
USE_FRIENDLY_NAMES	No	Set this option to Yes if you want the Reverse-Engineering process to generate user-friendly names for datastore columns based on the field name specified in the DDX file.
USE_LOG	Yes	Set to Yes if you want the reverse-engineering process activity be logged in a log file.
LOG_FILE_NAME	/temp/reverse.log	Name of the log file.

4. Click **Apply**. The model is created, but contains no datastores yet.

5. Click **Reverse**. Now, the model contains datastores that you can see in the Models view.

Use Oracle Data Quality Input and Output Files in Interfaces

You can now create in Oracle Data Integrator interfaces sourcing or targeting the data Quality input and output files.

For example, you can:

- Create interfaces to load the input file using datastores from various sources.
- Create interfaces to re-integrate the output data back into the sources after cleansing.

Run a Quality Project from Data Integrator

The `OdiDataQuality` tool executes the batch file to run the Oracle Data Quality project. This tool takes as a parameter the path to the `runprojectN` script file. It can run either in synchronous (the tool waits for the quality process to complete) or asynchronous mode.

For more information on the **OdiDataQuality** tool and its parameters, refer to the `OdiDataQuality` topic.

Sequence the process in a package

Create a package in Oracle Data Integrator sequencing the following process:

1. One or more Interfaces creating the Quality input file, containing the data to cleanse.
2. **OdiDataQuality** tool step launching the Oracle Data Quality process.
3. One or more Interfaces loading the data from the Oracle Data Quality output files into the target datastores.

Using Web Services

Web Services can be invoked:

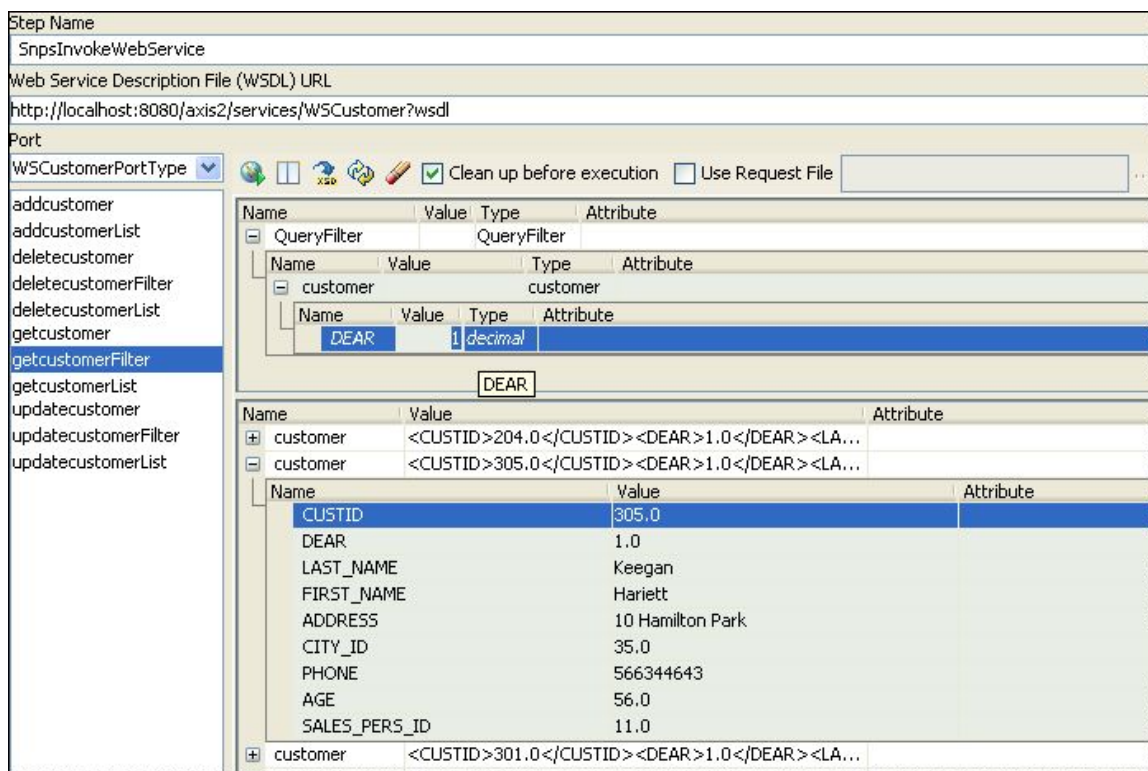
- For testing Data Services: This method allows you to check that your data services are running correctly. See *Setting Up Data Services* for more information.
- In packages using the `SnpsInvokeWebService` tool: This tool allows you to invoke any third party web service, and save the response in a XML file that can be processed with Oracle Data Integrator.

A specific graphical interface exists to help you create Web Service requests.

Graphical interface for Web Services

The graphical interface for Web Services appears as shown below:

- At the top, the basic properties: Step name, WSDL location and port.
- At bottom left, the list of operations for the selected port. A second tab displays the HTTP request options.
- At the bottom right, the SOAP editor displays the web service request and response.



Basic properties

The basic properties are:





- **Step Name:** The name of the package step.
- **Web Service Description File (WSDL) URL:** This file describes the Web Service and the format of the requests and responses.
- **Port:** If the WSDL defines several ports on which Web Services can be accessed, the list of ports appears here.



In the **Options** tab, the HTTP requests options appear:

- **Timeout:** The web service request waits for a reply for this time before considering that the server will not provide a response and an error is produced.
- **HTTP Authentication:** If you check this box, you should provide a user and password to authenticate on your HTTP server.

Toolbar

The toolbar provides the following features:

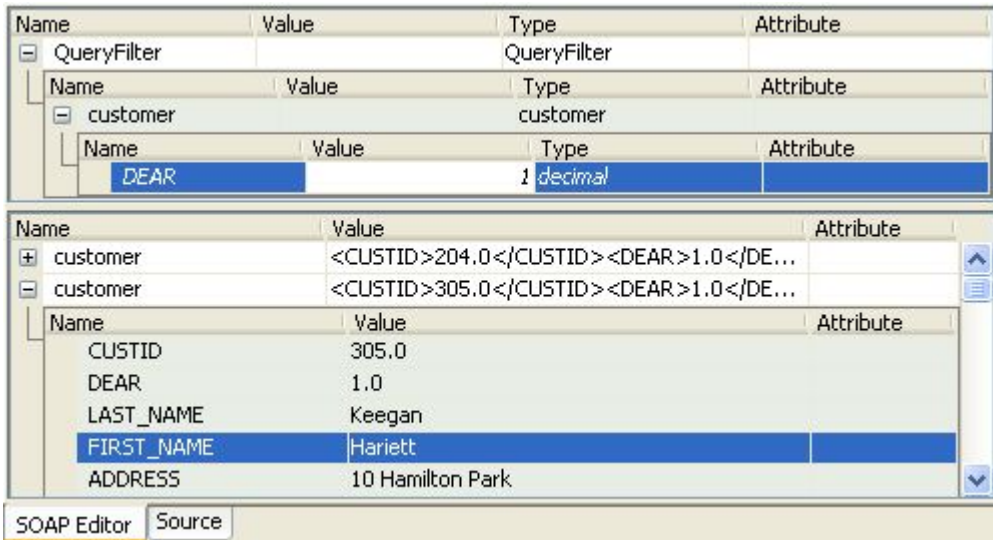
-  **Invoke Web Service:** Invokes immediately the current Web Service, displaying the response in the SOAP editor.
-  **Switch Panel Position:** Tiles vertically or horizontally the SOAP editor.
-  **Export Response XSD:** Saves the current response XML schema description to a file.
-  **Restore Default Request:** Discards the current request and reverts to a default, blank request structure.

-  **Delete Empty Optional Components:** Removes all blank optional elements from the query. This may be necessary to construct a valid query.
- **Clean up before execution:** Automatically delete empty optional elements in the SOAP request when the  **Invoke Web Service** button is used. This checkbox has no effect on package steps at run-time.
- **Use Request File:** Uses a SOAP request stored in a file instead of the contents of the SOAP editor.
- **Timeout (ms)** : Specifies a maximum period of time to wait for the request to complete.

SOAP Editor

The SOAP Editor allows you to graphically build the SOAP request for the Web Service and display the response.

If creating a `SnpsInvokeWebService` step, this SOAP request filled in the editor is saved with the step.




The screenshot shows the SOAP Editor interface with two main panels. The upper panel displays a hierarchical tree of query elements:

Name	Value	Type	Attribute
QueryFilter		QueryFilter	
customer		customer	
DEAR		1 decimal	

The lower panel displays the structure of the response:

Name	Value	Attribute
customer	<CUSTID>204.0</CUSTID><DEAR>1.0</DE...	
customer	<CUSTID>305.0</CUSTID><DEAR>1.0</DE...	
CUSTID	305.0	
DEAR	1.0	
LAST_NAME	Keegan	
FIRST_NAME	Hariett	
ADDRESS	10 Hamilton Park	

At the bottom of the editor, there are two tabs: "SOAP Editor" (selected) and "Source".

The upper part of the editor shows the hierarchical structure of the query, the lower part shows the structure of the response. This arrangement can be changed using the  **Switch Panel Position** button.

The raw XML source of the SOAP request and response are shown on the **Source** tab.

In the editor, you can fill in the **Value** (and optionally the **Attributes**) for each element of your request.

Warning: An empty element is passed as is to the Web service. For strings, this corresponds to an empty string. For numbers or date types, this may cause an error to occur. If you want to send a null string, number or date, it is recommended to use the `nil="true"` attribute.

To remove empty elements, use the  **Delete Empty Optional Components** option.

Optional elements are displayed in *italic*. Repeatable elements are labelled with **...(n*)** after the name.

Right-click any element to perform one of the following operations, if possible:

- **Duplicate content** - copies the structure and content of the element.
- **Duplicate structure** - copies the structure but leaves all fields blank.

- **Delete** - deletes the element.
- **Export Request** - exports the entire soap request to an XML file.

Results


This part of the interface appears only when using a SnpsInvokeWebService step in a package, to control how the response is written to a XML file.

The screenshot shows a configuration dialog with four fields:

- File Mode:** A dropdown menu set to 'NEW_FILE'.
- Result File:** A text box containing 'c:\DataServices\wscustomer.txt' with a browse button ('...').
- XML Charset:** A text box containing 'UTF 8'.
- Java Charset:** A text box containing 'UTF-8'.



- **File Mode** (-RESPONSE_MODE): One of NEW_FILE, FILE_APPEND, NO_FILE
- **Result File** (-RESPONSE_FILE): The name of the result file to write.
- **XML Charset** (-RESPONSE_XML_ENCODING): The name of the character encoding to write into the XML file.
- **Java Charset** (-RESPONSE_FILE_CHARSET): The name of the character encoding used when writing the file.

See SnpsInvokeWebService for more information on these parameters.

Note: The result file parameters are only taken into account at run-time. No result file is generated when using the  **Invoke Web Service** button.

Invoking a Web Service


To create a Web Service request:

1. Create a SnpsInvokeWebService tool step in a package, or right-click a datastore and select **Test Web Service** in the popup menu.
2. Fill in the location of the WSDL. You can use either:
 - A URL for a WSDL that has been deployed to a server (e.g. `http://host:8080/axis2/services/WSCustomer?wsdl`)
 - A local file location (e.g. `file:///C:/DataServices/WSCustomer.wsdl`)
3. Choose a **Port**, if more than one is available. If no ports are available, there is a problem with the WSDL.
4. Choose an **Operation** from the list on the left.
5. Fill in request in the **SOAP Editor**. You can also use an external request file instead.
6. (Optional) Click the  **Delete empty optional elements** button to remove optional elements which have not been filled in. Some Web Services treat blank elements as invalid.
7. Click the  **Invoke Web Service** button to immediately invoke the Web Service. The response is shown in the SOAP Editor.
8. If you are creating a SnpsInvokeWebService tool step, define the result file parameters then click **Apply** to save your step.

Processing a Web Service Response

When using SnpsInvokeWebService to call a web service, the response is written to an XML file.

Processing this file can be done with Oracle Data Integrator, using the following guidelines:

- Invoke the web service once and use the  **Export Response XSD** option to export the XML schema.
- Create an XML model for the SOAP response, based on this XML schema file and reverse-engineer the XSD to have your model structure. See [Creating and Reverse-Engineering a Model for an XML file](#) for more information.
- You can now on process the information from your responses using regular Oracle Data Integrator interfaces. See [Choosing the Right KMs for XML files](#) for more information.

Note: Each XML file is defined as a model in Oracle Data Integrator. It is recommended to use model folders to arrange them. See [Organizing Models into Model Folders](#) for more information.

Using User Functions

User functions allow one to define customized functions that can be used in interfaces or procedures.

These functions are implemented in one or more technologies.

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 order will be executed, the `NullValue` function will be replaced by one syntax or the other.

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.

Create a user function

To create a user function:

1. Click the **user functions** node in the **project** or in **projects**
2. Right click and select **Insert User Function**.
3. Fill in the following fields:
 - **Name:** Name of the user function. 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. Example:
`NullValue($(variable), $(default))`
4. Click **OK**.

The function appears in the tree view. Since it has no implementation, it is unusable.

To create an implementation:

1. In the **implementations** tab of the **user function**, click **add**.
2. In the **Implementation syntax**, type the code of the implementation. For instance:
`nl($ (variable) , $ (default))`
3. Check the boxes for the implementation's **Associated technologies**
4. Check the **Automatically include new technologies** box if you want the new technologies to use this syntax.
5. Click **OK**.

To change an implementation:

1. in the **implementations** tab of the **user function**, select an **implementation**, then click **Edit**.
2. Change **Implementation syntax** and the **Associated technologies** to this implementation
3. Check the **Automatically include new technologies** box if you want the new technologies to use this syntax.
4. Click **OK**.

To remove an implementation:

1. In the **implementations** tab of the **user function**, select an **implementation** then click **Delete**.

Using a User functions

The user functions can be used in all Oracle Data Integrator expressions:

- Mapping,
- Filters,
- Joins,
- Constraints,
- ...

A user function can be used directly by specifying its **syntax**. Example:`NullValue (CITY_NAME, 'No City')`

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. Select an object in the **Projects** or **Models** tree view.
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 view. 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 view 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 below for more information.

To flag an object with string, numeric and date markers:

1. Double click the object in the **Projects** or **Models** tree view.
2. In the object edition windows, select the **Markers** tab
3. Click the  **Insert a Marker** button.
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 view.

Filtering Using Markers


Markers can be used for informational purposes (for instance, 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 *Generate a group of Scenarios* 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 the Designer tree view, click the **Markers** node in the **Project**, or the **Global Markers** node in the **Other**. view.
2. Right-click and select **Insert Marker Group**.
3. Enter the **Group Name**, then define its **Display Properties** and **Attributes**.
4. Click the  **Insert a Marker** button 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.

5. Click **OK**.

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  icon appears next to it.

To edit an object's memo:

1. Right click on the object.
2. Select **Edit Memo**.
3. The object is opened, and the **Memo** tab 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:

1. Deselect the **Show Markers and Memo Flags** option from the **View** menu. This preference is stored on a per-machine basis.

Using Cross-References

Objects in Oracle Data Integrator (datastores, models, interfaces, etc) are interlinked by relationships varying from simple usage associations (an integration interface uses knowledge modules) to complex ones such as code-interpretation relationships (a variable is used in the mappings or filters of an interface). 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 integration interfaces 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.

To browse cross-references:

Cross-references may be browsed from the Designer tree views:

- In the **Projects** and **Others** tree views, the  **Used in** node appears under an object node with the list of 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 interfaces using this variable in the mappings, filters, etc will be displayed.

- In the **Models** tree view, the  **Used in** 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 interface, or in package steps. The  **Used to Populate** and  **Populated By** nodes display the datastores used to populate, or populated by, the current datastore.

These cross-referenced nodes may be expanded. The referencing or referenced objects may 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. A typical situation is when restoring an old version of a project without restoring all the associated models used in its integration interfaces.

Such a situation causes **Missing References** errors messages in Oracle Data Integrator when opening the objects (for example, the interfaces) which reference nonexistent objects. An object with missing cross-references is marked in the tree view 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. Edit the object with the missing reference marker. In the edition window, select the **Missing References** tab.
2. 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:

- Importing/restoring the missing referenced object. See Version Management and Import/Export sections 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).

Important: If a text (such as an interface mapping/join/filter, a procedure command, etc) contains a missing reference, the first change applied to this text is considered without any further check as fixing the missing cross-references on this text.



Using Version Management

Oracle Data Integrator provides project managers with 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**.

Important Note: Version management is supported for master repositories installed on the following database engines: Hypersonic SQL, IBM DB2 UDB, IBM DB2/400, Informix, Microsoft SQL Server, Oracle, Sybase AS Anywhere, Sybase AS Enterprise.

Working with Object Flags

When an object in Designer is created or modified, a flag is displayed in the tree view on the object icon to indicate its status:

-  : Object status is **inserted**.
-  : 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 (see versions for more informations), 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 of a project can be checked in as versions:

- Project, Folder
- Package, Scenario
- Interface, Procedure, Knowledge Module
- Sequence, User Function, Variable
- Model

To check in a version:

1. Select the object for which you want to check in a version.
2. Right-click, then select **Version > Create**.
3. A window appears. In this window, the **Previous Versions (>>)** button expands 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 details for this version into the **Description** field.
6. Click **OK**.

When a version is checked in, the flags for the object are reset.

To display previous versions of an object:

- When editing the object, the **Version** tab provides a list of versions checked in, with the check in date and the name of the user who performed the check in operation.

To restore a version:

1. Select the object for which you want to restore a version.
2. Right-click, then select **Version > Restore**.
3. A window appears with the list of existing versions.
4. Select the version you want to restore and then click **OK**.
5. Click **OK** to confirm the restore operation.

Warning: Restoring a version cannot be undone. It permanently erases the current object and replaces it by the selected version.

To browse versions:

Oracle Data Integrator contains a tool, the **Version Browser**, which is used to display the versions stored in the repository.

1. In the Designer menu, select **File > Version Browser...**
2. Use the object type and object name drop-down boxes to filter the objects for which you want to display the list of versions.

From the Version Browser, you can **Restore**, **Export** to XML file or **Delete** existing versions.

Note: The Version Browser displays the versions that existed when you opened it. Click the Refresh button to view all new versions created since then.

To delete a version with the Version Browser:

1. Open the Version Browser.
2. Select the version you want to delete.
3. Right-click, then select **Delete**.

The version is deleted.

To restore a version with the Version Browser:

1. Open the Version Browser.
2. Select the version you want to restore.
3. Right-click, then select **Restore**.
4. Click **OK** to confirm the restore operation.

The version is restored in the repository.

To export a version with the Version Browser:

This operation exports the version to a file without restoring it. This export can be imported in another repository.

1. Open the Version Browser.
2. Select the version you want to export.
3. Right-click, then select **Export**.
4. Select the **Export Directory** and specify the **Export Name**. Check the **Replace Existing Files without Warning** checkbox to erase existing export files without confirmation.
5. Click **OK**.

The version is exported in the given location.

To view the differences between two versions, use the version comparison tool.

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 dependant 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** tree view in Designer.

The following objects may be added into solutions:

- Projects
- Models
- Scenarios
- Global Variables, User Functions and Sequences.

To create a solution:

1. Open the **Solutions** tree view.
2. Right-click, then select **Insert Solution**. A **New Solution** window appears.
3. Enter the **Name** of your solution and a **Description**.
4. Click **Apply**.

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

To work with elements in a solution:

- To add an element, drag the object from the tree view into the **Elements** list in the solution window.
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.
- 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.
- 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.
- Pressing the **Synchronize** option automatically adds required elements, removes unused elements and brings the solution up to date.

To synchronize a solution:

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.

1. Open the solution you want to synchronize.
2. Click the **Synchronize** button.
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 do this regularly to keep the solution contents up-to-date. You should also do it before checking in a solution version.

To check in and restore a solution:

- The procedure for checking in and restoring a solution version is similar to the method used for single elements. See version for more details.

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

Versions in Operator

You can use solutions to import scenarios into production in Operator.

To restore a scenario from a solution:

1. Go to the **Solutions** tab
2. Double-click a solution to open it
3. Select a scenario. Other elements such as projects and interfaces cannot be restored.
4. Press the **Restore** button.

The scenario is now accessible in the **Scenarios** tab.

You can also use the **Version Browser** to restore scenarios.

Version Comparison Tool

Oracle Data Integrator provides developers with 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 view:** The tree view 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, Interface, 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.

Using the Version Comparison Tool

Once the version of an object is created, the comparison tool can be used at different points in time.

Creating or checking in a version is covered in the topic Using Version Management.

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 view

1. From the **Projects** tree view in Designer, select the object whose versions you want to compare.
2. Right-click the object.
3. Select **Version > Compare**.
4. The **Compare with** window appears.
5. Select the version with which you want to compare the current version.
6. Click **OK**.
7. The version comparison tool opens.

Via the Versions tab of the object

1. In Designer, open 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.

Via the Version Browser

1. In Designer, select **File > Version Browser**
2. Select the two versions you want to compare.
3. Right-click and select **Compare**.
4. The version comparison tool opens.

The comparison window 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	modified (the value inside of this field 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

The version comparison tool provides two different types of filters for customizing the comparison results:

- **Object filters:** By checking the corresponding check boxes (**New** and/or **Deleted**) you can decide whether you want only newly added and/or deleted objects to be displayed.
- **Field filters:** By selecting the corresponding radio button (**All fields**, **Modified fields** or **No field**) you can decide whether you want all fields, modified fields or only objects (no fields) to be displayed.

Generating and Printing a Report of your Comparison Results

To generate a report in Designer:

1. In the version comparison window, click the **printer** icon.
2. In the Report Generation window, 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.

If the **Open the file after the generation** option is selected, Oracle Data Integrator will open Acrobat® Reader™ to view the generated report.

Note: In order to view the generated report, you must specify the location of **Acrobat® Reader™** in the user parameters.

5. Click **Generate**.

A report is written to the file specified in step 3, in **Adobe™ PDF** format.

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 and Object Locking. This two methods can be used simultaneously or separately.

Concurrent Editing Check

A user parameter, **Check for concurrent editing**, can be set to true to perform to prevent you from erasing the work performed by another user on the object you try to save. See User Parameters for more information.

If this parameter is set to true, 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

Automatic Object Locking

This mechanism is automatically activated. When an object is opened in a user interface, a popup window appears to ask if you want to lock the object. 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.

An object locked by you appears with a yellow lock icon. An object locked by another object 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.

Note that these windows are configured by the **Lock object when opening** and **Unlock object when closing** user parameters. See User Parameters for more information.

Releasing locks when closing the user interface

When closing Oracle Data Integrator, a window appears asking to unlock or save objects that you have locked or kept opened.

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.

Manually manage locks

You can also manually manage locks on objects.

To manually lock an object:

1. Select the object in the tree view
2. Right-click, then select **Locks > Lock**.

A lock icon appears after the object in the tree view.

To manually unlock an object:

1. Select the object in the tree view
2. Right-click, then select **Locks > Unlock**.

A lock icon appears after the object in the tree view.

To manage all locks:

1. Select **File > Locked objects...**

All locked objects that you can unlock appear in a new window. You can unlock or lock objects from this list..

Important: A user with the **Supervisor** privilege can remove locks for all other users.

Execution

Executing a Scenario from Designer or Operator

This mode of execution consists of launching a scenario from **Designer** or **Operator**.

To launch a scenario from Designer or Operator:

1. Select the required scenario in the **Projects** view (for Designer) or the **Scenarios** view (in Operator).
2. Right-click, then select **Execute**
3. Select the **Context** and the required **Agent**. To execute the scenario on the local machine, choose the **Local (without agent)** option.

4. Click **OK**.
5. If the scenario uses variables as parameters, choose which values to assign them. Selecting "Last value" for a variable uses its current value, or default value if none is available.
6. When Oracle Data Integrator has created and launched the new session, the message **Session started** is displayed.
7. You can monitor the execution of the session in **Operator**.

Executing a Scenario from an OS Command

You can launch a scenario from an operating system command.

To launch a scenario from an OS command:

1. Launch a **Shell** (Unix), a **Command Prompt** (Windows) or a **QSH** session (AS/400)
2. In this shell, launch the "startscen.bat" (on Windows) or startscen.sh (on Unix and AS/400) command.

Syntax: startscen <Name> <Version> <Context code> [<Log_Level>] [-SESSION_NAME=<session name>] [-KEYWORDS=<keywords>] [-NAME=<agent_name>] [-v=<trace level>] [<variable>=<value>]*

Warning: 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 instance :

```
startscen.bat SCEN 001 GLOBAL "-v=5" "PROJ1.STATE=MICHIGAN" (Windows)
./startscen.sh SCEN 001 GLOBAL -v=5 PROJ1.STATE=MICHIGAN (Unix)
```

The table below 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.

Parameters	Description
<Name>	Name of the scenario (mandatory).
<Version>	Version of the scenario (mandatory). If the version specified is -1, the last version of the scenario is executed.
<Context>	Execution context of the scenario (mandatory).
-v=<trace level>	Allows a trace to be displayed (verbose mode). There are five trace levels: <ol style="list-style-type: none"> 1. Displays the start and end of each session 2. Displays level 1 and the start and end of each step 3. Displays level 2 and each task executed 4. Displays the SQL queries executed, as well as level 4 5. A complete trace, often requested during a support call As some of these traces can be voluminous, it is recommended that you redirect them to a text file using the following command:

	<p>in Windows: "-v=5" > trace.txt</p> <p>in Unix: -v=5 > trace.txt</p>
<p><Log_Level></p>	<p>Defines the logging level for this scenario's execution. All the tasks whose logging level is lower than or equal to this value will be saved in the log at the end of execution. In the event of the interface ending abnormally, all tasks, regardless of this value, will be saved.</p> <p>This parameter is in the format LEVEL<n> where <n> is the expected logging level, between 0 and 5. The default log level is 5.</p> <p>Example : startscen.bat SCENAR 1 GLOBAL LEVEL5</p>
<p>- SESSION_NAME=<session_name> ></p>	<p>Name of the session that will appear in the execution log.</p>
<p>-KEYWORDS=<keywords></p>	<p>List of keywords attached to this session. These keywords make session identification easier. The list is a comma-separated list of keywords.</p>
<p>-NAME=<agent_name></p>	<p>Agent name that will appear in the execution log for this session.</p>
	<p>Caution: This agent name does not correspond to a logical or physical agent in the topology. This parameter does not enable to run a session on a remote agent. The SnsStartScen API provides this feature.</p>
<p><variable>=<value></p>	<p>Allows one 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>.</p> <p>This parameter can be repeated to assign several variables.</p>
<p>Work repository connection parameters</p>	<p>These parameters, which are used to specify the connection to the work repository, are specified in the file odiparams.</p>
<p>-SECU_DRIVER=<driver name></p>	<p>JDBC driver to access the master repository. For example: oracle.jdbc.driver.OracleDriver</p>
<p>-SECU_URL=<url></p>	<p>JDBC URL to access for the master repository. For example: jdbc:oracle:thin:@168.67.0.100:1522:ORCL</p>
<p>-SECU_USER=<user></p>	<p>User for the master repository connection (the database user).</p>

-SECU_PASS=<password>	Password for the master repository connection user. This password must be encrypted using the command <code>agent ENCODE <password></code> .
-ODI_USER=<user>	Oracle Data Integrator User whose rights will be used to execute the scenario. If this user has no rights on the context for instance, the execution will not work. Default value: SUPERVISOR
-ODI_PASS=<password>	Password of the Oracle Data Integrator User. This password must be encrypted using the command <code>agent ENCODE <password></code> .
- WORK_REPOSITORY=<work repository name>	Name of the work repository containing the scenarios to be executed.

Managing Executions through Web Services

This chapter explains how to use a web service to perform the following Oracle Data Integrator execution tasks :

- Execute a Scenario
- Restart a Session
- List Sessions
- List Contexts

An example of a simple SOAP request as well as the corresponding SOAP response will be given for every execution task.

Note the following:

- **Before considering the use of ODI Web Services**, you must have a working knowledge of Oracle Data Integrator and the creation and execution of scenarios.
- **Prior to any execution**, you must Install the Oracle Data Integrator Public Web Service on your web service container.
- **Concerning the SOAP request:**
 - The web service accepts password in a plain text format. Consequently, it is strongly recommended to use secured protocols (HTTPS) to invoke web services over a non-secured network.
 - The agent must be accessible from the web service container, and it must have access to the ODI repositories.
 - The entire `RepositoryConnection` structure is not mandatory in the SOAP request if the agent is already connected to the work repository. For a scheduler agent, for example, the agent has already all the repository connection information. In this case, only the `ODIUser` and `ODIPassword` sub-elements are required.
- **Concerning the SOAP response:**

The response depends on the `SyncMode` used for the scenario execution:

 - In **synchronous** mode (`SyncMode=1`), the response is returned once the session has completed, and reflects the execution result.

- In **asynchronous** mode (`SyncMode=2`), the response is returned once the session is started, and only indicates the fact whether the session was correctly started or not.

Executing an ODI Scenario Using a Web Service

The Oracle Data Integrator Public Web Services feature allows you to run a scenario from a web service.

To execute a scenario using a web service, you can invoke the `OdiInvoke` web service with the appropriate parameters. The web service port to use is called `invokeScenario`. This web service commands an agent to connect to a given work repository, and to start a specific ODI scenario. Parameters are similar to the ones used when Executing a Scenario from an OS Command .

A simple SOAP request for this web service is provided below. Refer to the WSDL file for the comprehensive list of parameters.

```
<invokeScenarioRequest>
  <invokeScenarioRequest>
    <RepositoryConnection>
      <!-- Connection information to the repository -->
      <!-- This example is an Oracle Repository -->
      <JdbcDriver>oracle.jdbc.driver.OracleDriver</JdbcDriver>
      <JdbcUrl>jdbc:oracle:thin:@srv011:1521:ORA10G</JdbcUrl>
      <JdbcUser>repo</JdbcUser>
      <JdbcPassword>snp65934</JdbcPassword>
      <OdiUser>SUPERVISOR</OdiUser>
      <OdiPassword>PASS</OdiPassword>
      <WorkRepository>WORKREP</WorkRepository> <!-- Work
      Repository CODE -->
    </RepositoryConnection>
    <Command>
      <!-- Scenario, version and execution context -->
      <ScenName>LOAD_DW</ScenName>
      <ScenVersion>001</ScenVersion>
      <Context>GLOBAL</Context>
      <SyncMode>1</SyncMode>
    </Command>
    <Agent>
      <!-- Agent that will execute the scenario -->
      <Host>srv001</Host>
      <Port>20910</Port>
    </Agent>
  </invokeScenarioRequest>
</invokeScenarioRequest>
```

The scenario execution returns a SOAP response as shown below:

```
<odi:invokeScenarioResponse xmlns:odi="xmlns.oracle.com/odi/OdiInvoke">
  <odi:CommandResultType>
```

```
<odi:Ok>true</odi:Ok>
<odi:SessionNumber>1148001</odi:SessionNumber>
</odi:CommandResultType>
</odi:invokeScenarioResponse>
```

Restarting an ODI Session Using a Web Service

The Oracle Data Integrator Public Web Services allow you to restart an ODI session from a web service.

To restart a session using a web service, you can invoke the `OdiInvoke` web service. The port to use is called `invokeSession`.

This web service commands an agent to connect a given work repository, and to restart a specific session.

A simple SOAP request for this web service is provided below. Refer to the WSDL file for the comprehensive list of parameters.

```
<invokeSessionRequest>
  <invokeSessionRequest>
    <RepositoryConnection>
      <!-- Connection information to the repository -->
      <!-- This example is an Oracle Repository -->
      <JdbcDriver>oracle.jdbc.driver.OracleDriver</JdbcDriver>
      <JdbcUrl>jdbc:oracle:thin:@srv011:1521:ORA10G</JdbcUrl>
      <JdbcUser>repo</JdbcUser>
      <JdbcPassword>snp65934</JdbcPassword>
      <OdiUser>SUPERVISOR</OdiUser>
      <OdiPassword>PASS</OdiPassword>
      <WorkRepository>WORKREP</WorkRepository> <!-- Work
      Repository CODE -->
    </RepositoryConnection>
    <Command>
      <!-- Session Number -->
      <SessionNumber>3001</SessionNumber>
      <SyncMode>1</SyncMode>
    </Command>
    <Agent>
      <!-- Agent that will execute the session -->
      <Host>srv001</Host>
      <Port>20910</Port>
    </Agent>
  </invokeSessionRequest>
</invokeSessionRequest>
```

The session execution returns a SOAP response as shown below:

```
<odi:invokeSessionResponse xmlns:odi="xmlns.oracle.com/odi/OdiInvoke">
```



```

<odi:CommandResultType>
  <odi:Ok>true</odi:Ok>
  <odi:SessionNumber>3001</odi:SessionNumber>
</odi:CommandResultType>
</odi:invokeSessionResponse>

```

Listing ODI Execution Contexts Using a Web Service

The Oracle Data Integrator Public Web Services allow you to retrieve the list of all the ODI contexts present in a repository from a web service.

To retrieve the list of all the ODI contexts defined in the repository using a web service, you can invoke the `OdiInvoke` web service. The web service port to use is called `listContext`. This web service commands an agent to connect to a given master repository, and to retrieve all contexts with their names and codes.

A simple SOAP request for this web service is provided below. Refer to the WSDL file for the comprehensive list of parameters.

```

<listContextRequest>
  <listContextRequest>
    <!-- Connection information to the repository -->
    <!-- This example is an Oracle Repository -->
    <JdbcDriver>oracle.jdbc.driver.OracleDriver</JdbcDriver>
    <JdbcUrl>jdbc:oracle:thin:@srv011:1521:ORA10G</JdbcUrl>
    <JdbcUser>repo</JdbcUser>
    <JdbcPassword>snp65934</JdbcPassword>
    <OdiUser>SUPERVISOR</OdiUser>
    <OdiPassword>PASS</OdiPassword>
  </listContextRequest>
</listContextRequest>

```

The session execution returns a SOAP response as shown below:

```

<odi:listContextResponse xmlns:odi="xmlns.oracle.com/odi/OdiInvoke/">
  <odi:ContextList>
    <odi:ContextName>Global</odi:ContextName>
    <odi:ContextCode>GLOBAL</odi:ContextCode>
  </odi:ContextList>
</odi:listContextResponse>

```

Listing ODI Scenarios Using a Web Service

The Oracle Data Integrator Public Web Services allow you to list the set of scenarios in an ODI repository from a web service.

To retrieve the list of ODI scenarios defined in the repository using a web service, you can invoke the `OdiInvoke` web service. The web service port to use is called `listScenario`.

This web service commands an agent to connect to a given work repository, and to list all ODI scenarios present in the repository with their names and versions.

A simple SOAP request for this web service is provided below. Refer to the WSDL file for the comprehensive list of parameters.

```
<listScenarioRequest>
  <listScenarioRequest>
    <RepositoryConnection>
      <!-- Connection information to the repository -->
      <!-- This example is an Oracle Repository -->
      <JdbcDriver>oracle.jdbc.driver.OracleDriver</JdbcDriver>
      <JdbcUrl>jdbc:oracle:thin:@srv011:1521:ORA10G</JdbcUrl>
      <JdbcUser>repo</JdbcUser>
      <JdbcPassword>snp65934</JdbcPassword>
      <OdiUser>SUPERVISOR</OdiUser>
      <OdiPassword>PASS</OdiPassword>
    </RepositoryConnection>
    <WorkRepository> WORKREP</WorkRepository><!-- Work Repository
    CODE -->
  </listScenarioRequest>
</listScenarioRequest>
```

The session execution returns a SOAP response as shown below:

```
<odi:listScenarioResponse
xmlns:odi="xmlns.oracle.com/odi/OdiInvoke/">
  <odi:ScenarioList>
    <odi:ScenName>LOAD_CUSTOMER_DIMENSION</odi:ScenName>
    <odi:ScenVersion>001</odi:ScenVersion>
  </odi:ScenarioList>
  <odi:ScenarioList>
    <odi:ScenName>LOAD_DW</odi:ScenName>
    <odi:ScenVersion>001</odi:ScenVersion>
  </odi:ScenarioList>
  <odi:ScenarioList>
    <odi:ScenName>LOAD_DATAWAREHOUSE</odi:ScenName>
    <odi:ScenVersion>001</odi:ScenVersion>
  </odi:ScenarioList>
  <odi:ScenarioList>
    <odi:ScenName>LOAD_DATAWAREHOUSE</odi:ScenName>
    <odi:ScenVersion>002</odi:ScenVersion>
  </odi:ScenarioList>
</odi:listScenarioResponse>
```

Executing a Scenario from an HTTP URL

With the **Metadata Navigator** module, it is possible to launch a scenario from a web page or an HTTP URL.

Note: To execute a scenario this way, you must first Install Metadata Navigator.

Principles

Metadata Navigator provides a **StartScen servlet** that responds to **HTTP POST** requests. This servlet provides the same features as the OdiStartScen tool.

- The StartScen servlet is called by using an HTTP POST request on the `/snpsrepxp/startscen.do` resource of your Metadata Navigator Server. Appropriate HTTP parameters should be passed with the request.
- The servlet returns an HTTP response in XML, HTML or plain text format.

Parameters for the StartScen servlet

Parameter	Description
<code>agent_name</code>	Network name or IP address of the machine running the agent.
<code>agent_port</code>	Port the agent is listening on.
<code>master_driver</code>	JDBC driver used to access the master repository. For example: <code>oracle.jdbc.driver.OracleDriver</code>
<code>master_url</code>	JDBC URL used to access the master repository. For example: <code>jdbc:oracle:thin:@168.67.0.100:1522:ORCL</code>
<code>master_user</code>	Database user for the master repository connection.
<code>master_psw</code>	Password for the database user. This password must be encrypted using this command: <code>agent ENCODE <password></code>
<code>work_repository</code>	Name of the work repository where the scenario to be executed is stored.
<code>snps_user</code>	Name of the Oracle Data Integrator user whose rights will be used to execute the scenario. If this user has no rights on the context, the scenario will not be able to be executed.
<code>snps_psw</code>	Password for the Oracle Data Integrator user name. This password must be encrypted using this command: <code>agent ENCODE <password></code>
<code>scen_name</code>	Name of the scenario.
<code>scen_version</code>	Version number of the scenario. If the version specified is <code>-1</code> , the most recent version of the scenario is executed.
<code>context_code</code>	Execution context to launch the scenario in.
<code>log_level</code>	Defines the logging level for this scenario's execution. All tasks whose logging level is lower than or equal to this value will be retained in the log at the end of execution. However, if the scenario terminates abnormally, all tasks will be kept, regardless of this setting. This parameter is an integer from 0 to 5.
<code>http_reply</code>	Type of HTTP response. This response indicates if the session has been started or not. Possible response types are XML, HTML and text. The default is XML.

Valid values for this parameter are XML | HTML | TXT.

In addition to the response in XML, HTML or text format, all HTTP responses from the StartScen servlet are returned as cookies in the HTTP header.

Scenario parameter values

For scenarios with variable parameters, you must provide the parameter values as HTTP parameters.

For instance, if you have a variable named `VAR1`, you should define an HTTP request parameter `VAR1=<VAR1_VALUE>`.

Servlet response

The servlet response indicates if the session has been started correctly.

Warning: Notification that the session started correctly does not indicate whether the session itself has succeeded.

The response contains 3 elements:

Element	Description
<code>snps_exe_ok</code>	Boolean (<code>true</code> <code>false</code>) indicating whether the session was started or not.
<code>snps_session_no</code>	A unique id number for the session in the repository.
<code>snps_error_msg</code>	Error message, if the scenario failed to start correctly (<code>snps_exe_ok = false</code>)

Sample responses

HTTP header cookies

3 cookies appear in the header : `snps_exe_ok`, `snps_session_no` and `snps_error_msg`.

XML

```
<snps_scen_result>
  <snps_exe_ok>true|false</snps_exec_ok>
  <snps_session_no>session number</snps_session_no>
  <snps_error_msg>error message</snps_error_msg>
</snps_scen_result>
```

Text

```
snps_exe_ok=true|false
snps_session_no=session number
snps_error_msg=error message
```

Samples

Starting a scenario with an HTML form

In this example, a form is used to make a call to the StartScen servlet. It should be installed in the Metadata Navigator installation directory. The HTML page source is available in the /demo/http/ directory of your Oracle Data Integrator installation.

```
1 <html>
2   <title>HTTP StartScen Example</title>
3   <body>
4     <h1>Enter appropriate values for your Oracle Data Integrator
scenario</h1>
5     <form method="post" name="form1" action="./startscen.do">
6       <table>
7         <tr>
8           <td><b>HTTP Parameter Name</b></td>
9           <td><b>HTTP Parameter Value</b></td>
10        </tr>
11        <tr>
12          <td><b>agent_name</b></td>
13          <td><input type="text" name="agent_name"
value="" /></td>
14        </tr>
15        <tr>
16          <td><b>agent_port</b></td>
17          <td><input type="text" name="agent_port"
value="" /></td>
18        </tr>
19        <tr>
20          <td><b>master_driver</b></td>
21          <td><input type="text" name="master_driver"
value="" /></td>
22        </tr>
23        <tr>
24          <td><b>master_url</b></td>
25          <td><input type="text" name="master_url"
value="" /></td>
26        </tr>
27        <tr>
28          <td><b>master_user</b></td>
29          <td><input type="text" name="master_user"
value="" /></td>
30        </tr>
31        <tr>
32          <td><b>master_psw</b></td>
33          <td><input type="text" name="master_psw"
value="" /></td>
```

```
34         </tr>
35         <tr>
36             <td><b>work_repository</b></td>
37             <td><input type="text" name="work_repository"
value="" /></td>
38         </tr>
39         <tr>
40             <td><b>snps_user</b></td>
41             <td><input type="text" name="snps_user"
value="" /></td>
42         </tr>
43         <tr>
44             <td><b>snps_psw</b></td>
45             <td><input type="text" name="snps_psw" value="" /></td>
46         </tr>
47         <tr>
48             <td><b>scen_name</b></td>
49             <td><input type="text" name="scen_name"
value="" /></td>
50         </tr>
51         <tr>
52             <td><b>scen_version</b></td>
53             <td><input type="text" name="scen_version"
value="" /></td>
54         </tr>
55         <tr>
56             <td><b>context_code</b></td>
57             <td><input type="text" name="context_code"
value="" /></td>
58         </tr>
59         <tr>
60             <td><b>log_level</b></td>
61             <td><input type="text" name="log_level"
value="" /></td>
62         </tr>
63         <tr>
64             <td><b>http_reply</b> (XML|HTML|TXT) </td>
65             <td><input type="text" name="http_reply"
value="" /></td>
66         </tr>
67         <tr>
68             <td>Add all other necessary variables for your
scenario here</td>
69             <td/>
70             <td><b>NAME_OF_YOUR_VARIABLE</b></td>
```

```

71         <td><input type="text"
name="NAME_OF_YOUR_VARIABLE" /></td>
72     </tr>
73 </table>
74     <p><input type="submit" /></p>
75 </form>
76 </body>
77 </html>

```

Starting a scenario from an HTTP URL

The following example demonstrates how to start a scenario from a simple URL. It calls the servlet on the remote server where Metadata Navigator is installed. The HTML page source is available in the `/demo/http/` directory of your Oracle Data Integrator installation. Note that the parameters (indicated here in green) will not be valid in your configuration, and should be modified.

```

1 <html>
2 <head>
3     <title>Script Example of HTTP StartScen</title>
4     <script type="text/javascript">
5         <!--
6         function submitForm(form)
7         {
8             alert("The scenario is being launched");
9             w = window.open(form.action, "resultwindow",
",height=360,width=360,scrollbars=yes, resizable=yes, toolbar=no,
status=yes, menubar=no");
10            w.focus();
11            form.submit();
12        }
13        -->
14    </script>
15 </head>
16 <body>
17     <form method="post" name="form1"
action="http://mars:8080/snpsrepexp/startscen.do" target="resultwindow">
18
19     <p> Click <a
href="JavaScript:submitForm(document.form1)">HERE</a> to start the
scenario </p>
20     <input type="hidden" name="agent_name" value="MARS" />
21     <input type="hidden" name="agent_port" value="20910" />
22     <input type="hidden" name="master_driver"
value="oracle.jdbc.driver.OracleDriver" />
23     <input type="hidden" name="master_url"
value="jdbc:oracle:thin:@mars:1521:mars" />
24     <input type="hidden" name="master_user" value="snpm32" />

```

```
25     <input type="hidden" name="master_psw"
value="NENDKGNEJMKCHBHDHEHJDBGBGFDGGH" />
26     <input type="hidden" name="work_repository"
value="WorkRep3D_DEMO_SGS" />
27     <input type="hidden" name="snps_user" value="SUPERVISOR" />
28     <input type="hidden" name="snps_psw"
value="LELKI E LGLJMDLKMGEHJDBGBGFDGGH" />
29     <input type="hidden" name="scen_name" value="LOAD_SALES" />
30     <input type="hidden" name="scen_version" value="5" />
31     <input type="hidden" name="context_code" value="GLOBAL" />
32     <input type="hidden" name="log_level" value="5" />
33     <input type="hidden" name="http_reply" value="HTML" />
34     <!--- List of variables -->
35     <input type="hidden" name="PROJECT.VAR1" value="VAR1_VALUE" />
36     <input type="hidden" name="PROJECT.VAR2" value="VAR2_VALUE" />
37 </form>
38 </body>
39 </html>
```

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:

In cases where some procedures or interfaces 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 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 the Designer module in the target environment (connection to the target repository)
4. Import the scenario from the export file

Scheduling a Scenario with an External Scheduler

To integrate the launching of a scenario with an external scheduler, simply integrate the launching using the OS command in the scheduler.

If the scenario ends correctly, the return code will be 0. If not, the return code will be different than 0. This code is accessible as an operating system environment variable provided for that purpose.

As an example, this is how you would schedule version 3 of a scenario named DW in a GLOBAL context from a Windows server named mercure on the first day of every month at midnight using the Windows AT scheduler:

```
AT \mercure 00 :00 /every 1 "c:\program files\odi\bin\startscen" DW 3  
GLOBAL
```

Restarting a Session

Any session that has encountered an error, or been stopped by the operator can be restarted. It can be restarted from Operator, or through a command line.

Note: 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).

To restart a session from Operator:

1. Open **Operator**.
2. Select the session to restart from the tree view.
3. Right-click then select **Restart**.
4. Click **OK** in the window that appears.

When Oracle Data Integrator has restarted the session, the message **Session started** is displayed.

To restart a session from an OS command:

1. Open a shell or a Windows command prompt.
2. Change to the `/bin` directory under the Oracle Data Integrator installation directory.
3. Type in the following command:
`restartsession.bat <session number> [-V=<verbose level>]` for Windows

or

```
./restartsession.sh <session number> [-V=<verbose level>] for UNIX
```

Oracle Data Integrator restarts the session. You can monitor the execution of the session in Operator.

To restart a Session Using a Web Service

The Oracle Data Integrator Public Web Services allow you to restart a session from a web service.

For more information, see [Restart a Session Using a Web Service](#).

See also:

- [Launching a scenario from an OS command](#)
- [Executing a scenario from Designer](#)

Other Operations

Connecting to a Work Repository

To connect to a work repository and launch the Designer module:

1. In the **Start Menu**, select **Programs > Oracle Data Integrator > Designer**, or launch the **Designer** script (`bin/designer.bat` or `bin/designer.sh`).
2. Click on the button **New** (first button to the right of the field **Login name**).
3. Complete the fields:
 - Oracle Data Integrator Connection:**
 - **Login name:** A generic alias (for example: Repository)
 - **User:** SUPERVISOR (in capitals)
 - **Password:** SUNOPSIS (in capitals)
 - DBMS connection (Master Repository):**
 - **User:** User id/login of the owner of the tables you have created [for the master repository](#) (not the work repository).
 - **Password:** This user's password.
 - **List of drivers:** choose the driver required to connect to the DBMS hosting **the master repository** you have just created.
 - **URL:** The complete path for the data server hosting the **master repository**. For more information, refer to the section [JDBC URL Sample](#).
 - Work Repository:**
 - **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.
4. Click on **Test** to check that the connection is working.
5. Click **OK**. The module **Designer** opens.

Important Note: The default password of the SUPERVISOR account is SUNOPSIS. For security reasons, you should change this password as soon as you can.

Changing a User's Password

Administrators can change users' passwords through the user's properties in **Security Manager**. A user may also change their own password in any graphical module.

To change a user's password:

1. Log in as the user whose password you want to change.
2. From the **File** menu, select **Change Password**.

3. Type the current password and then the new password twice.
4. Click **OK** to save.

The user password is changed.

Warning: If the password is stored in the connection properties, then you must update the connection information the next time you login through a graphical module . You must also update any parameter files that contain this user's login/password information.

Note: A password should always meet the Password Policy defined by the administrator. If it does not, an error occurs when you try to save your changes.

Setting User Parameters

Oracle Data Integrator saves user parameters such as default directories, windows positions, etc.

User parameters are saved in the `userpref.xml` file in `/bin`.

To set user parameters:

1. Select **User Parameters** from the **File** menu.
2. In the **Editing User Parameters** window, change the value of parameters as required.
3. Click **OK** to save and close the window.

A list of the possible user parameters is available in the Reference Manual.

Designer: Generating a Report

To generate a report in Designer:

1. Select the object for which you wish to generate a report.
2. Right-click, select **Print**, then select the type of report you wish to generate.
3. 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. Click **OK**.

A report is written to the filename specified in step 2, in **Adobe™ PDF** format.


If the **Open the file after the generation?** option is selected, Oracle Data Integrator will open **Acrobat® Reader™** to view the generated report.

Note: In order to view the generated report, you must specify the location of **Acrobat® Reader™** in the user parameters.

The following types of reports can be generated for each type of object.

Object	Reports
Project	Knowledge modules
Folder	Folder, Packages, Interfaces, Procedures
Model	Model
Sub-model	Sub-model

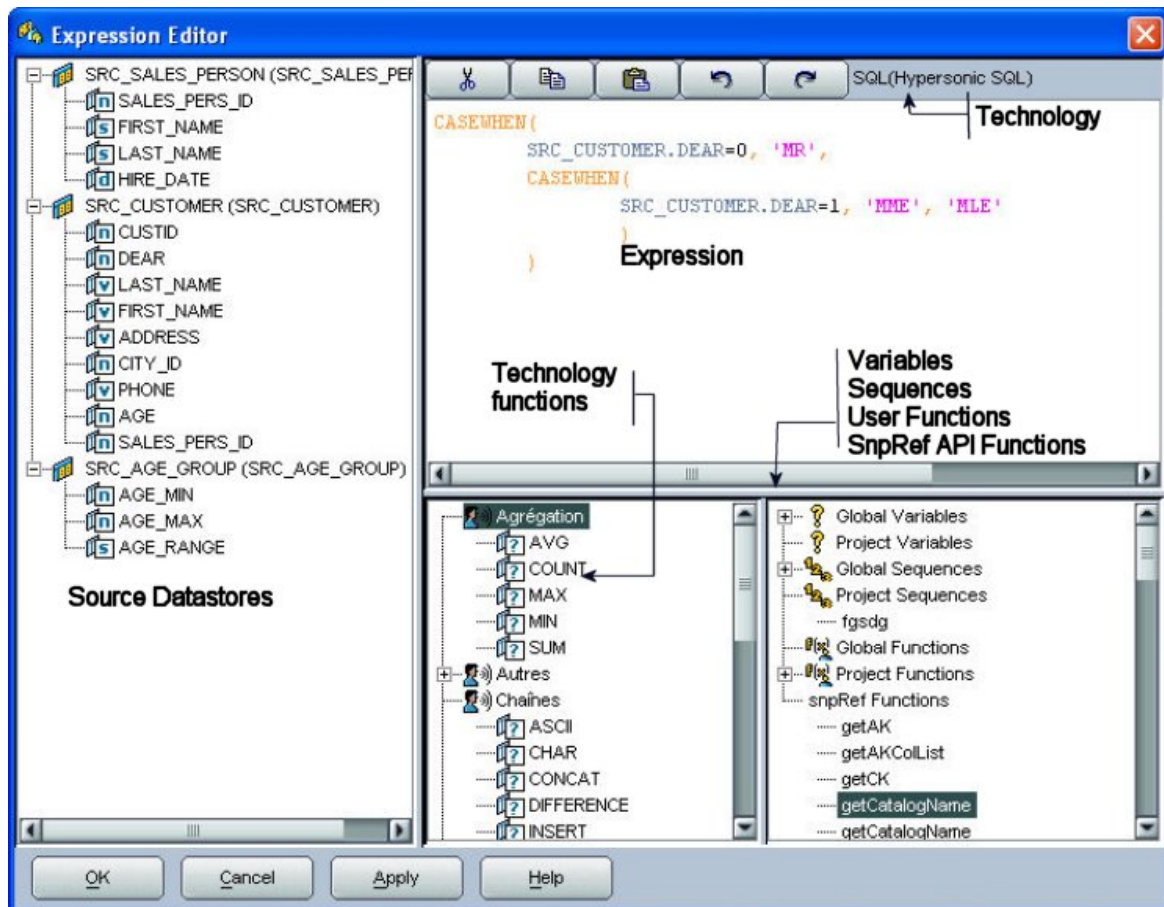
Using the Expression Editor

The Expression Editor is launched by clicking on the button . It provides help in writing expressions for mappings, filters, joins or other queries. By using the Expression Editor, you can avoid most common syntax errors. It also contains a valuable reference for functions and other objects.

The Expression Editor is shown below. It is comprised of the following parts:

- **Source datastores:** When editing an expression for an interface, contains the names of source datastores and columns.
- **Expression:** The current text of the expression. You can directly type code here, or drag and drop elements from the other panels.
- **Technology functions:** The list of language elements and functions appropriate for the given **Technology**.
- **Variable, Sequences, User Functions and odiRef API:** 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) are available using the toolbar buttons.



Searching for an object

Oracle Data Integrator' search feature helps you quickly find objects in the Designer, Operator or Topology Manager graphical modules.




Each search is performed within a given scope. A **Search Scope** is a pre-defined set of object types corresponding to a functional domain of Oracle Data Integrator.

For example, the *Meta-data* scope searches all metadata-related objects (Datastores, models, references, etc), while *Procedure* scope searches procedures, procedure commands and options.

You can search on:

- all scopes and all types of Oracle Data Integrator objects;
- all object types in a given scope; or
- a specific object type.

To carry out a search:

1. Click the  **Search View** button in the module toolbar to open the search panel.
2. Select the **Search Scope**, or select *<All Scopes>* for a list of all Oracle Data Integrator objects.
3. Select the **Object Type** (optionally filtered by the scope), or select *<All Object Types>* to search all types of objects.
4. Specify the search criteria:
 - **Name:** This field is always available. It is the name of the object you wish to find.
 - **Favorite Criteria:** Fields corresponding to the most useful criteria for the given type of object. If no **Object Type** is selected, this panel is unavailable.
 - **Date and User:** Filters the search by the user, date of creation, or most recent update of the object.
5. Select the **Advanced** tab for more search criteria for the selected object type. If no **Object Type** is selected, this tab only contains the **Name** field.
6. Check the **Match Case** box if all text matches should be case sensitive.
7. Click the **Search** button.
The search starts. A progress bar is displayed.
8. When searching is complete, the results are displayed in a **Hierarchical View** or in a **Linear view**. Switch between views by using the tabs at the bottom.
You can edit, delete or view the objects by right-clicking on them.
9. You can also save the current search criteria in a file by clicking the  **Save Search Criteria** button. To use these saved criteria later, use the  **Load Search Criteria** button.

Search Tips

- Specify a scope and sufficient criteria to avoid retrieving too many objects. If too many objects match the given type and criteria, then the search may become slow.
- For text fields (such as the **Name**), the value entered as the criteria is matched anywhere within the object's text field. For example, if you search for the word "Data" in the **Name** field, then objects called "Project Data Warehouse" or "Data Mart Structure", etc. will be returned. For other field types, such as numbers or dates, only exact matches will be returned.

Import/Export

Importing a Work Repository

Importing or exporting a work repository allows you to transfer all models and projects from one repository to another.

To import a work repository:

1. In Designer, click on **File > Import > Work Repository...**
2. Select an **Import Mode**, the **Folder** or **Zip File** to import from
3. Click **OK**.

The specified file(s) are imported into the work repository.

See also:

Exporting a Work Repository

Import Modes

Exporting the Master Repository

Importing the Master Repository

Exporting the Topology

Importing the Topology

Exporting a Work Repository

Importing or exporting a work repository allows you to transfer all models and projects from one repository to another.

To export a work repository:

1. In Designer, click on **File > Export > Work Repository...**
2. Specify the following parameters:
 - **Export to Directory:** Directory to create the export file(s) in.
 - **Make a Compressed Export:** Exports all objects to a single compressed file, rather than a separate files.
 - **Zip File name:** Name of the compressed file, if using the option **Make a Compressed Export**.
 - **Character Set:** Encoding to use in the export file. This encoding is specified in the XML file header: `<?xml version="1.0" encoding="ISO-8859-1"?>`
 - **Java Character Set:** Java character set used to generate the file.
3. Click **OK**.

The export file(s) is/are created in the specified Export Directory.

See also:

Importing a Work Repository

Exporting the Master Repository

Importing the Master Repository

Importing the Topology

Exporting the Topology

Importing an Object

Importing and exporting allows you to transfer objects (Interfaces, Knowledge Modules, Models, ...) from one repository to another.

To import an object in Oracle Data Integrator:

1. Select the appropriate object for import in the tree. If such an object does not exist, then create it.
2. Right-click on the object, and select **import**, then the type of the object you wish to import.
3. Specify the **Export Directory**, and select one or more files in **Import Name**. Select the **Import Mode**, then click on **OK**.

The XML-formatted files are imported into the work repository, and become visible in Oracle Data Integrator.

See also:

Exporting an Object

Import Modes

Exporting an Object

Exporting and importing allows you to transfer Oracle Data Integrator Objects 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 tree view.
2. Right-click on the object, and select **Export....** If this menu item does not appear, then this type of object does not have the export feature.
3. Set the Export parameters, and click **OK**.
You must at least specify the **Export Name**. If the **Export Directory** is not specified, then the export file is created in the **Default Export Directory**.

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

See also:

Importing an Object

Object export parameters

Exporting Multiple Objects

Exporting Multiple Objects

You can export one or more objects at once, using the **Export Multiple Objects** menu item. This lets you export them to a zip file or a directory, and lets you re-use an existing list of objects to export.

A more powerful mechanism for doing this is using Solutions. See Using Version Management.

How to export multiple objects at once:

1. Select **Export Multiple Objects** from the menu of Designer, Topology Manager, Security Manager or Operator.
2. Specify the options of the export:
 - **Export to directory:** the directory to export to. This must be an existing directory.

- **Export as zip file:** Specify whether to produce a zip file or not. The objects are either exported as .xml files directly into the directory, or as a zip file containing .xml files.
 - **Zip file name:** If producing a zip file, you must name it.
 - **Export child objects:** Choose whether to export child objects (steps of packages, folders of projects etc) or not. Usually you should check this box.
 - **Replace existing files without warning:** If unchecked, a warning message is displayed before overwriting any files in the export process.
 - **XML Version:** The version of XML that will be written to the headers of the exported files.
 - **Character set:** The name of the character set that will be written to the XML headers.
 - **Java character set:** The name of the Java character set that will be used during the encoding process. For non-European character sets, you must change this.
3. Specify the list of objects to export
- Drag and drop objects from Oracle Data Integrator' graphical modules. You can mix objects from Topology Manager with objects from Designer, for instance.
 - Click **Load a list of objects...** to load a previously saved list of objects. This is useful if regularly exporting the same list of objects.
 - To save a list of objects, check **Save this list in:** and specify a file name. If the file exists already, it will be overwritten without warning.

To import multiple objects at once, you must use a Solution.

See also:

Exporting an Object

Introduction to Operator

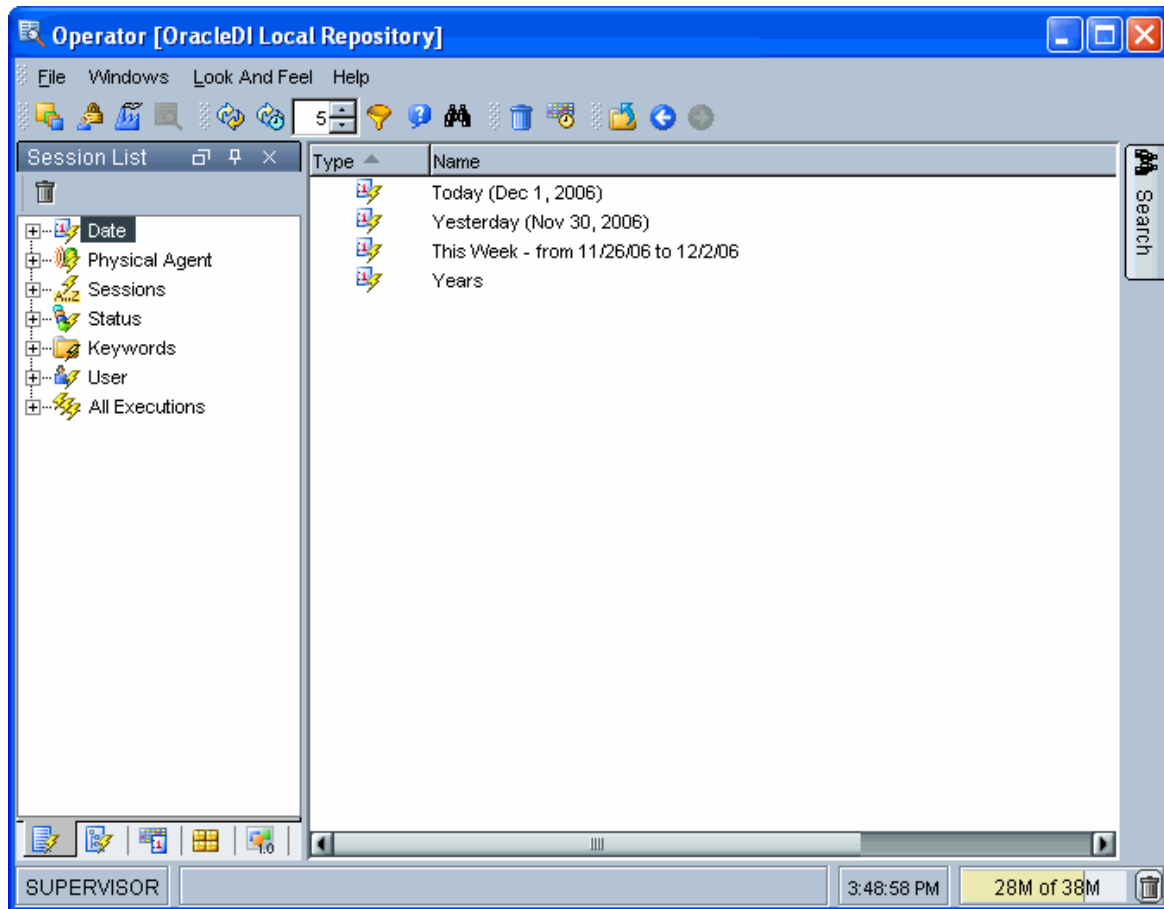
Through the **Operator** module, you can manage your interface executions in the **sessions**, as well as the **scenarios** in production.

The Operator module stores this information in a work repository, while using the topology defined in the master repository.

- Working with Operator

Operator's Interface

The Operator GUI appears as follow:



The Menu

The **Menu** contains pull-down menus to access the following features:

- Import
- Log purge

- Scheduling
- Display options
- Open modules or tree views
- Change the user's password and options

The Toolbar

The **Toolbar** lets you:

- Open other modules
- Browse the log
- Purge the log
- Display the scheduling information
- Refresh the log manually or automatically
- Open the on-online help

The Tree Views

Operator objects available to the current user are organized into the following tree views:

- **Session List** displays all sessions organized per date, physical agent, state, keywords, etc
- **Hierarchical Sessions** displays the execution sessions also organized in a hierarchy with their child sessions
- **Scheduling** displays the list of physical agents and schedules
- **Scenarios** displays the list of scenarios available

Each tree view appears in a floatable frames that may be docked to the sides of the main window. These frames can be also be stacked up. When several frames are stacked up, tabs appear at the bottom of the frame window to access each frame of the stack.

Tree view frames can be moved, docked and stacked by selecting and dragging the frame title or tab. To lock the position of views, select **Lock window layout** from the **Windows** menu.

If a tree view frame does not appear in the main window or has been closed, it can be opened using the **Windows > Show View** menu.

It is possible in each tree view to perform the following operations:

- Expand and collapse nodes by clicking on them
- Activate the methods associated to the objects (Edit, Delete, ...) through the popup menus
- Edit objects by double-clicking on them, or by drag and dropping them on the **Workbench**.

The Workbench

The Workbench displays the list of sub-objects for the object currently selected in the tree view. For example, if a step is selected in the tree view, the list of tasks for this step will be displayed in the workbench. The columns appearing in each list can be customized through the popup menu of the column titles.

The windows for object being edited or displayed appear in the **Workbench**.

Working with Operator

Sessions

Session, step, tasks

A **session** is an execution (of a scenario, an interface, a package or a procedure, ...) 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 an interface or a single variable, for example, a session has only one session step.

The **task** is the smallest execution unit. It corresponds to a procedure command in a KM, a procedure, assignment of a variable, etc

Status

A Session, step or task always has a status.


There are six possible status values:

- **Done** : The session, step or task was executed successfully.
- **Error** : The session, step or task has terminated due to an error.
- **Running** : The session, step or task is being executed.
- **Waiting** : The session, step or task is waiting to be executed.
- **Warning** (Tasks only) : 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 terminated, a **session** takes the status of the last executed step (**Done** or **Error**). When terminated, 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**)

Handling Errors

To analyze an error:

1. In the Operator tree view, identify the session, the step and the task in error.
2. Double click the Error marker  on the task. The task window opens:
 - In the **execution** tab, the **return code** and **message** give the error that stopped the session.
 - In the **definition** tab, the failed **order** is displayed.

Managing the Log

Oracle Data Integrator provides several solutions for managing your log data:

- Filtering Sessions in Operator to display only certain execution sessions in Operator
- Purging the Log to remove the information of past sessions

- Importing the Log and Exporting the Log for archiving purposes

Scenarios

To run a scenario from Operator:

1. Select the required scenario in the **Scenarios** view of Operator.
2. Right click and select **Execute**.
3. Select the **Context** and the required **Agent** (for execution by a client station, choose the option Local (no agent)).
4. Click **OK**.

Oracle Data Integrator has created and launched an execution session when the message Session started is displayed. You can monitor the execution of the session in the log.

Before running a scenario, you need to have the scenario generated from Designer or imported from a file.


To delete a scenario:

1. Right click the scenario and select **Delete**.
2. Click **OK** to confirm the deletion.
3. Specify whether to delete linked sessions or not. A linked session is one that was created by executing the scenario.
 - To avoid this question in the future, check the **Remember this choice** checkbox. To change this setting later, modify the **Delete sessions with scenarios** user parameter.
4. Click **OK**.


Filtering Sessions in Operator

Filtering log sessions allows you to display only certain execution sessions in **Operator**, 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 certain sessions:

1. Open **Operator**.
2. Click on the  **Filter** button on the toolbar.
3. Select the appropriate values for the following parameters. The settings by default select all sessions.
 - **Session Number** - use blank to show all sessions.
 - **Session Name** - use % as a wildcard. Hence "DWH%" would match 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
4. Click **Apply** for a preview of the current filter.

5. Click **Ok**.

Sessions that do not match these values are masked in the session list. The  **Filter** button on the toolbar appears to be pushed in.

To deactivate the filter:

1. If the  **Filter** button appears to be pushed in on the toolbar, click on this button.

The current filter is deactivated, and all sessions appear in the list.

Displaying Scheduling Information

The Scheduling Information allows you to visualize the agents' scheduled tasks.

Important: 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.

To Display the Scheduling information from Operator:

1. Open **Operator**
2. Click on the **Scheduling Information** button in the toolbar

The **Scheduling Information** window for all agents appears.

To Display the Scheduling information from Topology Manager:

1. Open **Topology Manager**
2. In **Physical Architecture** view, select the **Physical Agent** you wish to display the planning.
3. Right click and select **Scheduling Information**

The **Scheduling Information** window for this agent appears.

Purging the Log

Purging the log allows you to remove the information of past sessions. This procedure cleans the Work Repository from these past sessions. It is advised to perform a purge regularly. This purge can be automated using the OdiPurgeLog tool.

To purge the Log:

1. Open **Operator**
2. Click on the **Purge Log** button in the toolbar, or select **File > Purge Log** in the menu bar.
3. Set the date and time range for the sessions to delete, and optionally specify **context**, **agent** and **user** filters. You can also specify a **session name** mask, using % as a wildcard. Only the sessions in the time range and matching the three filters and the session mask will be removed.
If you check the **purge scenario reports** box, scenario reports (appearing under the **execution** node of each scenario) will also be purged.
4. Click **OK**.

Oracle Data Integrator removes the sessions from the log.

Note: It is also possible to delete sessions by selecting one or more sessions in Operator and pressing the **DEL** key. Be cautious when using this method. It might lead to performance

problems when deleting a lot of sessions. In that case, the **Purge Log** button in the toolbar should be used.

Restarting a Session

Any session that has encountered an error, or been stopped by the operator can be restarted. It can be restarted from Operator, or through a command line.

Note: 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).

To restart a session from Operator:

1. Open **Operator**.
2. Select the session to restart from the tree view.
3. Right-click then select **Restart**.
4. Click **OK** in the window that appears.

When Oracle Data Integrator has restarted the session, the message **Session started** is displayed.

To restart a session from an OS command:

1. Open a shell or a Windows command prompt.
2. Change to the `/bin` directory under the Oracle Data Integrator installation directory.
3. Type in the following command:
`restartsession.bat <session number> [-V=<verbose level>]` for Windows
or
`./restartsession.sh <session number> [-V=<verbose level>]` for UNIX

Oracle Data Integrator restarts the session. You can monitor the execution of the session in Operator.

To restart a Session Using a Web Service

The Oracle Data Integrator Public Web Services allow you to restart a session from a web service. For more information, see [Restart a Session Using a Web Service](#).

See also:

- [Launching a scenario from an OS command](#)
- [Executing a scenario from Designer](#)

Importing Scenarios in Production

A scenario generated from Designer can be imported in a development or execution repository through the Operator Module, to run in production.

Similarly, a solution containing several scenarios can be imported to easily transfer a group of scenarios at once. See [Using Version Management](#) for more information.

To import a scenario in production:

1. Open **Operator**.
2. Select **File > Import Scenario...** in the menu bar, or right-click in the **Scenarios** view and select **Import Scenario...**
3. Specify the **Import Directory**, and select one or more scenarios to import.
4. Select the Import Mode.
5. Check the **Import schedules** option if you want to import the schedules exported with the scenario as well.
6. Click **OK**.

The imported scenarios appear on the **scenarios** node. If the scenario was in a scenario folder at the time it was exported, it will be re-imported into the same scenario folder.

To import a solution:


1. Go to the **Solutions** view.
2. Right-click the tree view and select **Import Solution...**
3. Specify the **Import Directory**, and select one or more solutions to import.
4. Select the Import Mode.
5. Click **OK**.
6. Double click the solution to open it.
7. Select one or more scenarios that you wish to restore, from either the **Principal Elements** or **Required Elements** views.
8. Click the **Restore** button.

Note: Only scenarios may be restored from within Operator.

Scenario Folders

In Operator, scenarios can be grouped into scenario folders to facilitate organization. Scenario folders can contain other scenario folders.

To create a scenario folder:

1. Go to the **Scenarios** view
2. Right-click anywhere on the tree view and select **Insert scenario folder...** or click the  **Add Scenario Folder** button.
3. Drag and drop scenarios to move them into the scenario folder.

To delete a scenario folder:

1. Right click the scenario folder and select **Delete**.
2. All scenarios and other scenario folders contained within it are deleted as well.

Using Session Folders

Session folders automatically group sessions that were launched with certain keywords. They are created under the **Keywords** node on the **Session List** view. Each session folder has one or

more keywords associated with it. Then, any session launched with all the matching keywords is automatically categorized beneath it.

To create a new session folder:

1. In Operator, go to the **Session List** view.
2. Right click the **Keywords** node and select **Insert Session Folder**.
3. Specify a **Folder Name**.
4. Specify one or more keywords. After entering each keyword, click the **Add** button to add it to the list.

Note: Only sessions with all the keywords of a given session folder will be shown below that session folder. Keyword matching is case sensitive.

The following examples demonstrate how session folder keywords are matched.

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 scenario with keywords:

1. Use the SnpsStartScen tool with the `-KEYWORDS` parameter. Keywords should be separated by commas.

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 recategorized.

Importing and Exporting the Log

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. Open **Operator**
2. Select **File > Import > Import the log...** in the menu bar.
3. In the **Import of the Journal window** select if you want to import the journal from a **folder** or from a **ZIP file**.
4. Specify the **folder** or **ZIP file**.
5. Click **OK**.

The specified **folder** or **ZIP file** is imported into the work repository.

Warning: It is forbidden to import a journal into a repository having the same ID.

Exporting Log Data

Exporting log data allows you to export log files for archiving purposes.

To export the log:

1. Open **Operator**
2. Select **File > Export the log...** in the menu bar.
3. In the **Import of the Log window** set the export parameters described below, and click **OK**.
4. The log data is exported in the specified location.

Log Export Parameters

The following options are available for the log export in Operator:

Properties	Description
Export to directory	Directory in which the export file will be created.
Exports 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.
From / To	Date of execution: specify either a date From or a date To , or both.
Agent	Agent used to execute the session.
Context	Session's execution Context
Session State	The possible states are Done, Error, Queued, Running, Waiting and Warning.
User	User who launched the session
Session Name	Use % as a wildcard. Hence "DWH%" would match 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 <code>encoding</code> in the XML file header. <code><?xml version="1.0" encoding="ISO-8859-1"?></code>
Java Character Set	Java character set used to generate the file.

Note: You can also export the log data using the OdiExportLog tool.

Topology

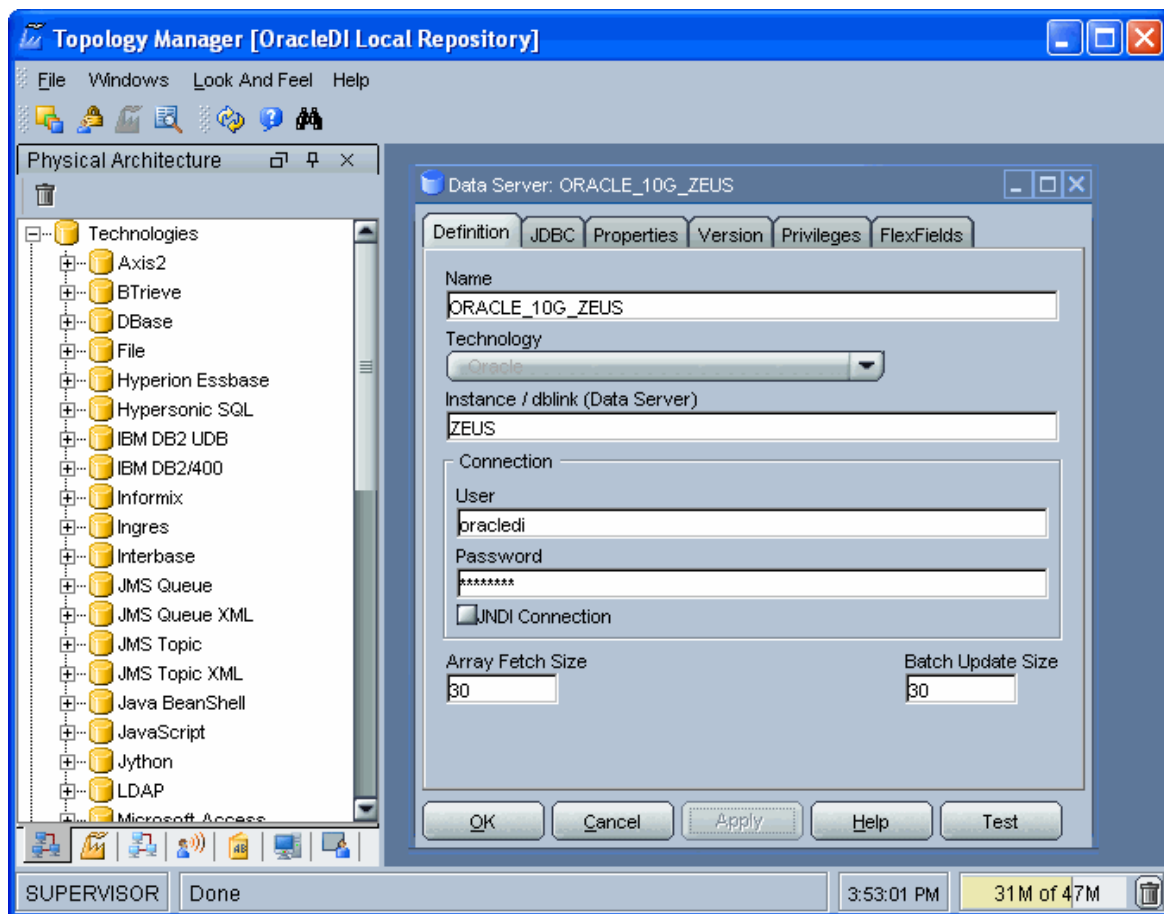
Introduction to Topology Manager

Using the **Topology Manager** module, 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 **languages** and the **agents**. In addition, Topology allows you to manage the **repositories**.

The Topology module stores this information in a master repository. This information can be used by all the other modules.

Topology Manager's Interface

The Topology Manager GUI appears as follow:



The Menu

The **Menu** contains pull-down menus to access the following features:

- Import/Export

- Wizards
- Display options
- Open modules or tree views
- Change the user's password and options

The Toolbar

The **Toolbar** lets you:

- Open other modules
- Refresh the Tree views
- Open the on-online help

The Tree Views

Topology Manager objects available to the current user are organized into the following tree views:

- The **physical architecture**, containing the **technologies** with their associated **data servers** and **physical schemas**, and the **physical agents**,
- The **logical architecture**, containing the **technologies** with their associated **logical schemas**, and the **logical agents**,
- The **contexts** linking logical and physical architectures,
- The **languages**, describing the different type of languages available.
- The **repositories**, including the actual master repository and the attached work repositories.

Each tree view appears in a floatable frames that may be docked to the sides of the main window. These frames can be also be stacked up. When several frames are stacked up, tabs appear at the bottom of the frame window to access each frame of the stack.

Tree view frames can be moved, docked and stacked by selecting and dragging the frame title or tab. To lock the position of views, select **Lock window layout** from the **Windows** menu.

If a tree view frame does not appear in the main window or has been closed, it can be opened using the **Windows > Show View** menu.

It is possible in each tree view to perform the following operations:

- Insert or import root objects to the tree view by clicking the appropriate button in the frame title
- Expand and collapse nodes by clicking on them
- Activate the methods associated to the objects (Edit, Delete, ...) through the popup menus
- Edit objects by double-clicking on them, or by drag and dropping them on the **Workbench**.

The Workbench

The windows for object being edited or displayed appear in the **Workbench**.

What is the Topology?

Through the **Topology Manager** module, the user has at his disposal a genuine physical and logical cartography of the architecture and the components he works on through Oracle Data Integrator.

- Create the Topology

Physical Architecture

The physical architecture defines the different elements of the information system, as well as the characteristics taken into account by Oracle Data Integrator.

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

Note: Each type of database (Oracle, DB2, etc) , file format (XML, File), or application software is represented in Oracle Data Integrator by a technology.

The physical components that store and return data are defined as Data Servers. A data server that can store different information according to a business logic, can be divided into Physical Schemas. A data server is always linked to a single technology.

Note: Every database server, JMS message file, group of flat files, etc, that is used in 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 definition of the Physical Agents, the Java software components that allow Oracle Data Integrator jobs to be executed on a remote machine.

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).

Logical Architecture

The logical architecture allows a user to group physical schemas containing datastores that are structurally identical but located in separate places, into Logical Schemas, structured per technology.

According to the same philosophy, the logical architecture defines the Logical Agents, allowing a unique name to be given to all the physical agents with the same function in different contexts.

For example: The **logical schema** "Accounting" may correspond to two **physical schemas**:
- "Accounting Oracle sample", used in the "development" **context**
- "Accounting corporate", used in the "production" **context**.

The two **physical schemas** are structurally identical (they contain accounting data), but physically different. They are located on two Oracle schemas, and perhaps on two different Oracle servers (data servers).

Languages

This type of component defines the characteristics specific to each language linked to a technology and used by Oracle Data Integrator.

Repositories

This branch of the topology contains information relating to the two types of repositories: the master repository and the work repositories.

Hosts

Hosts and Usages enable to manage user access to the graphical modules.

Defining the Topology

Creating the Topology

The following steps are a guideline to create the topology:

1. Create the contexts suitable for your configuration.
2. Create the data servers corresponding to the servers used in Oracle Data Integrator.
3. For each data server, create the physical schemas.
4. Create logical schemas and associate them with physical schemas in the contexts.
5. Create the physical agents for each agent running on a machine (as a listener, or in scheduler mode).
6. Create logical agents and associate them with physical agents in the contexts.

Creating a Context

To create a context:

1. Connect to **Topology Manager**
2. Select **Topology > Contexts** in the tree.
3. Right click and select **Insert context**.
4. 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 to work on this context.
5. Click **OK**.

Creating a Data Server

A **Data Server** corresponds to a Database or JMS server instance, a scripting engine ... connected with Oracle Data Integrator in order to issue commands. Under this data server, subdivisions called **Physical Schemas** are created.

Important Note : Frequently used technologies have their data server creation methods detailed in the section Use Oracle Data Integrator with Please refer to this section before proceeding.

Pre-requisites

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

- Installation of a JDBC Driver,
- Installation of a Client Connector,
- Datasource configuration,
- ...

Refer to the documentation of the technology you are connecting to through the data server. 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.

Creation of the Data Server

To create a Data Server:

1. Connect to **Topology Manager**
2. Select **Topology > Physical Architecture > Technologies** in the tree, then select the technology you want to attach your data server to.
3. Right click and select **Insert 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.
 - **... (data server):** This is the physical name of the data server. Enter this name if your data servers can be inter-connected in a native way. This parameter is not mandatory for all technologies.
 - **User/Password:** User name and password for connecting to the data server. This parameter is not mandatory for all technologies.
5. Define the connection parameters for the data server,
 - If the data server supports **JNDI** access:
 1. Check the **JNDI Connection** box.
 2. Go to the **JNDI** tab, and fill in the following fields:
 - **JNDI authentication**
 - **JNDI User/Password:** User/password connecting to the JNDI directory
 - **JNDI Protocol:** Protocol used for the connection.
 - **JNDI Driver:** The driver allowing the JNDI connection.
 - **JNDI URL:** The URL allowing the JNDI connection.
 - **JNDI Resource:** The directory element containing the connection parameters.
 - If the data server supports **JDBC** access:
 - Un-check the **JNDI Connection** box.
 - Go to the **JDBC** tab, and fill in the following fields:
 - **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.
6. Click **Test**.
7. Click **Test** in the **Test Connection** window.
8. A window indicating a **Successful Connection** appears. Click **OK**.
9. Click **OK** to validate the creation of the data server

The creation window for the first **Physical Schema** for this data server appears.

Testing a Data Server Connection

To test a connection to a data server:

1. Open the window for the **data server** you want to test.
2. Click the **Test** button.
3. Select the **agent** that will carry out the test. "Internal" indicates that the local station will attempt to connect.
4. Click **Detail** to obtain the characteristics and capacities of the JDBC driver.
5. 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**, in which Oracle Data Integrator will look for the source and target data structures for the interfaces.
- A **Work Schema** in which Oracle Data Integrator can create and manipulate temporary work data structures associated to the sources and targets contained in the Data Schema.

Important Note: Frequently used technologies have their physical schema creation methods detailed in the Using Oracle Data Integrator with ... section. Please refer to this section before proceeding.

Note: 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.

Note: Some technologies do not support the creation of temporary structures. The work schema is useless for these technologies.


Note: The user specified in the data server to which the Physical Schema is attached must have appropriate privileges on the schemas.

Creation of the Physical Schema

To Create an Physical Schema:

Note: If you have just created a Data Server, then ignore step 1. The **Physical Schema** window should already be opened.

1. Select the **Data Server**, right click and select **Insert Physical Schema**. The **Physical Schema** window appears.
2. If the technology supports multiple schemas:
 1. 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.
 2. 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). For more information, see Physical Schema
4. Go to the **Context** tab.
5. Select a **Context** and an existing **Logical Schema** for this new **Physical Schema**, then go to step 9.
If no Logical Schema for this technology exists yet, proceed to step 7.
6. Click the  button.
7. Select an existing **Context** in the left column, the 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.

Warning ! A **Logical Schema** can be associated to only one **Physical Schema** in a given **Context**.

9. Click **OK**.

Creating a Logical Schema

To create a logical schema:

1. Connect to **Topology Manager**
2. Select **Topology > Logical Architecture > Technologies** in the tree, then select the technology you want to attach your schema to.
3. Right click and select **Insert 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. Click **OK**.

Creating a Physical Agent

The Agent is a Java service which can be placed as listener on a TCP/IP port.

This service allows:

- on demand execution of sessions (model reverses, packages, scenarios, interfaces, etc) from Oracle Data Integrator modules. For this, you must launch a listener agent.
- execution of scheduled scenarios, in addition to on demand executions. The physical agent contains an optional **scheduler** that allows scenarios to be launched automatically according to predefined schedules. For this, you must launch a scheduler agent.

A physical agent can delegate executions to other physical agents. This process enables load balancing between agent.

To Create an Physical Agent:

1. Connect to **Topology Manager**
2. Right click and select **Insert Agent**.
3. Fill in the following fields:
 - **Name:** Name of the agent used in the Java graphical interface.
 - **Host:** Network name or IP address of the machine the agent has been launched on.

- **Port:** Listening port used by the agent. By default, this port is the 20910.
 - **Maximum number of sessions** supported by this agent.
4. If you want to setup load balancing, go to the **Load balancing** tab and select a set of physical agent to which the current agent can delegate executions.
 5. If the agent is launched, click **Test**. A successful connection window should appear.
 6. Click **OK**.

Creating a Logical Agent

To create a logical agent:

1. Connect to **Topology Manager**
2. Select **Topology > Logical Architecture > Agents** in the tree.
3. Right click and select **Insert Logical Agent**.
4. Fill in the agent **name**.
5. 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.
6. Click **OK**.

Automatically Reverse-engineering Datatypes

A reverse of the datatypes avoids having to enter datatypes in the master repository by recovering this information directly on the data server level. However, this function is not accepted by all data server access drivers.

Note: This procedure should only be implemented on the topology's new technologies.

To automatically reverse the technology's datatypes:

1. Create and test a data server connection for the technology.
2. Open the **Technology** window
3. Click the **Reverse** button. The reversed **datatypes** are displayed under the technology in the tree.
4. Complete the information for converting each datatype into the other technologies from the **convert to** tab in the **Datatype .** window.

Managing Repositories

Connecting to the Master Repository

To connect to the Master repository:

1. In the **Start Menu**, select **Programs > Oracle Data Integrator > Topology Manager**, or launch the **Topology Manager** script (`bin/topology.bat` or `bin/topology.sh`)
2. Click on the button **New** (first button to the right of the field **Login name**)
3. Complete the fields:
Oracle Data Integrator Connection:

- **Login name:** A generic alias (for example: Repository:
 - **User:** SUPERVISOR (use capitals)
 - **Password:** SUNOPSIS (use capitals)
- DBMS Connection (Master Repository):**
- **User:** User id / login of the owner of the tables you have created for the master repository
 - **Password:** This user's password
 - **Drivers' List:** choose the driver required to connect to the DBMS supporting the master repository you have just created
 - **URL:** The complete path of the data server hosting the repository. For more information, refer to the section JDBC URL Sample
4. Click on **Test** to check the connection is working.
 5. Validate by **OK**, then **OK. Topology Manager** opens.

Important Note: The default password of the SUPERVISOR account is SUNOPSIS. For security reasons, you should change this password as soon as you can.

Creating the Master Repository

Creating the master repository consists of creating the tables and the automatic importing of definitions for the different technologies.

To create the master repository:

1. In the **Start Menu**, select **Programs > Oracle Data Integrator > Repository Management > Master Repository Creation**, or Launch `bin/recreate.bat` or `bin/recreate.sh`.
2. Complete the fields:
 - **Driver:** the driver used to access the technology which will host the repository. For more information, refer to the section JDBC URL Sample.
 - **URL:** The complete path for the data server to host the repository. For more information, refer to the section JDBC URL Sample.
 - **User:** The user id / login of the owner of the tables (previously created under the name **snpm**).
 - **Password:** This user's password.
 - **ID:** A specific ID for the new repository, rather than the default 0. This will affect imports and exports between repositories.
 - **Technologies:** From the list, select the technology your repository will be based on.
 - **Language:** Select the language of your master repository.
3. Validate by **OK**.

Creating the dictionary begins. You can follow the procedure on your console. To test your master repository, refer to the section Connecting to the master repository.

Creating a Work Repository

Several work repositories can be designated with several master repositories if necessary. However, a work repository can be linked with only one master repository for version management purposes.

For more information on work repositories, see the reference manual.

To launch a work repository creation:

1. Connect to your master repository through the module **Topology**. For more information, refer to the section Connecting to the master repository.
2. In the icon list **Topology** -> **Repositories** -> **Work repositories**, click with the right button, then choose **Insert work repository**. A window appears, asking you to complete the connection parameters for your work repository.
3. In the **connection** window, complete the following parameters:
 - o **Name**: Type the name for your work repository connection.
 - o **Technology**: Choose the technology of the server to host your work repository.
 - o **User**: User id / login of the owner of the tables you are going to create and host of the work repository.
 - o **Password**: This user's password.
 - o Tab **JDBC** -> **JDBC Driver**: The driver required for the connection to the DBMS to host the work repository. For more information, refer to the section JDBC URL Sample.
 - o Tab **JDBC** -> **URL JDBC**: The complete path of the data server to host the work repository. For more information, refer to the section JDBC URL Sample.
4. Click on **Test**.

Caution: do not attempt to close this window by **OK** if you haven't tested your connection properly.

5. Click on **OK** to validate the parameters for connecting with the server to host your work repository. A window appears, asking you to give a unique name and user id code number to your repository.
6. In the window **Work Repository**, complete the following parameters:
 - o **ID**: give a unique number to your repository, from 1 to 998 included.
 - o **Name**: give a unique name to your work repository (for example: WorkRep1).
 - o **Type**: Choose "Designer" in the list.
7. Validate by **OK**. The creation of your work repository begins and you can follow the different steps on the console.
8. When the work repository has been created, the Work Repository window closes. You can now access this repository through the modules **Designer** and **Operator**. For more information, refer to the section Connecting to the work repository.

Exporting the Master Repository

The Master Repository Import/Export procedure allows you to transfer the whole repository (Topology and Security domains included) from one repository to another.

To export a master repository:

1. In Topology Manager, click on **File > Export > Master Repository ...**
2. Fill in the export parameters:
 - **Export to directory**: Directory in which the export file(s) will be created.
 - **Export to zip file**: When this option is selected, a unique compressed file containing all export files will be created. Otherwise, a set of export files is created.
 - **Zip File name**: Name of the compressed file if the option **Make a Compressed Export** is selected.

- **Character Set:** Encoding specified in the export file. Parameter `encoding` in the XML file header.
- **Java Character Set:** Java character set used to generate the file.
- **Export versions:** Exports 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.
- **Export solutions:** Exports all stored solutions that are stored in the repository. You may wish to unselect this option for similar reasons.

3. Click **OK**.

The export file(s) are created in the Export Directory specified.

See also:

Importing the Master Repository

Exporting the Topology

Importing the Topology

Importing the Master Repository

The Master Repository Import/Export procedure allows you to transfer the whole repository (Topology and Security domains included) from one repository to another.

It can be performed in Topology, to import the exported objects in an existing repository, or while creating a new master repository.

To import a master repository in an existing master repository:

1. In Topology Manager, click on **File > Import > Master Repository ...**
2. Select the **Import Mode**, the import **Folder** or **Zip File** and click on **OK**.

The specified file(s) are imported into the current master repository.

To create a new master repository using an export:

1. In the **Start Menu**, select **Programs > Oracle Data Integrator > Repository Management > Master Repository Import**, or Launch `bin/mimport.bat` or `bin/mimport.sh`.
2. Fill in the following fields:
 - **Driver:** the driver used to access the technology which will host the repository. For more information, refer to the section JDBC URL Sample.
 - **URL:** The complete path for the data server to host the repository. For more information, refer to the section JDBC URL Sample.
 - **User:** The user id / login of the owner of the tables (previously created under the name **snpm**).
 - **Password:** This user's password.
3. Click the **Test** button to test your connection. The result should be successful.
4. Fill in the following fields
 - **Identifier:** Numeric identifier of the new master repository.

Warning! All master repositories should have distinct identifiers. Check that the identifier you are choosing is not the identifier of an existing repository.

- **Technologies:** From the list, select the technology your repository will be based on.
- **Language:** Select the language of your master repository.

5. If using a compressed export file, check the **Use a Zip File** box and select the file containing your master repository export. If using an uncompressed export, select the directory containing the export.
6. Click **OK**.

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

See also:

Exporting the Master Repository

Exporting the Topology

Importing the Topology

Import Modes

Changing the work repository password

To change the work repository password:

1. In the **Repositories** view of **Topology Manager**, double-click the work repository. The repository edition window opens.
2. Click the **Change password** button
3. Enter the old password and the new one.
4. Click the **OK** button.

Managing Hosts & Usages

Oracle Data Integrator gives you access to a number of module (Designer, Topology, etc.). Hosts and Usages let you manage access to these modules.

Definitions

In Oracle Data Integrator:

- A machine running Oracle Data Integrator Modules is called a **Host**.
- A given module used on this machine is called a **Usage**.

A Host may have several Usages. For example, the administrator's machine should have the "Topology" and "Security" Usages.

Usages

A Usage determines if a Host may or not use one of the modules provided in the Oracle Data Integrator.

There are three Usage types of usages:

- **Always allowed:** The host can always use the given module.
- **Denied:** The host cannot use the given module.
- **Automatic** (default behaviour): The host automatically takes access to the given module when connecting to the repository with this module.

You cannot have two usages for the same module on the same host.

Default behaviour

When a new machine connects a module to the repository, a Host identifying this machine is automatically added to the hosts list, and a Usage (Automatic) for the launched module is added in the usage list for this host.

Administrating Hosts & Usages

The administrator can manually add and remove hosts. He can also add, edit and remove usages, in order to allow or prevent the use of certain modules from certain machines.

For example the administrator can prevent the developers' machines from using the Topology module by setting the usage for their hosts to "Denied".

Important Note: Removal operations on Hosts and Usages are restricted. You need a specific **Deletion Key**, provided by the technical support, for each host or usage to remove .


Creating and Editing Hosts and Usages

To manually create a Host:

Important Note: It is not possible to edit a host's parameters after its creation. It is strongly recommended that you use automatic host creation (by connecting the repository using a module from the given host), to avoid incorrect hosts creation. Once these hosts are created with automatic usages, edit these usages.

1. Connect to **Topology Manager**.
2. Select **Topology > Hosts** in the tree.
3. Right click and select **Insert Host**.
4. Fill in the following fields in the **Definition** tab:
 - **Machine Name:** Unique name identifying the machine. It is usually the machine's network name.
 - **IP Address:** IP address of the machine on the network common to the machine and the Repository server.
5. Click **OK**.

To edit the Usages of a Host:

1. Connect to **Topology Manager**.
2. Select **Topology > Hosts** in the tree. Click on the host you want to edit.
3. Right-click and select **Edit**.
4. Go to the **Usage** tab.
5. Click the  button to add a usage.
6. Select for each usage the **Module** and **Usage type**.
7. Click **OK**.


Removing Hosts and Usages

To delete a host:

1. Connect to **Topology Manager**.
2. Select **Topology > Hosts** in the tree. Click on the host you want to edit.
3. Right-click and select **Delete**.
4. A window appears prompting you for a **Deletion Key**.
5. Click the **Create unlock file** button.
6. Select a location and give a name for the unlock file, and then click **OK**.
7. Send the unlock file (.duk file) to the technical support. You should receive a **Deletion Key**.
8. Enter the key provided in the **Deletion Key** field.
9. Click **OK**.

The host disappears from the hosts list.

To remove a usage:

1. Connect to **Topology Manager**.
2. Select **Topology > Hosts** in the tree. Click on the host you want to edit.
3. Right-click and select **Edit**.
4. Go to the **Usage** tab.
5. Select the usage, and then click the  button.
6. A window appears prompting you for a **Deletion Key**.
7. Click the **Create unlock file** button.
8. Select a location and give a name for the unlock file, and then click **OK**.
9. Send the unlock file (.duk file) to the technical support. You should receive a **Deletion Key**.
10. Enter the Key provided in the **Deletion Key** field.
11. Click **OK**.

The usage disappears from the host's usages.

Exporting Hosts

Hosts are exported when Exporting the Master Repository or Exporting the Topology.

Managing Agents

Launching a Listener Agent

The Agent is a Java TCP/IP service that can be placed as listener on a TCP/IP port.

This agent works as follows:

- When it starts, it runs as a listener on its port.
- When an execution query arrives on the agent, it executes the tasks associated with the query then returns as a listener on the port.

Note: A listener agent has no scheduled executions, and does not handle scheduled scenarios.

To launch a listener agent:

1. Open a UNIX shell, Dos, or QSH (for AS/400) session
2. Launch the appropriate command file (agent.bat on Windows, agent.sh on Unix or AS/400-QSH), with any necessary parameters.

The agent is then launched as listener.

Examples:

- On Windows: `agent "-port=20300" "-v=5"` launches the listener agent on the port 20300 with a level 5 trace on the console.
- On UNIX: `./agent.sh -name=AGENTNT` launches a listener agent and names it AGENTNT.

Agent Parameters

The table below lists the different parameters that allow the agent to be configured. The parameters are preceded by the "-" character and the possible values are preceded by the "=" character. You must take the string delimiting syntax specific to the operating system on which you enter the command into account.

Parameters	Description
<code>-port=<port></code>	Port on which the agent is listening. If this parameter is not specified, the agent runs as listener on the default port 20910.
<code>-help</code>	Displays the options and parameters that can be used, without launching the agent.
<code>-name=<agent name></code>	<p>This is the name of the physical agent used.</p> <p>Oracle Data Integrator needs this parameter to identify the agent that executes a session in the following cases:</p> <ul style="list-style-type: none"> • More than one agent is declared on the same machine (on different ports). • Your machine's IP configuration is insufficient to allow Oracle Data Integrator to identify the agent. This problem occurs frequently on AS/400. • The agent's IP address does not allow the agent to be identified (127.0.0.1 or "loopback").
<code>-v=<trace level></code>	<p>Allows the agent to generate traces (verbose mode). There are five trace levels:</p> <ol style="list-style-type: none"> 1. Displays the start and end of each session. 2. Displays level 1, and the start and end of each step. 3. Displays level 2, and each task executed. 4. Displays the SQL queries executed. 5. Complete trace, generally reserved for support. <p>As some of these traces can be voluminous, it is recommended that you redirect them to a text file using the following command:</p> <ul style="list-style-type: none"> • under Windows: <code>agent.bat "-v=5" > trace.txt</code> • under Unix: <code>agent.sh -v=5 > trace.txt</code>

See also:

- Launching a scheduler agent
- Launching a web agent
- Executing a scenario from Designer
- Executing a scenario from an OS command
- Stopping an agent

Launching a Scheduler Agent

The Agent is a Java TCP/IP service that can be used to execute scenarios via predefined schedules or on demand.

This agent works as follows:

- When it starts, it connects to the work repository to get its scheduled executions (all the scenario schedules planned for it), then runs as listener on its port..
- When an execution query arrives on this agent, it executes the tasks associated with the query.
- When a scenario must be launched in accordance to the scheduled executions in memory, the agent executes it, then returns as listener on the port.
- The agent's scheduled executions can be updated at any time by clicking on the **scheduling update** button in the **physical agent** window.

Note: A scheduler agent has all the functionality of a listener agent.

To launch a scheduler agent:

1. Open a UNIX shell, Dos, QSH (for AS/400) session
2. Launch the appropriate command file (agentscheduler.bat on NT, agentscheduler.sh on Unix or AS/400-QSH), with any necessary parameters.

The agent starts up and goes to retrieve its scheduled executions on the work repository.

Examples:

- On Windows: `agentscheduler "-port=20300" "-v=5"`
- On UNIX: `agentscheduler -name=AGENTNT`

Agent Parameters

Caution: To launch a scheduler agent, the parameters for connecting to the work repository containing the scheduling must be specified. These parameters are generally given in the odiparams file.

The table below lists the different parameters that allow the agent to be configured according to your needs. The parameters are preceded by the "-" character and the possible values are preceded by the "=" character. You must take the string delimiting syntax specific to the operating system on which you enter the command into account.

Parameters	Description
<code>-port=<port></code>	Port on which the agent is listening. If this parameter is not

	specified, the agent runs as listener on the default port 20910.
-help	Displays the options and parameters that can be used, without launching the agent.
	This is the name of the physical agent used. Oracle Data Integrator needs this parameter to identify the agent that executes a session in the following cases:
-name=<agent name>	<ul style="list-style-type: none"> • More than one agent is declared on the same machine (on different ports). • Your machine's IP configuration is insufficient to allow Oracle Data Integrator to identify the agent. This problem occurs frequently on AS/400. • The agent's IP address does not allow the agent to be identified (127.0.0.1 or "loopback").
	Allows the agent to generate traces (verbose mode). There are five trace levels:
-v=<trace level>	<ol style="list-style-type: none"> 1. Displays the start and end of each session. 2. Displays level 1, and the start and end of each step. 3. Displays level 2, and each task executed. 4. Displays the SQL queries executed. 5. Complete trace, generally reserved for support. <p>As some of these traces can be voluminous, it is recommended that you redirect them to a text file using the following command:</p> <ul style="list-style-type: none"> • in Windows: <code>agent.bat "-v=5" > trace.txt</code> • in UNIX: <code>agent.sh -v=5 > trace.txt</code>
Work repository connection parameters	These parameters, which are used to specify the connection to the work repository, are specified in the odiparams file.
-SECU_DRIVER=<driver name>	JDBC driver to access the master repository. For example: <code>oracle.jdbc.driver.OracleDriver</code>
-SECU_URL=<url>	JDBC URL to access for the master repository. For example: <code>jdbc:oracle:thin:@168.67.0.100:1522:ORCL</code>
-SECU_USER=<user>	User of the master repository connection (the database user).
-SECU_PASS=<password>	Password of the user of the master repository connection. This password must be encrypted using the command <code>agent ENCODE <password></code> .
- WORK_REPOSITORY=<work repository name>	Name of the work repository containing the scenarios to be executed.

See also:

- Launching a listener agent

- Launching a web agent
- Executing a scenario from Designer
- Executing a scenario from an OS command
- Displaying the Scheduling Information
- Stopping an agent

Displaying Scheduling Information

The Scheduling Information allows you to visualize the agents' scheduled tasks.

Important: 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.

To Display the Scheduling information from Operator:

1. Open **Operator**
2. Click on the **Scheduling Information** button in the toolbar

The **Scheduling Information** window for all agents appears.

To Display the Scheduling information from Topology Manager:

1. Open **Topology Manager**
2. In **Physical Architecture** view, select the **Physical Agent** you wish to display the planning.
3. Right click and select **Scheduling Information**

The **Scheduling Information** window for this agent appears.

Launching a Web Agent

A web agent is a Java TCP/IP service that can be used to execute scenarios via a web interface.

This agent works as follows:

- On startup, it launches an HTTP server.
- This HTTP server serves web pages contained in the `/bin/web` directory, which can contain the Scenario Launcher applet.
- When a user starts a scenario through the Scenario Launcher applet, the agent executes the scenario and passes the result to the applet.

Note: A web agent has all the functionality of a scheduler and listener agent.

To launch a web agent:

1. Open a Unix shell, Windows command line, or QSH (for AS/400) session
2. Launch the appropriate command file (agentweb.bat on Windows, agentweb.sh on Unix or AS/400-QSH), with any necessary parameters.

The agent starts up.

Examples:

- On Windows: `agentweb.bat "-port=20300" "-WEB_PORT=8080"`
- On Unix: `agentweb.sh`

Agent Parameters

Caution: To launch a web agent, the parameters for connecting to the work repository where the scenarios are stored must be specified. These parameters are generally stored in the `odiparams` file.

The table below lists the different parameters that allow the agent to be configured according to your needs. The parameters are preceded by the "-" character and the possible values are preceded by the "=" character. You should take into account the syntax for delimiting strings specific to the command line used.

Parameters	Description
<code>-port=<port></code>	Port on which the agent is listening. If this parameter is not specified, the agent listens on the default port of 20910.
<code>-help</code>	Displays the options and parameters that can be used, without launching the agent.
<code>-name=<agent name></code>	<p>This is the name of the physical agent used.</p> <p>Oracle Data Integrator needs this parameter to identify which agent executed a given session in the following situations:</p> <ul style="list-style-type: none"> • More than one agent is declared on the same machine (on different ports). • Your machine's IP configuration prevents Oracle Data Integrator from identifying the agent. This problem occurs frequently on AS/400. • The agent's IP address does not allow the agent to be identified (127.0.0.1 or "loopback").
<code>-v=<trace level></code>	<p>Specifies the level of verbosity for the agent. There are five trace levels:</p> <ol style="list-style-type: none"> 1. Displays the start and end of each session. 2. Also displays the start and end of each step. 3. Also displays each task executed. 4. Also displays each SQL query executed. 5. Displays all information, generally reserved for support calls. <p>As some of these levels can produce a lot of output, it is recommended that you redirect them to a text file using the following command:</p> <ul style="list-style-type: none"> • On Windows: <code>agentweb.bat "-v=5" > trace.txt</code> • On Unix: <code>agentweb.sh -v=5 > trace.txt</code>
<code>-WEB_PORT=<http port></code>	HTTP port for the agent to listen for web requests on.
Work repository connection parameters	These parameters, which specify how to connect to the work repository, are specified in the file <code>odiparams</code>.
<code>-SECU_DRIVER=<driver name></code>	The JDBC driver used to access the master repository.

	For example: <code>oracle.jdbc.driver.OracleDriver</code>
<code>-SECU_URL=<url></code>	The JDBC URL used to access the master repository. For example: <code>jdbc:oracle:thin:@168.67.0.100:1522:ORCL</code>
<code>-SECU_USER=<user></code>	The database user for the master repository connection.
<code>-SECU_PASS=<password></code>	Password of the user for the master repository connection. This password must be encrypted using the command <code>agent ENCODE <password></code> .
<code>-WORK_REPOSITORY=<work repository name></code>	Name of the work repository that contains the scenarios to be executed.

See also:

- Launching a listener agent
- Launching a scheduler agent
- Executing a scenario from Designer
- Executing a scenario from an OS command
- Stopping an agent

Stopping an Agent

You can stop a listener, scheduler or web agent which is listening on a TCP/IP port through the `agentstop` command.

To stop an agent:

1. Open a Unix shell, CMD, or QSH (for AS/400) session
2. Launch the appropriate command file (`agentstop.bat` on Windows, `agentstop.sh` on Unix or AS/400-QSH), with any required parameters.

The listening agent is stopped.

Important Note: For security reasons, it is only possible to stop an agent from a command line launched on the same machine as where the agent's process was started.
It is not possible to stop a remote agent.

Examples:

- On Windows: `agentstop "-PORT=20300"` stops the listener agent on the port 20300.
- On UNIX: `./agentstop.sh` stops the listener agent on the default port.

Command Parameters

The table below lists the different parameters for the command. The parameters are preceded by the "-" character and the possible values are preceded by the "=" character. You should take into account the syntax for delimiting strings used by the relevant operating system.

Parameters	Description
------------	-------------

- `PORT=<port>` Port on which the agent to be stopped is listening. If this parameter is not specified, the default port of 20910 is used.

See also:

- Launching a listener agent
- Launching a scheduler agent
- Launching a web agent

Load Balancing

Oracle Data Integrator implements load balancing between physical agents.

Concepts

Each physical agent is defined with:

- a **Maximum number of sessions** it can execute simultaneously,
- 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.

Determining the Maximum number of sessions

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 Data Integrator 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.

If the user parameter "Use new load balancing" is in use, 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 some from another agent.

Note: An agent can be linked to itself. An agent not linked to itself is only able to delegate sessions to its linked agents, and will never execute a session.

Note: Delegation works on cascades of linked agents. Besides, it is possible to make loops in agents links. This option is not recommended.

Agent unavailable

When for a given agent the number of running sessions is equal to its **Maximum number of sessions**, the agent will set incoming sessions in a "queued" status until the number of running sessions falls below the maximum of sessions for this agent.

Setting up load balancing

To setup load balancing:

1. Define a set of physical agents, and link them to a root agent (see Create a Physical Agent).
2. Start the root and the linked agents.
3. Run the executions on the root agent. Oracle Data Integrator will balance the load of the executions between its linked agents.

Note: The execution agent for a session can be seen in the session window in Operator.

Note: For load balancing to work between agents, it is necessary to have name the agents, that is start them with the `-NAME` parameter. See *Launching a Listener Agent* for more details.

See also:

- Physical Agent
- Creating a Physical Agent
- Session

Other Operations

Encrypting a Password

Passwords that are used in the scripts that are referenced when launching the agents or the scenarios must be encrypted for security reasons.

To encrypt a password:

1. Open a shell or a DOS command prompt.
2. CD to the `/bin` directory in the Oracle Data Integrator installation directory.
3. Type in the following command:
`agent ENCODE <password>`
where `<password>` is the password you wish to encrypt.

Oracle Data Integrator returns a string which is your encrypted password.

There is no method to decrypt an encrypted password.

Topology: Generating a report

To generate a topology report:

1. In **Topology Manager**, select **File > Print**, then the report you want to generate. You can generate reports for the **Logical Architecture**, the **Physical Architecture** and the **Contexts**.
2. Fill in the output file location, or select an output file.
3. Click **OK**

A report in **Adobe™ PDF** format is generated in the file specified in step 2. This file is located in your default directory for pdf generation if no directory was specified.

If the **Open the file after the generation?** option is selected, Oracle Data Integrator will open **Acrobat® Reader™** to view the generated report.

Note: In order to visualize the generated report, you must specify the location of the **Acrobat® Reader™** program in the Users parameters.

Setting User Parameters

Oracle Data Integrator saves user parameters such as default directories, windows positions, etc. User parameters are saved in the `userpref.xml` file in `/bin`.

To set user parameters:

1. Select **User Parameters** from the **File** menu.
2. In the **Editing User Parameters** window, change the value of parameters as required.
3. Click **OK** to save and close the window.

A list of the possible user parameters is available in the Reference Manual.

Import/Export

Exporting the Topology/Security Settings

Exporting then importing the topology or security allows you to transfer a domain from one master repository to another.

The domains that can be exported are given below:

- **Topology**: the full topology (logical and physical architectures).
- **Logical Topology**: technologies (connection, datatype or language information), logical agents and logical schemas.
- **Security**: users, profiles, privileges and hosts. This export is possible from **Security Manager**.
- **Execution Environment**: technologies, datatypes, contexts, physical schemas and agents.

To export the topology/security:

1. In Topology Manager (or Security Manager), select one of the following menu options of the **File > Export** menu :
 - **Topology...**
 - **Logical Topology...**
 - **Security Settings...**
 - **Execution Environment...**
2. Fill in the export parameters:
 - **Export to directory**: Directory in which the export file(s) will be created.
 - **Export to zip file**: When this option is selected, a unique compressed file containing all export files will be created. Otherwise, a set of export files is created.
 - **Zip File name**: Name of the compressed file if the option **Make a Compressed Export** is selected.
 - **Character Set**: Encoding specified in the export file. Parameter `encoding` in the XML file header.
 - **Java Character Set**: Java character set used to generate the file.
3. Click **OK**.

The export file(s) are created in the Export Directory specified.

See also:

Exporting the Master Repository

Importing the Master Repository

Importing the Topology

Exporting an Object

Exporting and importing allows you to transfer Oracle Data Integrator Objects 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 tree view.
2. Right-click on the object, and select **Export....** If this menu item does not appear, then this type of object does not have the export feature.
3. Set the Export parameters, and click **OK**.
You must at least specify the **Export Name**. If the **Export Directory** is not specified, then the export file is created in the **Default Export Directory**.

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

See also:

Importing an Object

Object export parameters

Exporting Multiple Objects

Importing the Topology/Security Settings

Exporting then importing the topology or security allows you to transfer a domain from one master repository to another.

To import a topology export:

1. In Topology Manager, click on **File > Import** then one of the following options:
 - **Topology ...**
 - **Logical Topology...**
 - **Execution environment...**
2. Select the **Import Mode**, the import **Repository** or **Zip File**.
3. Click **OK**.

The specified file(s) are imported into the master repository.

To import a Security export:

1. In Topology Manager, click on **File > Import > Security Settings...** then one of the following options:
2. Select the **Import Mode**, the import **Repository** or **Zip File**.
3. Click **OK**.

The specified file(s) are imported into the master repository.

See also:

Exporting the Master Repository

Importing the Master Repository

Exporting the Topology

Import Modes

Importing an Object

Importing and exporting allows you to transfer objects (Interfaces, Knowledge Modules, Models, ...) from one repository to another.

To import an object in Oracle Data Integrator:

1. Select the appropriate object for import in the tree. If such an object does not exist, then create it.
2. Right-click on the object, and select **import**, then the type of the object you wish to import.
3. Specify the **Export Directory**, and select one or more files in **Import Name**. Select the **Import Mode**, then click on **OK**.

The XML-formatted files are imported into the work repository, and become visible in Oracle Data Integrator.

See also:

Exporting an Object

Import Modes

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.

This information is useful for support purposes.

You can customize the format of this file.

To produce the technical environment file:

1. In Topology Manager, select **File > Export > Export Technical Environment**. The **Export technical environment** window opens.
2. Specify the following properties:
 - **Export Directory**
 - **File Name**
 - **Character set** of export file
 - **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. See the list of codes below.
 - **Record separator**
 - **Field separator**
 - **Field delimiter**
3. Press **OK**.

Record Codes

Properties	Description
Oracle Data Integrator	Code used to identify rows that describe the current version of

Information Record Code	Oracle Data Integrator and the current user. This code is used in the first field of the record.
Master Record Code	Code for rows containing information about the Master Repository.
Work Record Code	Code for rows containing information about the Work Repositories.
Agent Record Code	Code for rows containing information about the various agents that are running.
Technology Record Code	Code for rows containing information about the data servers, their versions, etc.

Security Manager

Introduction to Security Manager

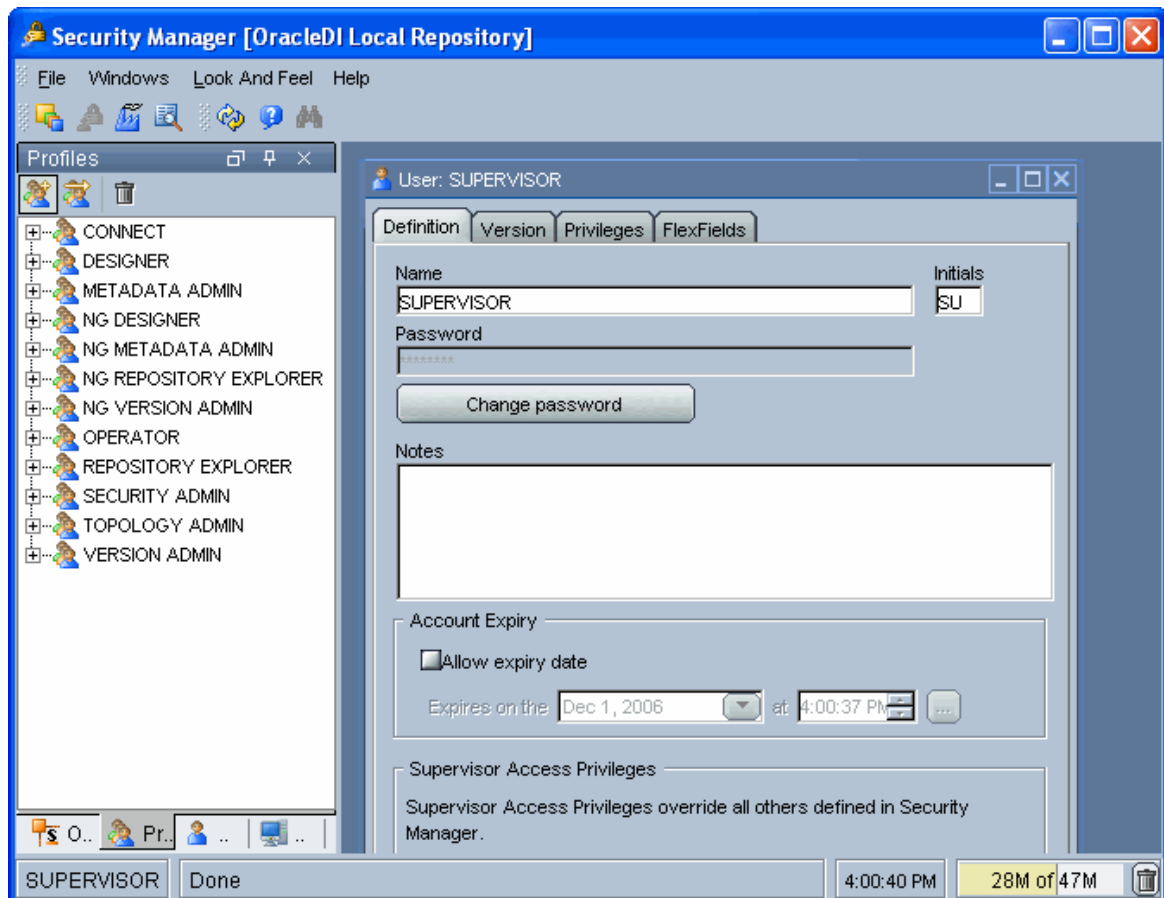
Using the **Security Manager** module, you can manage security in Oracle Data Integrator. The Security Manager module allows **users** and **profiles** to be created. It is used to assign user rights for **methods** (edit, delete, etc) on generic **objects** (data server, datatypes, etc), and to fine-tune these rights on the object **instances** (Server 1, Server 2, etc).

The Security Manager module stores this information in a master repository. This information can be used by all the other modules.

- Define the Security Policy

Security Manager's Interface

The Security Manager GUI appears as follow:



The Menu

The **Menu** contains pull-down menus to access the following features:

- Import/Export

- Wizards
- Display options
- Open modules or tree views
- Change the user's password and options

The Toolbar

The **Toolbar** lets you:

- Open other modules
- Refresh the Tree views
- Open the on-online help

The Tree Views

Security Manager objects available to the current user are organized into the following tree views:

- The **objects**, describing each Oracle Data Integrator elements type (datastore, model,...),
- The users **profiles** and their **authorizations**,
- the **users** and their **authorizations**.

Each tree view appears in a floatable frames that may be docked to the sides of the main window. These frames can be also be stacked up. When several frames are stacked up, tabs appear at the bottom of the frame window to access each frame of the stack.

Tree view frames can be moved, docked and stacked by selecting and dragging the frame title or tab. To lock the position of views, select **Lock window layout** from the **Windows** menu.

If a tree view frame does not appear in the main window or has been closed, it can be opened using the **Windows > Show View** menu.

It is possible in each tree view to perform the following operations:

- Insert or import root objects to the tree view by clicking the appropriate button in the frame title
- Expand and collapse nodes by clicking on them
- Activate the methods associated to the objects (Edit, Delete, ...) through the popup menus
- Edit objects by double-clicking on them, or by drag and dropping them on the **Workbench**.

The Workbench

The windows for object being edited or displayed appear in the **Workbench**.

Defining the Security Policy

Security Concepts

Objects, instances and methods

An **object** is a representation of an element that can be handled through Oracle Data Integrator (agents, models, datastores, etc). The objects are the visible part of Oracle Data Integrator object

components (Java classes). It is necessary to bring together the notions of object and object instance (or instances), which in Oracle Data Integrator are similar to Object-Oriented programming notions.

An **instance** (object instance) is attached to an object type (an object). For example, the project MY_PROJ_1 is an instance of the project object type. Similarly, another instance of a project-type object is YOUR_PROJ_2.

A **method** is a type of action which can be performed on an **object**. Each object has a series of methods that are specific to it. The notion of method in Data Integrator is similar to the one in Object-Oriented programming.

Warning! Objects and methods are predefined in Oracle Data Integrator and should not be changed.

Note: Specific objects have a **double name** (Agent/Context, Profile/Method, etc.). These objects represent links between objects. These links are also objects. For instance, Agent/Context corresponds to a the physical/logical agent association made through the contexts. The privileges on this object enable to changes this association in the topology.

Profiles

A **profile** represents a generic rights model for working with Oracle Data Integrator. One or more profiles can be assigned to a user.

An **authorization by profile** is placed on an **object's method** for a given **profile**. It allows a user with this profile to be given - either optionally or automatically - the right to this object via the method.

- The presence of an authorization by profile for a method, under an object in the tree of a profile, shows that a user with this profile is entitled (either optionally or automatically) to this object's instances via this method.
- The absence of authorization by profile shows that a user with this profile cannot - in any circumstance - invoke the method on an instance of the object.

Generic vs Non-Generic profiles

Generic profiles have the **Generic privilege** option checked for all objects' methods. This implies that a user with such a profile is **by default** authorized for all methods of all instances of an object to which the profile is authorized.

Non-Generic profiles are **not by default** authorized for all methods on the instances since the **Generic privilege** option is unchecked for all objects' methods. The administrator must grant the user the rights on the methods for each instance.

If an administrator wants a user to have the rights on no instance by default, but wishes to grant the rights by instance, the user must be granted a non-generic profile.

If an administrator wants a user to have the rights on all instances of an object type by default, the user must be granted a generic profile.

Users

A user in the **Security Manager** module represents an Oracle Data Integrator user, and corresponds to the login name used for connecting to a repository.

A user inherits the following rights:

- The profile rights he/she already has

- Rights on objects
- Rights on instances

An authorization by user is placed on a **method** of an **object** for a given **user**. It allows the user to be given, either optionally or automatically, the right to this object via the method.

- The presence, under an object in a user's tree, of an authorization by user for a method shows that the user has the right (either optionally or automatically) to the instances of this object via this method.
- The absence of authorization by user shows that the user cannot in any circumstances invoke the method on an instance of the object.

An **authorization by object instance** is placed on an **object** instance for a given **user**. It allows the user to be given rights to certain methods on this object instance.

The presence in a user's tree of an authorization by object instance for an instance shows that the user could have specific rights on the object's methods for the given instance (these rights are specified in the window). If an instance is not in the tree of the user's **instances**, then the authorizations by user or by profile on the object that has given the resulting instance apply.

Authorizations by object instances may be defined by work repository. An object instance may be replicated between work repositories (using export/import in *Synonym* modes) or may be usable in different work repositories (such as context) . When such an instance is "usable" in several repositories, privileges may be granted/denied in all repositories, in no repository, or in selected repositories.

For example, It is common to replicate projects from a *Development* repository to a *Test* repository. A developer may be granted *edit* privileges for his project in the *Development* repository, but not on the *Test* repository. On the *Test* repository, he will only be granted with *view* on the project.

Security Policies

There are two main approaches to security in Oracle Data Integrator:

- The strongly secured approach, where users have no rights on objects, using only non-generic profiles. The security administrator must give the users authorizations on instances (on a given interface, for instance). This policy is complex to setup.
- The generic approach, where users inherit the rights of the profiles they have. This policy is suitable for most cases.

It is possible to combine these two approaches.

To define a strongly secured policy:

1. Create appropriate profiles for your working methods, and give them the non-generic rights to objects. You may use the non-generic profiles provided in Oracle Data Integrator.
2. Create the users
3. Give the users the non-generic profiles.
4. Give the users rights on object instances. This operation must be repeated for each new instance created.

To define a generic policy:

1. Create appropriate profiles for your working methods, and give them generic rights to objects. You may use the generic profiles provided in Oracle Data Integrator.
2. Create the users
3. Give the users the generic profiles.

You can also define a password policy to encourage users to use secured password.

Creating Profiles

Creating a New Profile

To create a new profile:

1. Select **profiles** in the tree.
2. Right click and select **Insert profile**.
3. Enter the **name**.
4. Click **OK**.

The profile is displayed in the tree.

Deleting a Profile

To delete a profile:

1. Select the **profile** to be deleted.
2. Right click and select **Delete**.
3. Click **OK**.

The profile disappears from the tree.

Creating Users

Creating a New User

To create a new user:

1. Select **users** from the tree.
2. Right click and select **Insert User**.
3. Enter the **name**, the **initials** and the user's **password** (by clicking on the **input a password** button).
4. Click **OK**.

The user is displayed in the tree.

Assigning a Profile to a User

To assign a profile to a user:

1. Expand the tree to display the **user** you wish to assign the profile to.
2. Select the **profile** you wish to assign, then drag and drop it onto the **user** in the tree.
3. Click **OK**.

The profile is assigned to the user.

Removing a Profile from a User

To remove a profile from a user:

1. Expand the tree to display the **profile** you wish to delete, under the **user** branch.
2. Select the **profile** (under the user) to be deleted from the tree.
3. Right click and select **Delete**.
4. Click **OK**.

The profile is removed from the user.

Deleting a User

To delete a user:

1. Select the **user** to be deleted.
2. Right click and select **Delete**.
3. Click on **OK**.

The user disappears from the tree.

Managing Rights

Assigning an Authorization by Profile or User

To assign an authorization by profile or by user:

1. Expand the tree to display the **user** or the **profile** you wish to assign the authorization to.
2. Under the **object** select the **method** you wish to assign then drag and drop it onto the **user** or the **profile**.
3. Click **OK**.

The authorization is assigned to the user or the profile.

To assign authorizations on all the methods of an object to a profile or a user:

1. Expand the tree to display the **user** or the **profile** you wish to assign the authorization to.
2. Select the **object** to which you wish to assign all the methods then drag and drop it onto the **user** or the **profile**.
3. Click **OK**.

The authorizations on all the methods are assigned to the user or the profile.

Assigning an Authorization by Object Instance

To assign an authorization by object instance to a user:

1. Expand the **Security Manager** tree to display the **user** to whom you want to add the authorization.
2. Open **Designer** and expand its tree to display the object instance to which you want to assign an authorization.
3. Select the **object instance** in the **Designer** tree, then drag and drop it onto the **user** in **Security Manager**.

4. Fine-tune the rights in the authorization by object instance window which appears. You may want to implement following simple privileges policies on methods that you select in the list:
 - to grant all these methods in all repositories, press the **Allow selected methods in all repositories** button.
 - to deny all these methods in all repositories, press the **Deny selected methods in all repositories** button.
 - to grant all these methods in certain work repositories, press the **Allow selected methods in selected repositories** button, then select the repositories in the list.
5. Click **OK**.

Authorization by object instance is assigned to the user or the profile.

Note: Any method for which the user has generic rights is not listed in the authorization by object instance window.

Deleting an Authorization by Object Instance

To delete an authorization by object instance from a user:

1. Open the authorization by object instance window for the instance whose rights you want to change.
2. Click the **Deny All** button.
3. Click **OK**.

The authorizations are removed from the user. If, after this operation, the user no longer has any rights on an instance, the instance disappears from the tree in **Security Manager**.

You can remove all unused object authorizations by object instance using the Security Clean-up Tool.

Deleting an Authorization by Profile or User

To delete an authorization by profile or user:

1. Select the **method** you wish to delete under the **user** or **profile** branch.
2. Right click and select **Delete**.
3. Click **OK**.

The authorization is removed from the user or the profile.

To delete the authorizations for all the methods of an object from a profile or a user:

1. Select the **object** whose rights assignment you wish to delete under the **user** or the **profile** branch
2. Right click and select **Delete**.
3. Click **OK**.

The authorizations are removed from the user or the profile.

Cleaning up Unused Authorizations

Authorizations by object instance are stored in the Master Repository. However, if these objects are deleted from all work repositories, the authorization are not deleted. Indeed, you may wish to retain certain unused authorizations if they refer to objects currently stored in an exported file or in a stored solution, for instance.

The Security Clean-up Tool should be used 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 in order to check the existence of the objects in these repositories.

Note: Oracle Data Integrator objects themselves are never removed - only security privilege information.

To remove unused authorizations:


1. Select **File > Clean up Security Settings...** from the menu.
2. On the **Cleanable** tab, select any useless security settings you wish to remove.
 - Security settings that cannot be cleaned are shown on the Non-cleanable tab
 - Use the **Select All** button to select all settings for cleaning.
3. Press **Delete Selected Security Settings**.

Using Security Manager

Defining the Password Policy

The password policy consists of a set of rules applied on user passwords. This set of rules is checked when the password is defined by the user.

To define the password policy:

1. In **Security Manager**, select **File > Password Policy...**
The **Password Policy** window appears. In this window, a list of rules is displayed.
2. Click the **Add Rule**  button.
A new rule definition window appears. A rule is a set of condition that are checked on passwords.
3. Set a **Name** and a **Description** for this new rule.
4. Add conditions on the password value or length. You can define for example a minimum length for the passwords from this window.
5. Select if you want at least one condition or all of them to be valid to consider that the password meets the rule.
6. Click **OK**.
7. Add as many rules as necessary, and check those of the rules that you want active in your current policy. Only password meeting all the rules are considered as valid for the current policy.
8. You can also define a period of validity for the passwords. Passwords older than this number of days will automatically expire. The users will have to change them.
9. Click **OK** to update the password policy.

Exporting an Object

Exporting and importing allows you to transfer Oracle Data Integrator Objects 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 tree view.
2. Right-click on the object, and select **Export...**. If this menu item does not appear, then this type of object does not have the export feature.
3. Set the Export parameters, and click **OK**.
You must at least specify the **Export Name**. If the **Export Directory** is not specified, then the export file is created in the **Default Export Directory**.

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

See also:

Importing an Object

Object export parameters

Exporting Multiple Objects

Importing an Object

Importing and exporting allows you to transfer objects (Interfaces, Knowledge Modules, Models, ...) from one repository to another.

To import an object in Oracle Data Integrator:

1. Select the appropriate object for import in the tree. If such an object does not exist, then create it.
2. Right-click on the object, and select **import**, then the type of the object you wish to import.
3. Specify the **Export Directory**, and select one or more files in **Import Name**. Select the **Import Mode**, then click on **OK**.

The XML-formatted files are imported into the work repository, and become visible in Oracle Data Integrator.

See also:

Exporting an Object

Import Modes

Setting User Parameters

Oracle Data Integrator saves user parameters such as default directories, windows positions, etc.

User parameters are saved in the `userpref.xml` file in `/bin`.

To set user parameters:

1. Select **User Parameters** from the **File** menu.
2. In the **Editing User Parameters** window, change the value of parameters as required.
3. Click **OK** to save and close the window.

A list of the possible user parameters is available in the Reference Manual.

Use Oracle Data Integrator with ...

Oracle

Creating an Oracle Data Server

An Oracle **Data Server** corresponds to an Oracle Database Instance connected with a specific Oracle user account. This User will have access to several schemas in this instance, corresponding to the **Physical Schemas** in Oracle Data Integrator created under the data server.

Pre-requisites

JDBC Driver

You must install on each machine on which Oracle Data Integrator is installed the Oracle Type 4 JDBC Driver - **Oracle Thin Driver**. This driver directly uses the TCP/IP network layer and requires no other installed component. This driver appears as a `classes12.zip` file that you must copy into your Data Integrator installation directory's `/drivers` sub-directory. Oracle Data Integrator installs a default version of this driver. It is advised, if you have problems with Oracle, to replace the default driver by the one provided with your version of Oracle.

It is also possible to connect an Oracle Server through the Oracle JDBC OCI Driver, or even using ODBC. These methods do not provide very good performance. It is recommended to use the Type 4 driver.

Connection Informations

You must ask the Oracle DBA the following information:

- Network Name or IP address of the machine hosting the Oracle Database.
- Listening port of the Oracle listener.
- Name of the Oracle Instance (SID).
- TNS alias of the connected instance.
- Login and password of an Oracle User.

Creation of the Data Server

To create an Oracle Data Server:

1. Connect to **Topology Manager**
2. Select **Topology > Physical Architecture > Technologies > Oracle** from the tree.
3. Right click and select **Insert 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.
 - **Instance/dblink:** TNS Alias used for this Oracle instance. It will be used to identify the objects on this server from remote servers. This field is necessary for OCI drivers and for DBLink use.

- **User/Password:** Oracle user (with its password), having select privileges on the data schema and select/insert/object creation on the work schemas that will be indicated in the Oracle **Physical Schemas** created under this data server.
5. Fill in the following fields in the **JDBC** tab:
 - **JDBC Driver:** `oracle.jdbc.driver.OracleDriver`
 - **JDBC URL:** `jdbc:oracle:thin:@<network name or ip address of the Oracle machine>:<port of the Oracle listener (1521)>:<name of the Oracle instance>`
 6. Click **Test**.
 7. Click **Test** in the **Test Connection** windows.
 8. A window indicating a **Successful Connection** appears. Click **OK**. If the connection is not successful, please refer to the Common Errors with Oracle.
 9. Click **OK** to validate the creation of the data server

The creation window for this data server's first **Physical Schema** appears.
See [Creating an Oracle Physical Schema](#)

Creating an Oracle Physical Schema

A **Physical Schema** corresponds to a pair of Schemas in Oracle:


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

Note: The Oracle user specified in the data server to which the Physical Schema is attached must have appropriate privileges to perform such operations.

Creation of the Physical Schema

To Create an Oracle Physical Schema:

Note: If you have just created a Data Server, then ignore step 1. The **Physical Schema** window should already be opened.

1. Select the Oracle **Data Server**, right-click and select **Insert Physical Schema**. The **Physical Schema** window appears.
2. Select the Data Schema for this Oracle Data Integrator physical schema in **Schema (Schema)**. A list of the Oracle instance's schemas appears in this field.
3. Select the Work Schema for this Oracle Data Integrator physical schema in **Schema (Work Schema)**. A list of the Oracle instance's schemas appears in this field.
4. 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). For more information, see [Physical Schema](#)
5. Go to the **Context** tab.
6. Select a **Context** and an existing **Logical Schema** for this new **Physical Schema**, then go to step 9.
If no Oracle Logical Schema exists yet, proceed to step 7.
7. Click the  button.

8. Select an existing **Context** in the left column then type the name of a **Logical Schema** in the right column. This Oracle Logical Schema is automatically created and associated to this physical schema in this context.

Warning ! A **Logical Schema** can be associated to only one **Physical Schema** in a given **Context**.

9. Click **OK**.

Creating and Reverse-Engineer an Oracle Model

Create an Oracle Model

An Oracle **Model** is a set of datastores corresponding to views and tables contained in an Oracle Schema. A model is always based on a **Logical Schema**. In a given **Context**, the **Logical Schema** corresponds to a **Physical Schema**. The Data Schema of this Physical Schema contains the Oracle model's tables and views.

To create an Oracle Model:

1. Connect to **Designer**
2. Select **Models** in the tree
3. Right click and select **Insert Model**.
4. In the **Definition** tab, fill in the **Name** field.
5. In the **Technology** field, select **Oracle**.
6. In the **Logical Schema** field, select the Logical Schema on which your model will be based.
7. Go to the **Reverse** tab, and select a **Context** which will be used for the model's reverse-engineering. Click **Apply**.

The model is created, but contains no datastores yet.

Reverse-engineer an Oracle model

A model is created with no datastores. The **Reverse-engineering** operation consists of retrieving the structure of the model's tables and views to create the model's datastore definitions. There are two types of reverse-engineering, the **Standard** reverse-engineering, which uses only the abilities of the driver, and the **Customized** reverse-engineering, which uses a RKM (Reverse Knowledge Module) to get the structure of the objects directly from the Oracle dictionary.

Note: The Customized reverse-engineering is more complete and entirely customizable, but also more complex to run. It is recommended that you use the standard reverse-engineering for Oracle.

Standard Reverse-Engineering

To perform a Standard Reverse- Engineering on Oracle:

1. Go to the **Reverse** tab of your Oracle **Model**.
2. Fill in the following fields:
 - **Standard**
 - **Context:** Context used for the reverse-engineering

- **Types of objects to reverse-engineer:** List of object types that should be taken into account by the reverse-engineering process.
3. Go to the **Selective Reverse** tab.
 - Check the **Selective Reverse**, **New Datastores**, and **Objects to Reverse** boxes.
 4. A list of datastores to be reverse-engineered appears. Leave those you wish to reverse-engineer checked.
 5. Click the **Reverse** button, then on **Yes** to validate the changes.
 6. Oracle Data Integrator launches a reverse-engineering process of the selected datastores; A progress bar appears.

The reverse-engineered datastores appear under the model.

It is possible to refine the reverse-engineer using the options in the **Reverse** and **Selective Reverse** tabs. Refer to Model for more information.

Customized Reverse-Engineering

To perform a Customized Reverse-Engineering on Oracle by RKM (Oracle 7.3 and above):

Warning ! Using a RKM requires the import of this RKM in a project. Only the imported RKM are available for reverse operations.

1. Go to the **Reverse** tab of the Oracle **Model**.
2. Fill in the following fields:
 - **Customized**
 - **Context:** Context used for the reverse-engineering
 - **Object Type:** Type of the objects to reverse-engineer (Tables, view...)
 - **Mask:** %
 - **Select the KM:** RKM Oracle.<name of the importation project>
3. Click the **Reverse** button, then **Yes** to validate the changes.
4. Click **OK**
5. The **Session started** window appears. Click **OK**

Follow and validate the reverse-engineering operations for your Oracle model in the Operator. If it finished correctly, the reversed datastores appear under the reversed module in the tree. It is possible to refine the reverse-engineer using the options in the **Reverse** and **Selective Reverse** tabs. Refer to Model for more information. See also the RKM Oracle description for more information about this RKM.

Choosing the Right KMs for Oracle

The KM choice for an interface of a check determines the abilities and performances of this interface or check. The recommendations below help in the selection of the KM for different situations concerning an Oracle Data Server.

For generic information about the KM, see Knowledge Modules

Note: Only knowledge modules imported into a project are available for the interfaces of this project. To import a KM, see Import a KM.

Load data from and to Oracle

Oracle can be used as a source, target or staging area of an interface. The LKM choice in the Interface Flow tab to load data between Oracle and another type of data server is essential for interfaces' performances.

LKM Choice from Oracle

Recommendations for LKM in the following cases:

- Oracle Source to a Staging Area

If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Target or Staging Area Technology	Recommended KM	Notes
Oracle	LKM Oracle to Oracle (dblink)	Views created on the source server, synonyms on the target
Oracle	LKM ISO SQL to Oracle	If the use of DBLinks is not possible
MS SQL Server	LKM ISO SQL to MSSQL (bulk)	Uses SQL Server's bulk loader.
Sybase	LKM ISO SQL to Sybase (bcp)	Uses Sybase's bulk loader.
All	LKM ISO SQL to SQL	Generic KM

LKM Choice to Oracle

Recommendations for LKM in the following cases:

- Source to Oracle Staging Area

If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Source or Staging Area Technology	Recommended KM	Notes
Oracle	LKM Oracle to Oracle (dblink)	Views created on the source server, synonyms on the target
JMS	LKM JMS to SQL	JMS cannot be Staging Area
File	LKM File to Oracle (sql*loader)	Uses Oracle's bulk loader. File cannot be Staging Area.
File	LKM ISO File to SQL	Slower than Oracle's bulk loader. File cannot be Staging Area.
All	LKM ISO SQL to Oracle	Faster than the Generic LKM (Uses Statistics)
All	LKM ISO SQL to SQL	Generic KM

Check data in Oracle

The control strategies for Oracle can be chosen in an Oracle Model for a static control, but also for a Flow Control when the Staging Area is on Oracle.

CKM choice for Oracle

Recommendations for the KM to perform a check on Oracle. If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Recommended KM	Notes
CKM Oracle	Uses Oracle's Rowid to identify records
CKM ISO SQL	Uses the PK to identify records

Integrate data in Oracle

The data integration strategies in Oracle are numerous and cover several modes. The IKM choice in the Interface Flow tab determines the performances and possibilities for integrating.

IKM choice for Oracle

Recommendations for the KM to perform an integration on Oracle, depending on the chosen integration mode.

The IKM listed with a specific target are multi-technology KM. they are usable when Oracle is the Staging Area. They do not support flow control.

If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Mode	Target	Recommended KM	Notes
Update		IKM Oracle Incremental Update	Optimized for Oracle
Update		IKM ISO SQL Incremental Update	Generic KM
Specific		IKM Oracle Slowly Changing Dimension	Supports type 2 Slowly Changing Dimensions
Append	Any RDBMS	IKM ISO SQL to SQL Append	No flow control
Append	JMS	IKM ISO SQL to JMS Append	No flow control
Append	File	IKM ISO SQL to File Append	No flow control
Append		IKM ISO SQL Control Append	Generic KM

Common Errors with Oracle

Detect the errors coming from Oracle

Errors appear often in Oracle Data Integrator in the following way:

```
java.sql.SQLException: ORA-01017: invalid username/password; logon
denied
at ...
at ...
...
```

the `java.sql.SQLException` code simply indicates that a query was made to the database through the JDBC driver, which has returned an error. This error is frequently a database or driver error, and must be interpreted in this direction.

Only the part of text in bold must first be taken in account. It must be searched in the Oracle documentation. If its contains an error code specific to Oracle, like here (in red), the error can be immediately identified.

If such an error is identified in the execution log, it is necessary to analyze the SQL code send to the database to find the source of the error. The code is displayed in the **description** tab of the erroneous **task**.

The most common errors with an Oracle server are detailed below, with their principal causes.

Common Errors

Connection Errors

UnknownDriverException

The JDBC driver is incorrect. Check the name of the driver.

I/O Exception: Connection

```
refused(DESCRIPTION=(TMP=)(VSNNUM=135290880)(ERR=12505)(ERROR_STACK=(ERROR
=(CODE=12505)(EMFI=4))))
```

The instance name in the JDBC URL is invalid.

I/O Exception: The Network Adapter could not establish the connection

The IP address, machine name of Oracle listener port is incorrect in the JDBC URL.

ORA-01017: invalid username/password; logon denied

The user and/or password specified in the data server definition is invalid. This error may also appear for certain Oracle Data Integrator commands, such as `SqlUnload`.

Protocol violation

This error indicates an incompatibility between the Oracle JDBC driver and the database you connect to. If it occurs at connection time, or at the first operation launched on the Oracle database, then install the version of the Oracle JDBC driver provided with your database installation.

ORA-00600 internal error code

Internal error of the Oracle database. May be caused by a driver incompatibility.

ORA-12154 TNS:could not resolve service name

TNS alias resolution. This problem may occur when using the OCI driver, or a KM using DBLinks. Check the configuration of the TNS aliases on the machines.

ORA-02019 connection description for remote database not found

You use a KM using non existing DBLinks. Check the KM options and pre-requisites.

Errors in the interfaces

ORA-00900 invalid SQL statement

ORA-00923 FROM Keyword not found where expected.

The code generated by the interface, or typed in a procedure is invalid for Oracle. This is usually related to an input error in the mapping, filter of join. The typical case is a missing quote or an unclosed bracket.

A frequent cause is also the call made to a non SQL syntax, like the call to an Oracle stored procedure using the syntax `EXECUTE SCHEMA . PACKAGE . PROC (PARAM1 , PARAM2)`.

The valid SQL call for a stored procedure is:

```
BEGIN
SCHEMA . PACKAGE . PROC (PARAM1 , PARAM2 ) ;
END;
```

The syntax `EXECUTE SCHEMA . PACKAGE . PROC (PARAM1 , PARAM2)` is specific to SQL*PLUS, and do not work on the Oracle JDBC Thin driver.

ORA-00904 invalid column name

Keying error in a mapping/join/filter. A string which is not a column name is interpreted as a column name, or a column name is misspelled.

This error may also appear when accessing an error table associated to a datastore with a recently modified structure. It is necessary to impact in the error table the modification, or drop the error tables and let Oracle Data Integrator recreate it in the next execution.

ORA-00903 invalid table name

The table used (source or target) does not exist in the Oracle schema. Check the mapping logical/physical schema for the context, and check that the table physically exists on the schema accessed for this context.

ORA-00972 Identifier is too Long

There is a limit in the object identifier in Oracle (usually 30 characters). When going over this limit, this error appears. A table created during the execution of the interface went over this limit. and caused this error (see the execution log for more details).

Check in the topology for the oracle technology, that the maximum lengths for the object names (tables and columns) correspond to your Oracle configuration.

ORA-01790 expression must have same datatype as corresponding expression

You are trying to connect two different values that can not be implicitly converted (in a mapping, a join...). Use the explicit conversion functions on these values.

DB2 for iSeries

Creating a DB2/400 Data Server

A DB2/400 **Data Server** corresponds to one DB2 Database installed on an AS/400 system, connected with a specific user account. This user will have access to **libraries** in this system that corresponding to Oracle Data Integrator's **Physical Schemas** created under the data server.

Pre-requisites

JDBC Driver

It is preferable to use a Type 4 JDBC Driver for DB2/400. Two drivers are recommended : The **IBM JT/400**, or the **HiT JDBC/400** Driver. These drivers directly use the TCP/IP network layer and require no other components installed on the client machine.

It is also possible to connect through ODBC with the IBM Client Access component installed on the machine. This method does not have very good performance and does not support the reverse engineering and some other features. It is therefore not recommended.

IBM JT/400 and Native Drivers

This driver appears as a `jt400.zip` file you must copy into your Oracle Data Integrator installation directory's `/drivers` sub-directory. Oracle Data Integrator installs a default version of this driver, which is tested and validated with Oracle Data Integrator. This driver is free of charge.

To connect DB2/400 with a Java application installed on the AS/400 machine, IBM recommends that you use the JT/400 Native driver (`jt400native.jar`) instead of the JT/400 driver (`jt400.jar`). The Native driver provides optimized access to the AS/400, but works only from the AS/400.

To support seamlessly both drivers with one connection, Oracle Data Integrator provides the **Oracle Data Integrator Driver Wrapper for AS/400**. This wrapper connects through the Native driver if possible, otherwise it uses the JT/400 driver. It is recommended that you use this wrapper if running agents installed on AS/400 systems.

To install and configure the driver wrapper:

1. Check that the `snpsdb2.jar` file (The wrapper package) is installed as a driver for all machines. See *Installing JDBC / JMS drivers* for more information.
2. In Topology Manager, change the driver and URL to your AS/400 server with the following information:
 - Driver: `com.sunopsis.jdbc.driver.wrapper.SnpsDriverWrapper`
 - URL: `jdbc:snps400:<machine_name>[;param1=valeur1[;param2=valeur2...]]`
3. For the Oracle Data Integrator installations on AS/400 systems, set the following java properties in the agent startup command, then restart the agent:
 - `HOST_NAME`: comma separated list of AS/400 host names for the current machine.
 - `HOST_IP`: IP Address of the current machine.

To set properties for an agent started with the JAVA CL command, you can take example from the following syntax example.

```
JAVA CLASS('oracle.odi.Agent') +
  PARM('-SECU_DRIVER=com.ibm.as400.access.AS400JDBCdriver' ... '-
  PORT=20910') +
  CLASSPATH('/ODI/LIB/odi.zip:/ODI/DRIVERS/snpsdb2.jar:/ODI/DRIVERS/j
  t400native.jar'...) +
  ... +
  PROP((HOST_NAME 'HAL,HALM,HALW,HALE') (HOST_IP '192.168.0.13'))
```

Note that the `snpsdb2.jar` (driver wrapper) and `jt400native.jar` (Native Driver) files are in the classpath, and we have specified the properties in the last line.

If starting the agent using the UNIX shell script, just edit the `odiparams.sh` as below:

```
ODI_ADDITIONAL_JAVA_OPTIONS="-DHOST_NAME=HAL,HALM,HALW,HALE -
HOST_IP=192.168.0.13"
```

For all driver parameters, refer to JDBC Drivers Samples.

HiT JDBC/400

This driver is similar to the JT/400 driver. It is licensed separately from Oracle Data Integrator. For more information, see www.hitsw.com.

Connection Informations

You must ask the system administrator the following information:

- Network Name or IP address of the AS/400 system.
- Login and password of an AS/400 User.

Creation of the Data Server

To create a DB2/400 Data Server:

1. Connect to **Topology Manager**
2. Select **Topology > Physical Architecture > Technologies > IBM DB2/400** in the tree.
3. Right click and select **Insert 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.
 - **User/Password:** AS/400 user (with its password), having at least select privileges on the data libraries, and select/insert/object creation on the work libraries that will be indicated in the **Physical Schemas** created under this data server.
5. Fill in the following fields in the **JDBC** tab, according to the driver used:
 - If using **IBM JT/400** Driver
 - **JDBC Driver:** `com.ibm.as400.access.AS400JDBCdriver`
 - **JDBC URL:** `jdbc:as400://<network name or IP address of the AS/400 machine>`
 - If using **HiT JDBC/400** Driver
 - **JDBC Driver:** `hit.as400.As400Driver`
 - **JDBC URL:** `jdbc:as400://<network name or IP address of the AS/400 machine>`
6. Click **Test**.
7. Click **Test** in the **Test Connection** windows.
8. A window indicating a **Successful Connection** appears. Click **OK**. If the connection is not successful, please refer to Common Errors with DB2/400.
9. Click **OK** to validate the creation of the data server

The creation window for the first **Physical Schema** for this data server appears.
See [Creating a DB2/400 Physical Schema](#)

Creating a DB2/400 Physical Schema

A **Physical Schema** corresponds to a pair of libraries (or collections) in DB2/400:

- A (Data) **Schema**, in which Oracle Data Integrator will look for the source and target tables and files for the interfaces.
- A **Work Schema** in which Oracle Data Integrator can create and manipulate temporary work tables associated to the sources and targets contained in the Data Library.


Note: The user profile specified in the data server to which the Physical Schema is attached must have appropriate privileges to perform such operations.

Note: Most of the Knowledge modules make use of commitment control in the staging area and when writing to the target datastores. We strongly recommend that you use **collections** as staging areas on AS/400, and to have the target tables journaled.

Creation of the Physical Schema

To Create a DB2/400 Physical Schema:

Note: If you have just created a Data Server, then ignore step 1. The **Physical Schema** window should already be opened.

1. Select the DB2/400 **Data Server**, right-click and select **Insert Physical Schema**. The **Physical Schema** window appears.
2. Select the Data Library for this Oracle Data Integrator physical schema in **Schema (Schema)**. A list of the DB2/400 database's libraries appears in this field.
3. Select the Work Library for this Oracle Data Integrator physical schema in **Schema (Work Schema)**. A list of the of the DB2/400 database's libraries appears in this field.
4. 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). For more information, see Physical Schema
5. Go to the **Context** tab.
6. Select a **Context** and an existing **Logical Schema** for this new **Physical Schema**, then go to step 9.
If no DB2/400 Logical Schema exists yet, proceed to step 7.
7. Click the  button.
8. Select an existing **Context** in the left column, the type the name of a **Logical Schema** in the right row. This DB2/400 Logical Schema is automatically created and associated to this physical schema in this context.

Warning ! A **Logical Schema** can be associated to only one **Physical Schema** in a given **Context**.

9. Click OK.

Creating and Reverse-Engineering a DB2/400 Model

Create a DB2/400 Model

A DB2/400 **Model** is a set of datastores corresponding to views, tables and files contained in an DB2/400 library. A model is always based on a **Logical Schema**. In a given **Context**, the **Logical Schema** corresponds to one **Physical Schema**. The Data Schema of this Physical Schema contains the DB2/400 model's tables, files and views.

To create a DB2/400 Model:

1. Connect to **Designer**
2. Select **Models** in the tree
3. Right click and select **Insert Model**.
4. In the **Definition** tab, fill in the **Name** field.
5. In the **Technology** field, select **IBM DB2/400**.
6. In the **Logical Schema** field, select the Logical Schema on which your model will be based.
7. Go to the **Reverse** tab, and select a **Context** which will be used for the model's reverse-engineering. Click **Apply**.

The model is created, but contains no datastores yet.

Reverse-engineer a DB2/400 model

A model is created with no datastore. The **Reverse-engineering** operation consists in retrieving the structure of the model's tables, files and views to create the model's datastore definitions. There are two types of reverse-engineering: the **Standard** reverse-engineering, which uses only the abilities of the driver, and the **Customized** reverse-engineering, which uses a RKM (Reverse Knowledge Module) to get the structure of the objects directly from the DB2/400 dictionary.

Standard Reverse-Engineering

To perform a Standard Reverse-Engineering on DB2/400:

1. Go to the **Reverse** tab of your DB2/400 **Model**.
2. Fill in the following fields:
 - **Standard**
 - **Context**: Context used for the reverse-engineering
 - **Types of objects to reverse-engineer**: List of object types that should be taken into account by the reverse-engineering process.
3. Go to the **Selective Reverse** tab.
 - Check the **Selective Reverse**, **New Datastores**, and **Objects to Reverse** boxes.
4. A list of datastores to be reverse-engineered appears. Leave those you wish to reverse-engineer checked.
5. Click the **Reverse** button, then on **Yes** to validate the changes.
6. Oracle Data Integrator starts a reverse-engineering process on the selected datastores; A progress bar appears.

The reverse-engineered datastores appears under the model.

It is possible to refine the reverse-engineering using the options of the **Reverse** and **Selective Reverse** tabs. Refer to Model for more information.

Using CDC on iSeries

Oracle Data Integrator handles Changed Data Capture on iSeries with two methods:

- **Using Triggers** on the journalized tables. This method is set up with the *JKM DB2/400 Simple* or *JKM DB2/400 Consistent*. This CDC is not different from the CDC on other systems. Refer to the Change Data Capture topic for more information.

- **Reading Native iSeries Transaction Journals.** This method is set up with the *JKM DB2/400 Journal Simple* and used by the *LKM DB2/400 Journal to SQL*. Although it usually provided performances, this method does not support Consistent Set CDC, and requires a platform-specific configuration detailed below.

How does it work?

A iSeries transaction journal contains the entire history of the data changes for a given period. It is handled by the iSeries system for tables that are journaled. A journaled table is either a table from a collection, or a table for which a journal receiver and a journal have been created and journaling started.

Reading the transaction journal is performed by the a journal retriever `CDCRTVJRN` RPG program provided with Oracle Data Integrator. This program loads on demand the tables of the Oracle Data Integrator CDC infrastructure (J\$ tables) with the contents from the transaction journal.

This program can be either scheduled on the iSeries system or called by the KMs through a stored procedure also called `CDCRTVJRN`.

This stored procedure is automatically created by the *JKM DB2/400 Journal Simple* and invoked by the *LKM DB2/400 Journal to SQL* when data extraction is needed.

CDCRTVJRN Program Details

This program connects to the native iSeries journal for a given table, and captures changed data information into the Oracle Data Integrator Journal (J\$).

The program works as follows:

1. Journalized table attributes retrieval:
 - a. Table attributes retrieval: PK columns, J\$ table name, last journal reading date.
 - b. Attributes enrichment (short names, record size, etc.) using the `QSYS.QADBXREF` system table.
 - c. Location of the iSeries journal using the `QADBRTVFD()` API.
2. PK columns information retrieval:
 - a. PK columns attributes (short name, data types etc.) using the `QSYS.QADBIFLD` system table.
 - b. Attributes enrichment (real physical length) using the `QUSLFLD()` API.
 - c. Data preprocessing (RPG to SQL datatype conversion) for the primary key columns.
3. Extraction the native journal information into the J\$ table :
 - a. Native journal reading using the `QJoRetrieveJournalEntries()` API.
 - b. Conversion of the raw data to native SQL data and capture into the J\$ table.
 - c. Update of the changes count.

This program accepts the following parameters:

Parameter	RPG Type	SQL Type	Description
SbsTName	A138	Char(138)	Full name of the subscribers table in the following format: <Lib>.<Table>. Example: ODILIB.SNP_SUBSCRIBERS
JrnTName	A138	Char(138)	Full name of the table for which the extract is done from the

		journal. Example: <code>FINANCE.MY_COMPANY_ORDERS</code>
JrnSubscriber	A50 Char(50)	Name of the current subscriber. It must previously have been added to the list of subscribers.
LogMessages	A1 Char(1)	Flag activating logging in a spool file. Possible values are: <code>Y</code> enable logging, and <code>N</code> to disable logging.

Installing the CDC Components on iSeries

There are two major components installed on the iSeries system to enable native journal reading:

- The **CDCRTVJRN Program**. This program is provided in an archive that should be installed in the iSeries system. The installation process is described below.
- The **CDC Infrastructure**. It includes the standard CDC objects (J\$ tables, views, ...) and the **CDCRTVJRN Stored Procedure** created by the JKM and used by the LKM to read journals. This stored procedure executes the CDCRTVJRN program.

Important: The program must be set up in a library defined in the Topology as the default work library for this iSeries data server. In the examples below, this library is called `ODILIB`.

Installing the CDCRTVJRN Program

To install the CDCRTVJRN program:

1. Copy the program save file from the `/tools/cdc_ismseries/` directory to a temporary directory (`C:\temp`) on your system. In our examples, we name this copy `SAVESNPCDC`.
2. Connect to the iSeries system with a terminal.
3. Create the default work library if it does not exist yet:
Example: `CRTLIB LIB(ODILIB)`
4. Create in this library an empty save file that has the same name as the save file (mandatory)
Example: `CRTSAVF FILE(ODILIB/SAVESNPCDC)`
5. Upload the local save file on the iSeries system in the library and on top of the save file you have just created. Make sure that the upload is performed in binary mode. A FTP command sequence performing the upload is given below:

```
FTP 192.168.0.13
LCD C:\TEMP
BI
CD ODILIB
PUT SAVESNPCDC
BYE
```

6. Restore the objects of the `CDCSNPRELE` library from the save file, using the `RSTOBJ` command, to your target library:
`RSTOBJ OBJ(*ALL) SAVLIB(CDCSNPRELE) DEV(*SAVF) OBJTYPE(*ALL) SAVF(ODILIB/SAVESNPCDC) RSTLIB(ODILIB)`
7. You can check that the objects are correctly restored. The target library should contain a program object called `CDCRTVJRN`. Use the command below to view it:
`WRKOBJ OBJ(ODILIB/CDCRTVJRN)`

The CDCRTVJRN Stored Procedure

This procedure is used to call the CDCRTVJRN program. It is automatically created by the *JKM DB2/400 Journal Simple* KM when journalizing is started. Journalizing startup is described in the Change Data Capture topic.

The syntax for the stored procedure is provided below for reference:

```
create procedure ODILIB.CDCRTVJRN(
    SbstTName char(138), /* Qualified Subscriber Table Name */
    JrntTName char(138), /* Qualified Table Name */
    Subscriber char(50) , /* Subscriber Name */
    LogMsg char(1) /* Create a Log (Y - Yes, N - No) */
)
language rpgle
external name 'ODILIB/CDCRTVJRN'
```

Important: The stored procedure and the program are installed in a library defined in the Topology as the default work library for this iSeries data server.

Using the CDC with the Native Journals

Once the program is installed and the CDC is setup, using the native journals consists in using the *LKM DB2/400 Journal to SQL* to extract journalized data from the iSeries system. The retrieval process is triggered if the `RETRIEVE_JOURNAL_ENTRIES` option is set to Y for the LKM.

Problems While Reading Journals

CDCRTVJRN Program Limits

The following limits exist for the CDCRTVJRN program:

- Support iSeries version: v5r2. Please contact the support for other versions.
- The source table should be journaled and the iSeries journal should be readable by the user specified in the iSeries data server.
- The source table should have one PK defined in Oracle Data Integrator.
- The PK declared in Oracle Data Integrator should be in the 4096 first octets of the physical record of the data file.
- The number of columns in the PK should not exceed 16.
- The total number of characters of the PK column names added to the number of columns of the PK should not exceed 255.
- Large object datatypes are not supported in the PK. Only the following SQL types are supported in the PK: SMALLINT, INTEGER, BIGINT, DECIMAL (Packed), NUMERIC (Zoned), FLOAT, REAL, DOUBLE, CHAR, VARCHAR, CHAR VARYING, DATE, TIME, TIMESTAMP and ROWID
- Several instances of CDCRTVJRN should not be started simultaneously on the same system.
- Reinitializing the sequence number in the iSeries journal may have a critical impact on the program (program hangs) if the journal entries consumption date (`SNP_SUBSCRIBERS.JRN_CURFROMDATE`) is before the sequence initialization date. To work around this problem, you should manually set a later date in `SNP_SUBSCRIBERS.JRN_CURFROMDATE`.

Troubleshooting the CDCRTVJRN Program

The journal reading process can be put in trace mode:

- either by calling from your query tool the CDCRTVJRN stored procedure with the `LogMsg` parameter set to Y,
- or by forcing the `CREATE_SPOOL_FILE` LKM option to 1 then restarting the interface.

The reading process logs are stored in a spool file which can be reviewed using the `WRKSPLF` command.

You can also review the raw contents of the iSeries journal using the `DSPJRN` command.

Choosing the Right KMs for DB2/400

The KM choice for an interface or a check determines the abilities and performances of this interface or check. The recommendations below help in the selection of the KM for different situations concerning a DB2/400 Server.

For generic information about the KM, see Knowledge Modules

Note: Only knowledge modules imported into a project are available for the interfaces of this project. To import a KM, see Import a KM.

Load data from and to DB2/400

DB2/400 can be used as a source, target or staging area of an interface. The LKM choice in the Interface Flow tab to load data between DB2/400 and another type of data server is essential for interfaces' performances.

LKM Choice from DB2/400

Recommendations for LKM in the following cases:

- DB2/400 Source to a Staging Area

If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Target or Staging Area Technology	Recommended KM	Notes
Oracle	LKM ISO SQL to Oracle	Faster than the Generic LKM (Uses Statistics)
MS SQL Server	LKM ISO SQL to MSSQL (bulk)	Uses SQL Server's bulk loader.
Sybase	LKM ISO SQL to Sybase (bcp)	Uses Sybase's bulk loader.
All	LKM ISO SQL to SQL	Generic KM

LKM Choice to DB2/400

Recommendations for LKM in the following cases:

- Source to DB2/400 Staging Area

If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Source or Staging Area Technology	Recommended KM	Notes
JMS	LKM ISO JMS to SQL	JMS cannot be Staging Area
File	LKM ISO File to SQL	File cannot be Staging Area.
All	LKM ISO SQL to SQL	Generic KM

Check data in DB2/400

The control strategies for DB2/400 can be chosen in a DB2/400 Model for a static control, but also for a Flow Control when the Staging Area is on DB2/400.

CKM choice for DB2/400

Recommendations for the KM to perform a check on DB2/400. If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Recommended KM	Notes
CKM ISO SQL	Uses the PK to identify records

Integrate data in DB2/400

The data integration strategies in DB2/400 are numerous and cover several modes. The IKM choice in the Interface Flow tab determines the performances and possibilities for integrating.

IKM choice for DB2/400

Recommendations for the KM to perform an integration on DB2/400, depending on the chosen integration mode.

The IKM listed with a specific target are multi-technology KM. they are usable when DB2/400 is the Staging Area. They do not support flow control.

If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Mode	Target	Recommended KM	Notes
Update		IKM DB2/400 Incremental Update	Optimized for DB2/400
Update		IKM ISO SQL Incremental Update	Generic KM
Append	Any DBMS	IKM ISO SQL to SQL Append	No flow control
Append	JMS	IKM ISO SQL to JMS Append	No flow control

Append File	IKM ISO SQL to File Append	No flow control
Append	IKM ISO SQL Control Append	Generic KM

Common Errors with DB2/400

Decoding the error messages

Errors in Oracle Data Integrator appear often in the following way:

Error message with the ODBC driver

```
java.sql.SQLException: [IBM][Client Access ODBC Driver][32 bits][DB2/400 SQL]Communication link failure. Comm RC=0xb
at sun.jdbc.odbc.JdbcOdbc.createSQLException(Unknown Source)
at sun.jdbc.odbc.JdbcOdbc.standardError(Unknown Source)
...
```

Error message with the IBM JT/400 driver

```
java.sql.SQLException: The application server rejected the connection. (Signon was canceled.)
at com.ibm.as400.access.JDError.throwSQLException(JDError.java:336)
at
com.ibm.as400.access.AS400JDBCConnection.setProperties(AS400JDBCConnection.java:1984)
...
```

Error message with the HiT JDBC/400 driver

```
java.sql.SQLException: Cannot open a socket on host: ha, port: 8471 (Exception: java.net.UnknownHostException: ha).
at hit.as400sql.d.<init>([DashoPro-V1.3-013000])
at hit.as400.As400Driver.newConnection([DashoPro-V1.3-013000])
...
```

the `java.sql.SQLException` code simply indicates that a query was made to the database through the JDBC driver (or JDBC/ODBC bridge), which has returned an error. This error is frequently a database or driver error, and must be interpreted in this direction.

Only the part of text in bold must first be taken in account. It must be searched in the driver or database documentation. If it contains an error code specific to DB2/400, the error can be immediately identified.

If such an error is identified in the execution log, it is necessary to analyze the SQL code sent to the database to find the source of the error. The code is displayed in the **description** tab of the erroneous **task**.

The most common errors with a DB2/400 server are detailed below along with their principal causes.

Common Errors

Connection Errors

UnknownDriverException

The JDBC driver is incorrect. Check the name of the driver.

*The application requester cannot establish the connection.(<name or IP address>)
Cannot open a socket on host: <name or IP address>, port: 8471 (Exception:
java.net.UnknownHostException:<name or IP address>)*

Oracle Data Integrator cannot connect to the database. Either the machine name or IP address is invalid, the DB2/400 Services are not started or the TCP/IP interface on AS/400 is not started. Try to ping the AS/400 machine using the same machine name or IP address, and check with the system administrator that the appropriate services are started.

Datasource not found or driver name not specified

The ODBC Datasource specified in the JDBC URL is incorrect.

*The application server rejected the connection.(Signon was canceled.)
Database login failed, please verify userid and password.
Communication Link Failure. Comm RC=8001 - CWBSY0001 - ...*

The user profile used is not valid. This error occurs when typing an invalid user name or an incorrect password.

Communication Link Failure.

An error occurred with the ODBC connectivity. Refer to the Client Access documentation for more information.

Errors in the interfaces

SQL7008 &1 in &2 not valid for operation. The reason code is 3.

The iSeries 400 system implements commitment control by journaling. Any application that takes advantage of commitment control will require that the files used be journaled. Most of the Knowledge modules make use of commitment control in the staging area, and to write in the target datastores. We strongly recommend you use **collections** as staging areas on AS/400 and have the target tables journaled. It is possible to remove the use of commitment control in the knowledge modules by modifying them.

SQL5001 - Column qualifier or table &2 undefined.

SQL5016 - Object name &1 not valid for naming convention

Your JDBC connection or ODBC Datasource is configured to use the wrong naming convention. Use the ODBC Administrator to change your datasource to use the proper (*SQL or *SYS) naming convention, or use the appropriate option in the JDBC URL to force the naming conversion (for instance `jdbc:as400://195.10.10.13;naming=system`). Note that if using the system naming convention in the **Local Object Mask** of the **Physical Schema**, you must enter %SCHEMA/%OBJECT instead of %SCHEMA.%OBJECT.

"*SQL" should always be used unless your application is specifically designed for *SYS. Oracle Data Integrator uses the *SQL naming convention by default.

*SQL0204 &1 in &2 type *&3 not found.*

The table you are trying to access does not exist. This may be linked to an error in the context choice, or in the sequence of operations (E.g. : The table is a temporary table which must be created by another interface).

Hexadecimal characters appear in the target tables. Accentuated characters are incorrectly transferred.

The iSeries computer attaches a language identifier or CCSID to files, tables and even fields (columns). CCSID 65535 is a generic code that identifies a file or field as being language independent: i.e. hexadecimal data. By definition, no translation is performed by the drivers. If you

do not wish to update the CCSID of the file, then translation can be forced, in the JDBC URL, thanks to the flags `ccsid=<ccsid code>` and `convert _ccsid_65535=yes|no`. See the driver's documentation for more information.

SQL0901 SQL system error

This error is an internal error of the DB2/400 system.

SQL0206 Column &1 not in specified tables.

Keying error in a mapping/join/filter. A string which is not a column name is interpreted as a column name, or a column name is misspelled.

This error may also appear when accessing an error table associated to a datastore with a structure recently modified. It is necessary to impact in the error table the modification, or drop the error tables and let Oracle Data Integrator recreate it in the next execution.

Installing the Java Agent on iSeries and AS/400

Some databases are installed on iSeries and AS400 machines. It is possible to install Oracle Data Integrator execution agent on these machines, in order to:

- execute the loading processes on AS/400,
- execute OS400 system commands,
- reduce the network flow if both sources and targets are on AS/400.

The installation procedure below is designed to implement an Oracle Data Integrator execution agent on an AS/400 machine.

Preparing the System

Technical Pre-requisites

- AS/400 AS/400 V5R1, or V4R4M0, with the following PTF:
 - SF61800
 - SF55849
 - SF54922
- Programs to install:
 - IBM Toolbox for Java
 - Java Virtual Machine 1.3.1
- TCP/IP Services configured and launched on the AS/400 machine

Components

The Java/JDBC components used are:

- The Oracle Data Integrator Execution Agent
- The Java Virtual Machine 1.3.1
- The JDBC drivers for database access

Note about drivers: To connect DB2/400 from an AS/400, you should consider using Driver Wrapper for AS/400. See *Creating a DB2/400 Data Server* for more information.

Installing the Execution Agent

Installing the files

1. Create a directory tree on the AS/400 to store the Oracle Data Integrator Files:

```
MKDIR DIR('/odi')
MKDIR DIR('/odi/bin')
MKDIR DIR('/odi/lib')
MKDIR DIR('/odi/lib/scripting')
MKDIR DIR('/odi/drivers')
```

2. Transfer the files (using FTP, for instance) from the Oracle Data Integrator CD's /oracledi directory to the AS/400 directories created at step 1.

The files to copy are in:

```
/oracledi/bin/
/oracledi/lib/
/oracledi/lib/scripting (if you use Oracle Data Integrator Scripting)
/oracledi/drivers (only the drivers used need to be copied)
```

Creating the Java programs

It is necessary for performances reasons, on AS/400, to transform the Java packages (.class, .jar or .zip files) into **Java programs**.

To create a Java program from a .class, .jar or .zip file:

1. Launch the CL command:

```
CRTJVAPGM CLSF('<.class, .zip or .jar file location>')
OPTIMIZE(40)
```

For instance, to create a Java program from the odi.zip file (the Java agent), run:

```
CRTJVAPGM CLSF('/odi/lib/odi.zip') OPTIMIZE(40)
```

Note : Creating a Java program is a long operation, depending on the nature of the Java package. It is recommended you create programs only for the classes effectively used.

Launching the Execution Agent

You can launch the agent by two means:

- **Using the Shell interpreter: (QSH or STRQSH)** It is an OS/400 option implementing a UNIX-compatible shell. A number of UNIX commands are accessible (ls, chmod, chown, etc...) as well as the 'java' command. You may launch Oracle Data Integrator using the standard syntax for the from the Unix scripts (.sh extension) provided with Oracle Data Integrator in /bin. You must configure the odiparams.sh file before running the agent. The graphical modules (Topology, Designer, ...) are usable on AS/400.
- **Using OS/400 Commands (CL):** The RUNJVA or JAVA CL commands execute a Java application. It is convenient to use a CL program in order to launch the agent or the scenarios. You will find program template for these operations below.

Note about JVM version: When multiple Java machines are installed on the AS/400, it may be necessary to force the version of Java used in the Java command.

- For **QSH**: The flag `-Djava.version=<java version>` (example `-Djava.version=1.3.1`) of the Java command is used to force the version.
 - For the **OS/400 commands**: You pass the property as a JAVA command parameter: `-PROP((<property> <value>) (<property> <value>))`. For instance `-PROP((java.version 1.3.1))`

Note about Agent Naming: On AS/400, it is frequently required to explicitly name the agent when running it as a listener. You must therefore use the agent flag `-name=<agent name>` when running the agent.

Running the agent

```

          PGM          PARM(&NAME &PORT &VERB)
/* Command AGENT */
/* Parameters: */
/*   &NAME: physical name of the agent */
/*   &PORT: port number */
/*   &VERB: verbose mode -V=[1..5] */
/* Example of call: */
/*   CALL PGM(<myLib/myPGM>) PARM('-NAME=myAgt' '-PORT=20910' '-V=5') */
          DCL          VAR(&NAME) TYPE(*CHAR) LEN(128)
          DCL          VAR(&PORT) TYPE(*CHAR) LEN(30)
          DCL          VAR(&VERB) TYPE(*CHAR) LEN(30)
/* All classes below should be compiled */
/* with the CRTJVAPGM command */
/* with optimize 40. */
          DCL          VAR(&PROJ) TYPE(*CHAR) LEN(512) +
                      VALUE('/odi/lib/odi.zip:+
                          /odi/lib/sunjce_provider.jar:+
                          /odi/lib/commons-net.jar:+
                          /odi/lib/local_policy.jar:+
                          /odi/lib/jakarta-ant-optional.jar:+
                          /odi/lib/US_export_policy.jar:+
                          /odi/lib/jce1_2_2.jar')
/* Replace the drivers below with your own drivers. */
          DCL          VAR(&JDBC) TYPE(*CHAR) LEN(512) +
                      VALUE('/odi/drivers/jt400Native.jar:+
                          /odi/drivers/snpsdb2.jar:+
                          /odi/drivers/ojdbc14.jar')
/* Build the Java CLASSPATH */
          DCL          VAR(&PATH) TYPE(*CHAR) LEN(1024)
          CHGVAR       &PATH (&PROJ *tcat ':' *tcat &JDBC)
/* Start the Agent */
          SBMJOB       CMD(JAVA CLASS(oracle.odi.Agent)
CLASSPATH(&PATH) +
                      PARM('&NAME &PORT &VERB') +

```

```

                OPTIMIZE(40) +
                OUTPUT(*PRINT))
        ENDPGM

```

Running a scenario

```

                PGM                PARM(&SCEN &VERS &CTX &VERB)
/* Command STARTSCEN */
/* Parameters: */
/*   &SCEN: scenario name */
/*   &VERS: scenario version */
/*   &CTX: context */
/*   &VERB: verbose mode -V=[1..5] */
/* Example of call: */
/*   CALL PGM(<myLib/myPGM>) PARM('myScen' 'myVers' 'GLOBAL' '-V=5') */
                DCL                VAR(&SCEN) TYPE(*CHAR) LEN(30)
                DCL                VAR(&VERS) TYPE(*CHAR) LEN(30)
                DCL                VAR(&CTX) TYPE(*CHAR) LEN(30)
                DCL                VAR(&VERB) TYPE(*CHAR) LEN(30)
/* All classes below should be compiled */
/* with the CRTJVAPGM command */
/* with optimize 40. */
                DCL                VAR(&PROJ) TYPE(*CHAR) LEN(512) +
                VALUE('/odi/lib/odi.zip:+
                /odi/lib/sunjce_provider.jar:+
                /odi/lib/commons-net.jar:+
                /odi/lib/local_policy.jar:+
                /odi/lib/jakarta-ant-optional.jar:+
                /odi/lib/US_export_policy.jar:+
                /odi/lib/jce1_2_2.jar')
/* Replace the drivers below with your own drivers. */
                DCL                VAR(&JDBC) TYPE(*CHAR) LEN(512) +
                VALUE('/odi/drivers/jt400Native.jar:+
                /odi/drivers/snpsdb2.jar:+
                /odi/drivers/ojdbc14.jar')
/* Adapt all parameters below to your environment before use. */
                DCL                VAR(&DRV) TYPE(*CHAR) LEN(128) +
                VALUE('-
SECU_DRIVER=com.ibm.as400.access.AS400JDBCdriver')
                DCL                VAR(&URL) TYPE(*CHAR) LEN(128) +
                VALUE('-
SECU_URL=jdbc:as400://195.10.10.13;libraries=ODI')
                DCL                VAR(&USER) TYPE(*CHAR) LEN(30) +
                VALUE('-SECU_USER=QSECOFR')

```

```
DCL          VAR(&PASS) TYPE(*CHAR) LEN(128) +
              VALUE('-SECU_PASS=XYZ')
DCL          VAR(&WREP) TYPE(*CHAR) LEN(30) +
              VALUE('-WORK_REPOSITORY=WORKREP1')
DCL          VAR(&SUSER) TYPE(*CHAR) LEN(30) +
              VALUE('-ODI_USER=SUPERVISOR')
DCL          VAR(&SPASS) TYPE(*CHAR) LEN(128) +
              VALUE('-ODI_PASS=XYZ')
DCL          VAR(&PATH) TYPE(*CHAR) LEN(1024)
/* Build the Java CLASSPATH */
DCL          VAR(&PATH) TYPE(*CHAR) LEN(1024)
CHGVAR      &PATH (&PROJ *tcat ':' *tcat &JDBC)
/* Execute the Scenario */
SBMJOB      CMD(JAVA CLASS(oracle.odi.Agent)
CLASSPATH(&PATH) +
              PARM(&DRV &URL &USER &PASS &WREP &SUSER &SPASS
+
              SCEN &SCEN &VERS &CTX &VERB))
ENDPGM
```

Note: the passwords specified must be encrypted using the command `agent ENCODE <password>`.

Excel

Creating a Microsoft Excel Data Server

A Microsoft Excel **Data Server** corresponds to one Microsoft Excel spreadsheet that is accessible through your local network.

Pre-requisites

ODBC Configuration

Microsoft Excel spreadsheets can only be accessed through ODBC connectivity. A proper ODBC data source must be defined with the Microsoft ODBC Data Source Administrator.

Declaring a new ODBC Data Source

To declare a new ODBC data source, run the **Microsoft ODBC Data Source Administrator** from the **Configuration Panel**, then follow these steps:

1. Click the **Add** button to add a new Data source
2. Select the appropriate driver for Microsoft Excel: `Microsoft Excel Driver (*.xls)` and click **Finish**
3. Name this data source (this will be the "alias" that you need in Topology), and select the appropriate spreadsheet with the **Select Workbook...** button

4. In the **Options** uncheck the **Read Only** checkbox if you wish to write to the Excel spreadsheet
5. Click **OK**

Excel spreadsheet Configuration

A single Excel spreadsheet can contain several pages, and each page can contain several tables. Oracle Data Integrator should be able to locate each table, regardless of how the spreadsheet is organized. Oracle Data Integrator will identify the tables by their name. To name a table, you just have to follow these steps:

1. Open the spreadsheet in Microsoft Excel
2. Select all of the cells in the table
3. From the menu, select **Insert > Name > Define ...**
4. Enter a name for that table, and click **OK**. This name will appear as the datastore name when you reverse engineer the Excel Spreadsheet in Oracle Data Integrator.

Note: The first line of the table will correspond to the header line in which Oracle Data Integrator will fetch the column names during the reverse-engineering process.

Note: If you want to define an empty table, select only the first line in the naming process. Oracle Data Integrator will automatically add the new lines below this one when inserting data.

Creation of the Data Server

To create a Microsoft Excel Data Server:

1. Connect to **Topology Manager**
2. Select **Topology > Physical Architecture > Technologies > Microsoft Excel** in the tree
3. Right click and select **Insert 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.
 - **User/Password:** Not used here.
 - **Batch update :** 1
 - **Array Fetch :** 1
5. Fill in the following fields in the **JDBC** tab:
 - **JDBC Driver:** `sun.jdbc.odbc.JdbcOdbcDriver`
 - **JDBC URL:** `jdbc:odbc:<AliasName>`
where **<AliasName>** is the name of your ODBC data source.

Warning: to access a Microsoft Excel Spreadsheet via ODBC, you must first ensure that this spreadsheet is not currently open in a Microsoft Excel session. This can lead to unexpected results.

6. Click **Test**.
7. Click **Test** in the **Test Connection** windows.
8. A windows indicating a **Successful Connection** appears. Click **OK**. If the connection is not successful, refer to the Common Errors with Microsoft Excel.
9. Click **OK** to validate the creation of the data server


The creation window for the data server's first **Physical Schema** appears.
See [Create a Microsoft Excel Physical Schema](#)

Creating a Physical Schema for Microsoft Excel

A **Physical Schema** has no particular significance in Excel. Oracle Data Integrator needs only one physical schema for each Microsoft Excel **Data Server**.

To Create a Physical Schema for Microsoft Excel:

Note: If you have just created a Data Server, ignore step 1. The **Physical Schema** window should already be opened.

1. Select the Microsoft Excel **Data Server** then right-click and select **Insert Physical Schema**. The **Physical Schema** window appears
2. Check the **Default** box if you want this schema to be the default for this data server (The first physical schema is always the default). For more information, see [Physical Schema](#). Note that for Microsoft Excel, you do not need any other Physical Schemas
3. Go to the **Context** tab
4. Select a **Context** and an existing **Logical Schema** for this new **Physical Schema**, then go to step 7.
If no Microsoft Excel Logical Schema exists yet, proceed to step 5
5. Click the  button
6. Select an existing **Context** from the left column, then type the name of a **Logical Schema** into the right column. This Microsoft Excel Logical Schema is automatically created and associated with this physical schema in this context

Warning ! A **Logical Schema** can be associated with only one **Physical Schema** in a given **Context**

7. Click **OK**

Creating and Reverse-Engineering a Microsoft Excel Model

Creating a Microsoft Excel Model

A Microsoft Excel **Model** is a set of datastores that correspond to tables contained in a Microsoft Excel spreadsheet. A model is always based on a **Logical Schema**. In a given **Context**, the **Logical Schema** corresponds to one **Physical Schema**. This Physical Schema's corresponding Data Schema contains the tables for this Microsoft Excel model.

To create a Microsoft Excel Model:

1. Connect to **Designer**
2. Select **Models** in the tree
3. Right click and select **Insert Model**
4. In the **Definition** tab, fill in the **Name** field
5. In the **Technology** field, select **Microsoft Excel**
6. In the **Logical Schema** field, select the Logical Schema on which your model will be based

7. Go to the **Reverse** tab and select a **Context** that will be used when reverse-engineering the model. Click **Apply**.

The model is created but contains no datastore yet.

Reverse-Engineering a Microsoft Excel Model

A model is created with no datastores. The **Reverse-Engineering** operation recovers the structure of the model's tables to create the appropriate datastore definitions. There are two types of reverse-engineering: **Standard** reverse-engineering, which uses only the abilities of the driver, and **Customized** reverse-engineering, which uses a RKM (Reverse Knowledge Module) to get the structure of the objects.

Note: Out of the box, there is no RKM for Microsoft Excel. It is recommended that you use standard reverse-engineering.

Standard Reverse-Engineering

To perform a Standard Reverse-Engineering on Microsoft Excel:

1. Go to your Microsoft Excel **Model's Reverse** tab.
2. Fill in the following fields:
 - **Standard**
 - **Context:** Context used for the reverse-engineering
 - **Types of objects to reverse-engineer:** List of object types that should be taken into account by the reverse-engineering process.
3. Go to the **Selective Reverse** tab.
 - Check the **Selective Reverse**, **New Datastores**, and **Objects to Reverse** boxes
4. A list of datastores to be reverse-engineered appears. Uncheck those that you do not wish to reverse-engineer
5. Click **Reverse**, then **Yes** to validate the changes
6. A progress bar appears as Oracle Data Integrator starts reverse-engineering the selected datastores

The reverse-engineered datastores appear below the model.

It is possible to refine the reverse-engineering using the options in found in the **Reverse** and **Selective Reverse** tabs. Refer to Model for more information on this.

Choosing the Right KMs for Microsoft Excel

The choice of KMs in an interface determines the abilities and performances of this interface or check. The recommendations below help choose the right KM according to specific situations with a Microsoft Excel Server.

For generic information about the KM, see Knowledge Modules

Note: Only knowledge modules imported into a project are available for use by the project's interfaces. To import a KM, see Import a KM.

Note: It is possible, for simple Excel spreadsheets, to use the File access using CSV (comma delimited) file format, this method does not use ODBC access, and can manage voluminous files faster.

Loading data to and from Microsoft Excel

Microsoft Excel can be used as an interface's source or target (Not as a staging area). The choice of the LKM (in the Interface Flow tab) used to load data between Microsoft Excel and another type of data server is essential for the execution of the interfaces.

LKM Choice when data is coming from Microsoft Excel

LKM recommendations in the following cases:

- Microsoft Excel Source to a Staging Area

If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Target or Staging Area Technology	Recommended KM	Notes
Sybase	LKM ISO SQL to Sybase (bcp)	Faster than the Generic LKM (Uses Bulk Loading)
Microsoft SQL Server	LKM ISO SQL to MSSQL (bulk)	Faster than the Generic LKM (Uses Bulk Loading)
Oracle	LKM ISO SQL to Oracle	Faster than the Generic LKM (Uses Statistics)
All	LKM ISO SQL to SQL	Generic KM

Integrating data into Microsoft Excel

IKM choice for Microsoft Excel

IKM recommendations in the following cases:

- Staging Area to Microsoft Excel Target.

The Staging Area cannot be Microsoft Excel. **Staging Area Different From Target** option must be checked.

Mode	Target	Recommended KM	Notes
Append		IKM ISO SQL to SQL Append	No flow control. Static control is not possible in Excel.

Common Errors with Microsoft Excel

Decoding the error messages

Errors appear often in Oracle Data Integrator in the following way:

```
java.sql.SQLException: java.sql.SQLException: [Microsoft][ODBC Driver
Manager] Data source name not found and no default driver specified
RC=0x1b
at ...
...
```

the `java.sql.SQLException` code simply indicates that a query was made through the JDBC-ODBC bridge, which has returned an error. This error is frequently a database or driver error, and must be interpreted in this direction.

Only the part of text in bold must first be taken in account. It must be searched in the ODBC driver or Excel documentation. If its contains a specific error code, like here in red, the error can be immediately identified.

If such an error is identified in the execution log, it is necessary to analyze the SQL code to find the source of the error. The code is displayed in the **description** tab of the **task** in error.

The most common errors with Excel are detailed below, with their principal causes.

Common Errors

Connection Errors

UnknownDriverException

The JDBC driver is incorrect. Check the name of the driver.

*[Microsoft][ODBC Driver Manager] Data source name not found and no default driver specified
RC=0xb*

Datasource not found or driver name not specified

The ODBC Datasource specified in the JDBC URL is incorrect.

Errors in the interfaces

The Microsoft Jet Database engine could not find the object <object name>

The table you are trying to access does not exist or is not defined in the Excel spreadsheet.

Too few parameters. Expected 1.

You are trying to access a nonexisting column in the Excel spreadsheet.

Operation must use an updateable query.

This error is probably due to the fact that you have not unchecked the "read only" option when defined the Excel DSN. Uncheck this option and re-execute your interface.

File

Creating a File Data Server

A File **Data Server** is a container for a set of file folders (each file folder corresponding to a physical schema).

The `FILE_GENERIC` data server provided as default in Topology suits most of the needs. In most cases, it is not required to create a File Data Server, and you only need to Create a Physical Schema for a File.

Pre-requisites

JDBC Driver

In terms of performance, it is always better to use database utilities when handling flat files. Oracle Data Integrator includes knowledge modules using these utilities. However, for your convenience, Oracle Data Integrator provides a Type 4 JDBC Driver for Flat Files. This driver supports both ASCII and EBCDIC (Legacy) file formats.

It is also possible to connect to flat files through ODBC. This method has poor performance, and does not support reverse engineering as well as some other features. It is therefore not recommended.

JDBC driver for Flat Files

This driver is embedded with the product and comes free of charge with Oracle Data Integrator. It is installed when you install Oracle Data Integrator and does not require any further configuration.

Database utilities

Most databases will have their own utilities for interacting with flat files. All require that the database client software be accessible from the Agent or from the Oracle Data Integrator installation directory. Some examples are:

- Oracle: SQL*Loader
- Sybase: bcp
- Microsoft SQL Server: bcp
- Teradata: fastload and multiload

All utilities can be used directly from Oracle Data Integrator. Please refer to your database documentation for information on the proper installation of these utilities.

Creation of the Data Server

To create a File Data Server:

1. Connect to **Topology Manager**
2. Select **Topology > Physical Architecture > Technologies > File** from the tree
3. Right click and select **Insert Data Server**
4. Fill in the following fields in the **Definition** tab:
 - **Name:** Name of the Data Server as it will appear in Oracle Data Integrator
 - **User/Password:** Not used.
5. Fill in the following fields in the **JDBC** tab, according to the driver used:
 - **JDBC Driver:** `com.sunopsis.jdbc.driver.file.FileDriver`
 - **JDBC URL:** `jdbc:snps:dbfile`
6. Click **Test**
7. Click **Test** in the **Test Connection** window
8. A window indicating a **Successful Connection** should appear. Click **OK**
9. Click **OK** to accept the creation of the data server

The creation window for the first **Physical Schema** for this data server will appear. See *Creating a Physical Schema for a File*

Creating a Physical Schema for a File

A **Physical Schema** corresponds to a pair of directories :

- A (Data) **Schema**, in which Oracle Data Integrator will look for the source and target files for interfaces.
- A **Work Schema** in which Oracle Data Integrator will eventually create in order to manipulate temporary files associated to the sources and targets contained in the Data Library.

Note: Data and Work schemas each describe a directory. This directory must be accessible to the agent that will be used to run the transformations. The directory can be an absolute path (m:/public/data/files) or relative to the agent startup directory (../demo/files). It is strongly advised to use a UNC (independent from the execution location) for the path. When running the transformations with "no agent", the directory is relative to the directory where Oracle Data Integrator has been installed.


Note: In UNIX in particular, the agent must have read/write permission on these directories (even to access a read only file), as it will create error files for invalid file records.

Note: Keep in mind that file paths are different in Windows than they are in UNIX. Take the platform used by the agent into account when setting up this information.

Creation of the Physical Schema

To Create a File Physical Schema:

Note: If you have just created a Data Server, then ignore step 1. The **Physical Schema** window should already be opened.

1. In **Topology**, select the File **Data Server** that you wish to create a physical schema for, right-click and select **Insert Physical Schema**. The **Physical Schema** window will appear
2. Type in the path to the directory that contains your source or target files in **Directory (Schema)**
3. Type the Work directory for this physical schema into **Directory (Work Schema)**
4. 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). For more information, see Physical Schema
5. Go to the **Context** tab.
6. Select a **Context** and an existing **Logical Schema** for this new **Physical Schema**, then go to step 8.
If no File Logical Schema exists yet, proceed to step 7
7. Click the  button
8. Select an existing **Context** from the left column, then type the name of a **Logical Schema** in the right column. This File Logical Schema is automatically created and associated to this physical schema in this context

Warning ! A **Logical Schema** can be associated with only one **Physical Schema** in a given **Context**

9. Click **OK**

Creating and Reverse-Engineering a File Model

Create a File Model

An File **Model** is a set of **Datastores**, corresponding to files stored in a directory. A model is always based on a **Logical Schema**. In a given **Context**, the **Logical Schema** corresponds to one **Physical Schema**. The Data Schema of this Physical Schema is the directory containing all the Files (eventually in sub-directories) described in the model.

To create a File Model:

1. Connect to **Designer**
2. Select **Models** in the tree
3. Right-click then select **Insert Model**
4. In the **Definition** tab, fill in the **Name** field
5. In the **Technology** field, select **File**
6. In the **Logical Schema** field, select the Logical Schema on which your model will be based
7. Go to the **Reverse** tab and select a **Context** which will be used for the model's reverse-engineering. Click **Apply**

The model is created, but contains no datastores yet.

Reverse-engineer a File model

A model is created with no datastore. The **Reverse-engineering** operation consists in gathering the structure of the files in the model to create the model's datastore definitions.

There are three types of reverse-engineering:

- **Standard** reverse-engineering, which is only available for delimited files. It is performed per file.
- **Customized** reverse-engineering, which uses a RKM (Reverse Knowledge Module) to get the structure of all of the files of the model from a Microsoft Excel Spreadsheet.
- **COBOL Copybook** reverse-engineering, which is available for fixed files, if a copybook describing the file is provided. See Reverse a COBOL Copybook for more information.
- **Column Setup Wizard**, which is available for fixed files. See Reverse-engineering a Fixed File using the Wizard for more information.

Note: A specific RKM, the `RKM File from Excel`, is provided for File customized reverse engineering. This RKM must be imported into at least one project to be available here.

Standard Reverse-Engineering

To perform a Standard Reverse-Engineering for a file (delimited files only):

1. Right click on your File **Model** and select **Insert Datastore**
2. In the **Definition** Tab, fill in the following fields:
 - **Name:** name of this datastore
 - **Resource Name:** sub-directory (if needed) and name of the file. You can retrieve the file name with the (...) button
3. Go to the **Files** tab to describe the type of file
 - Format must be **Delimited**
 - Specify the number of lines for the **Header**. (If there is a header, the first line of the header will be used by Oracle Data Integrator to name the columns in the file)
 - Select a **Record Separator**

- Select or type the character used as a **Field Separator**
 - Enter a **Text delimiter** if your file uses one
 - Enter a **Decimal separator** if your file has decimals
4. Click **Apply** to save the definition of the file.
 5. Go to the **Columns** tab to reverse engineer the file structure
 - Click on the **Reverse** button
 - Check the format and length for the reverse engineered columns. Oracle Data Integrator will try to guess the types and lengths, but might also use some default values (usually 50 for strings).
 - Click **Apply** or **OK** to save the column descriptions

Customized Reverse-Engineering

For this reverse-engineering procedure, a Microsoft Excel Spreadsheet contains the description of the group of files. As an example, edit the `file_repository.xls` file found in the Oracle Data Integrator `/demo/excel` sub-directory. The following steps assume that you have modified this file with the description of the structure of your flat files.

To perform a customized reverse-engineering, you will need to perform the following steps:

1. **Add an ODBC Microsoft Excel Datasource** corresponding to the Excel Spreadsheet containing the files description.
2. **Define a Data Server, a Physical and a Logical Schema** to this spreadsheet
3. **Run the Customized Reverse-engineering** using the RKM File from Excel RKM.

To Add an ODBC Microsoft Excel datasource Driver (.xls):*

1. Launch the **Microsoft ODBC Administrator**
2. Add a **System Datasource**
3. Select the driver: `Microsoft Excel Driver (*.xls)`
4. Name the data source: **SUNOPSIS_XL_FILE_REPO** and select the file `/demo/excel/file_repository.xls`


To Define the Data Server, Physical and Logical Schema to the Microsoft Excel Spreadsheet:

1. Launch the **Topology Manager** module
2. Add a Microsoft Excel **Data Server** with the following parameters:

Name: EXCEL_FILE_REPOSITORY

JDBC Driver: `sun.jdbc.odbc.JdbcOdbcDriver`

JDBC URL: `jdbc:odbc:SUNOPSIS_XL_FILE_REPO`

- 1.
3. **Apply** the modifications
4. Add a **Logical Schema** on the default physical schema
5. In the **Context** tab of the physical schema, click 
6. In the new line, select the context for the reverse engineering and type in the second column `EXCEL_FILE_REPOSITORY`. This name is mandatory.
7. **Apply** the modifications

To run the customized reverse-engineering:

1. Open the **Designer** module
2. Import the **RKM File From Excel** Knowledge Module into at least one project
3. Click on the File **Model**, right click then select **Edit**.
4. In the **Reverse** Tab, set the following parameters:
 - Select **Customized**
 - **Context:** Reverse Context
 - **KM:** RKM File from Excel
5. Click **Reverse**
6. You can follow the reverse-engineering process in the **Execution Log**

Important: A Microsoft Excel logical schema must be defined. It must be named `EXCEL_FILE_REPOSITORY` and point to the file `file_repository.xls` or another file with a similar structure

Important: The Microsoft Excel file `file_repository.xls` should be closed before running the reverse engineering.

Choosing the Right KMs for Files

The KM choice for an interface or a check determines the abilities and performance of this interface or check. The recommendations below help in the selection of the KM for different situations concerning a File Server.

For generic information about the KM, see Knowledge Modules

Note: Only knowledge modules imported into a project are available for this project's interfaces. To import a KM, see Import a KM.

Load data to and from Files

Files can be used as a source or target in an interface, **NOT** as a staging area. The LKM choice in the Interface Flow tab is essential in determining the interface's performance.

LKM Choice from File

Recommendations for LKM in the following cases:

- File Source to a Staging Area

If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Target or Staging Area Technology	Recommended KM	Notes
Oracle	LKM File to Oracle (sql*loader)	Faster than the Generic LKM (Uses sql*loader)
All	LKM ISO File to SQL	Generic KM

Integrate data in File

IKM choice for File

Recommendations for LKM in the following cases:

- Staging Area to File Target

Recommendations for the KM to perform an integration into a File, depending on the integration mode chosen. If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Mode	Target	Recommended KM	Notes
Append	File	IKM ISO SQL to File Append	No flow control.

Notes: If you want to create the target file without deleting it if it exists (TRUNCATE option), you must insert the following step in this KM:

- **Technology:** File

- **Command:** CREATE TABLE

- Select **Ignore Error** or create an option that can be trigger the file creation in the Interface Flow tab.

Reverse-engineering a COBOL Copybook

COBOL Copybook reverse-engineering allows you to retrieve a legacy file structure from its description contained in a COBOL Copybook file in Data Integrator.

To reverse-engineer a COBOL Copybook:

1. Create or open a file datastore that has a **fixed** format
2. Go to the **column** tab
3. Click the **Reverse COBOL Copybook** button
4. Fill in the following fields:
 - **File:** Location of the Copybook file
 - **Character set:** Copybook file charset
 - **Description format (EBCDIC | ASCII):** description (copybook) file format
 - **Data format (EBCDIC | ASCII):** data file format
5. Click **OK**

The columns described in the Copybook are reverse-engineered and appear in the column list.

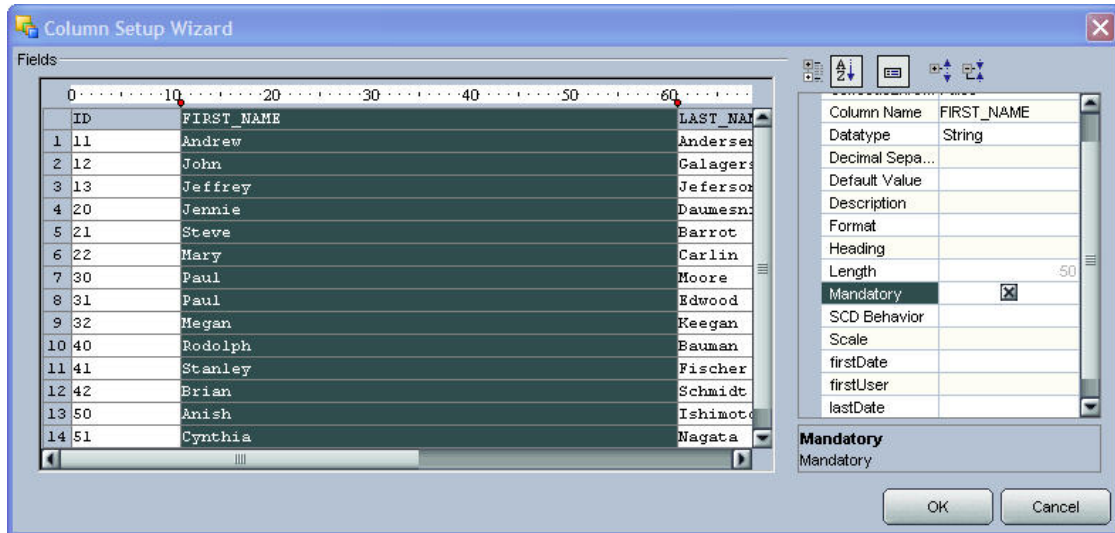
Note: Columns for which data type is not declared in Oracle Data Integrator appear with no data type.

Reverse-engineering a Fixed File using the Wizard

Oracle Data Integrator provides a wizard to graphically define the columns of a fixed file.

To reverse-engineer a fixed file using the Wizard:

1. Create or open a file datastore that has a **fixed** format.
2. Go to the **column** tab
3. Click the **Reverse** button. A window as shown below. This window displays the first records of your file.



4. Click on the ruler (above the file contents) to create markers delimiting the columns. You can right-click in the ruler to delete a marker.
5. Columns are created with pre-generated names (C1, C2, and so on). You can edit the column name by clicking in the column header line (below the ruler).
6. In the properties panel (on the right), you can edit all the parameters of the selected column. You should set at least the **Column Name**, **Datatype** and **Length** for each column.
7. Click **OK** when the columns definition is complete.

JMS

Creating a JMS Data Server

An **JMS data server** corresponds to one JMS provider/router that is accessible through your local network.

It exists two types of JMS data servers: **JMS Queue** and **JMS Topic**.

- The **JMS Queue** data server establishes a connection to a **Queue** in the JMS router
- The **JMS Topic** data server establishes a connection to a **Topic** in the JMS router

The messages are accessed through the **JMS router** (via the **JNDI directory service**).

The **Oracle Data Integrator JMS driver** loads the messages.

Pre-requisites

JNDI Configuration

The JNDI configuration depends on the chosen JMS router. Please refer to the JMS router specific documentation for details on the JNDI configuration.

Creation of the Data Server

To create a JMS Queue or JMS Topic Data Server:

1. Connect to **Topology Manager**
2. Select **Topology > Physical Architecture > Technologies > JMS Queue** (or **Topic**) in the tree view
3. Right click and select **Insert Data Server**
4. Fill in the following fields in the **Definition** tab:
 - **Name:** Name of the Data Server as it will appear in Oracle Data Integrator.
 - **User/Password:** Not used here. Leave these fields empty.
5. Fill in the following fields in the **JNDI** tab:
 - **JNDI Authentication:** Leave this field <None>.
 - **JNDI User:** Enter the username to connect to the JNDI directory (not mandatory).
 - **Password:** This user's password (not mandatory).
 - **JNDI Protocol:** Select from the list the JNDI protocol (not mandatory).
 - **JNDI Driver:** Name of the Java class to connect to the JNDI resource, for example `com.sun.jndi.ldap.LdapCtxFactory` for LDAP
 - **JNDI URL:** <JMS_RESOURCE>, for example `ldap://<host>:<port>/<dn>` for LDAP
 - **JNDI Resource:** Logical name of the JNDI resource corresponding to your JMS Queue (or Topic) connection.
Specify `QueueConnectionFactory` if you want to access a message queue and `TopicConnectionFactory` if you want to access a topic. Note that these parameters are specific to the JNDI directory and the provider.
6. Click **Test**.
7. Click **Test** in the **Test Connection** window.
8. A windows indicating a **Successful Connection** should appear. Click **OK**.
9. Click **OK** to validate the creation of the data server

The creation window for the data server's first **Physical Schema** appears.
See [Creating a JMS Physical Schema](#).

Choosing the Right KMs for JMS

The choice of KMs in an interface determines the interface's abilities and performance. The recommendations below help choose the right KM for integrating or loading JMS messages .

For generic information about the KM, see [Knowledge Modules](#)

Note: Only knowledge modules imported into a project are available for use in the project's interfaces. To import a KM, see [Import a KM](#).

Loading data from a MOM

JMS can be used as a source or a target in an interface. Data from a JMS message Queue or Topic can be loaded to any SQL compliant database used as a staging area. The choice of a KM in the **Interface Flow** tab is essential in determining an interface's performance.

Choosing a LKM for JMS

Recommendations for LKM in the following cases:

- JMS Source to a Staging Area

Target or Staging Area Technology	Recommended KM	Notes
All	LKM JMS to SQL	Generic KM.

Checking data in a MOM

It is not possible to perform data validation on a JMS Topic or Queue data server.

Integrating data in a MOM

Choosing a IKM for JMS

Mode	Target	Recommended KM	Notes
Append	JMS	IKM SQL to JMS Append	No flow control

JMS Specific KM Options

The JMS specific options are detailed below.

Option	Used to	Description
PUBLISH	Write	Check this option if you want to publish new messages in the destination. This option is set to Yes by default.
JMS_COMMIT	Write	Commit the publication. Uncheck this option if you don't want to commit your publication on your router. This option is set to Yes by default.
JMSDELIVERYMODE	Write	JMS delivery mode (1: Non Persistent, 2: Persistent). A persistent message remains on the server and is recovered on server crash.
JMSEXPIRATION	Write	Expiration delay in milliseconds for the message on the server [0..4 000 000 000]. 0 signifies that the message never expires. Warning! After this delay, a message is considered as expired, and is no longer available in the topic or queue. When developing interfaces it is advised to set this parameter to zero.
JMSPRIORITY	Write	Relative Priority of the message: 0 (lowest) to 9 (highest).
SENDMESSAGE TYPE	Write	Type of message to send (1 -> BytesMessage, 2 -

		>TextMessage).
JMSTYPE	Write	Optional name of the message.
CLIENTID	Read	Subscriber identification string. This option is described only for JMS compatibility. Not used for publication.
DURABLE	Read	D: Session is durable. Indicates that the subscriber definition remains on the router after disconnection.
MESSAGEMAXNUMBER	Read	Maximum number of messages retrieved [0 .. 4 000 000 000]. 0: All messages are retrieved.
MESSAGETIMEOUT	Read	Time to wait for the first message in milliseconds [0 .. 4 000 000 000]. if MESSAGETIMEOUT is equal to 0, then there is no timeout. MESSAGETIMEOUT and MESSAGEMAXNUMBER cannot be both equal to zero. if MESSAGETIMEOUT= 0 and MESSAGEMAXNUMBER =0, then MESSAGETIMEOUT takes the value 1. Warning! An interface may retrieve no message if this timeout value is too small.
NEXTMESSAGETIMEOUT	Read	Time to wait for each subsequent message in milliseconds [0 .. 4 000 000 000]. The default value is 1000. Warning! An interface may retrieve only part of the messages available in the topic or the queue if this value is too small.
MESSAGESELECTOR	Read	Message selector in ISO SQL syntax. See Using JMS Properties for more information on message selectors.


Creating a Physical Schema for JMS

The **Physical Schema** will be a storage location for the tables associated with the Queues or Topics.

To Create a Physical Schema for JMS:

Note: If you have just created a Data Server, ignore step 1. The **Physical Schema** window should already be opened.

1. In **Topology Manager**, select the appropriate JMS Queue/Topic **Data Server** for which you want to create a physical schema, right-click and select **Insert Physical Schema**. The **Physical Schema** window will appear.
2. Check the **Default** box if you want this schema to be the default for this data server (The first physical schema is always the default). For more information, see Physical Schema.

3. Go to the **Context** tab.
4. Select a **Context** and an existing **Logical Schema** for this new **Physical Schema**, then go to step 8.
If no Logical Schema exists yet, proceed to step 6.
5. Click the  button.
6. Select an existing **Context** from the left column, then type the name of a **Logical Schema** into the right column. This Logical Schema is automatically created and associated with this physical schema in this context

Warning ! A **Logical Schema** can be associated with only one **Physical Schema** in a given **Context**

8. Click **OK**

Note: Only one physical schema is required per JMS data server.

Defining a JMS Model

Creating a JMS Model

A **JMS Model** is a set of datastores corresponding to the Topics or Queues of a router. A model is always based on a **Logical Schema**. In a given **Context**, the **Logical Schema** corresponds to one **JMS Physical Schema**. The Data Schema corresponding to this Physical Schema contains the Topics or Queues.

To create a JMS Model:

1. Connect to **Designer**
2. Select **Models** in the tree view
3. Right-click then select **Insert Model**
4. In the **Definition** tab, fill in the **Name** field
5. In the **Technology** field, select **JMS Topic** or **JMS Queue**
6. In the **Logical Schema** field, select the Logical Schema on which your model will be based
7. Go to the **Reverse** tab and select a **Context**. Click **Apply**.

Note: It is not possible to reverse-engineer a JMS model. To create a datastore please refer to the Defining the JMS Datastores section.

The model is created with no datastores.

Defining the JMS Datastores

In Oracle Data Integrator, each **Datastore** is a JMS Topic or Queue. Each message in this topic or queue is a **row** of the datastore. A JMS message may carry any type of information and there is no meta-data retrieval method available. To define the Datastore formats, it is possible to either:

- create the datastore as a file datastore and manually declare the message structures,
- use the File reverse engineering through an Excel spreadsheet in order to automate the reverse engineering of messages,
- or duplicate a datastore from another model into the JMS model (drag and drop with the CTRL key pressed).

Important: The datastores' resource names must be identical to the name of JMS destinations (this is the logical JNDI name) that will carry the message corresponding to their data.

Declaring the Property Pseudo-Columns

The property pseudo-columns represent properties or header fields of a message. These pseudo-columns are defined in the Oracle Data Integrator model as columns in the JMS datastore, with JMS-specific datatypes. The JMS-specific datatypes are called `JMS_xxx` (for example: `JMS String`, `JMS Long`, `JMS Int`, etc).

To define these property pseudo-columns, simply declare additional columns named identically to the properties and specified with the appropriate JMS-specific datatypes. If you define pseudo-columns that are not listed in the standard set of JMS Properties, then they are considered as application-specific properties.

Warning: Property pseudo-columns must be defined and positioned in the JMS datastore after the columns making up the message payload. Use the **Order** field in the column definition to position these columns. The order of the pseudo-columns themselves is not important as long as they appear at the end of the datastore definition.

Warning: Pseudo-columns names are case-sensitive.

JMS Standard Properties

The JMS properties contained in the message header are described below.

Using the Properties

In Oracle Data Integrator, pseudo-columns corresponding to properties should be declared in accordance with the description provided below. The JMS type and access mode columns refer to the use of these properties in Oracle Data Integrator or in Java programs. In Oracle Data Integrator, some of these properties are used through the IKM options, and the pseudo-column values should not be set by the interfaces.

For more details on using these properties in a Java program, see <http://java.sun.com/products/jms/>.

Message Headers

These fields are defined in the JMS standard.

Property	JMS Type	Access (Read/Write)	Description
<code>JMSDestination</code>	JMS String	R	Name of the destination (topic or queue) of the message.
<code>JMSDeliveryMode</code>	JMS Integer	R/W (set by IKM option)	Distribution mode: 1 = Not Persistent or 2 = Persistent. A persistent message is never lost, even if a router crashes. When sending messages, this property is set by the <code>JMSDELIVERYMODE KM</code> option.

JMSMessageID	JMS String	R	Unique Identifier for a message. This identifier is used internally by the router.
JMSTimestamp	JMS Long	R	Date and time of the message sending operation. This time is stored in a UTC standard format ⁽¹⁾ .
JMSExpiration	JMS Long	R/W (set by IKM option)	Message expiration date and time. This time is stored in a UTC standard format ⁽¹⁾ . To set this property the JMSEXPIRATION KM option must be used.
JMSRedelivered	JMS Boolean	R	Indicates if the message was resent. This occurs when a message consumer fails to acknowledge the message reception.
JMSPriority	JMS Int	R/W (set by IKM option)	Relative priority for the message: 0 (low) to 9 (high). To set this property the JMSPRIORITY KM option must be used.
JMSReplyTo	JMS String	R/W	Name of the destination (topic or queue) the message replies should be sent to.
JMSCorrelationID	JMS String	R/W	Correlation ID for the message. This may be the JMSMessageID of the message this message generating this reply. It may also be an application-specific identifier.
JMSType	JMS String	R/W (set by IKM option)	Message type label. This type is a string value describing the message in a functional manner (for instance "SalesEvent", "SupportProblem"). To set this property the JMSTYPE KM option must be used.

Optional JMS-Defined Properties

The following fields are optional in the JMS standard.

Property	JMS Type	Access (Read/Write)	Description
JMSXUserID	JMS String	R	Client User ID.
JMSXAppID	JMS String	R	Client Application ID.

JMSXProducerTXID	JMS String	R	Transaction ID for the production session. This ID is the same for all the messages sent to a destination by a producer between two JMS commit operations.
JMSXConsumerTXID	JMS String	R	Transaction ID for current consumption session. This ID is the same of a batch of message read from a destination by a consumer between two JMS commit read operations.
JMSXRcvTimestamp	JMS Long	R	Message reception date and time. This time is stored in a UTC standard format ⁽¹⁾ .
JMSXDeliveryCount	JMS Int	R	Number of times a message is received. Always set to 1.
JMSXState	JMS Int	R	Message state. Always set to 2 (Ready).
JMSXGroupID	JMS String	R/W	ID of the group to which the message belongs.
JMSXGroupSeq	JMS Int	R/W	Sequence number of the message in the group of messages.

⁽¹⁾: The UTC (Universal Time Coordinated) standard is the number of milliseconds that have elapsed since January 1st, 1970

Using JMS Properties

In addition to their contents, messages have a set of **properties** attached to them. These may be JMS standard, provider-specific, or application-specific (user defined) properties.

JMS properties are used in Oracle Data Integrator as complementary information to the message, and are used, for example, to filter the messages.

Declaring Application-Specific Properties

When defining JMS message datastores, all pseudo-columns declared with a case-sensitive name not in the list of JMS standard properties, and with a JMS datatype is considered as an application specific property.

Filtering on the Router

With this type of filtering, the filter is specified when sending the JMS read query. Only messages matching the **message selector** filter are retrieved. The message selector is specified in Oracle Data Integrator using the `MESSAGE_SELECTOR` KM option

Warning: For topics, the message selector used by the subscriber is part of the subscription definition. Therefore, a filtered subscription will not store messages not matching the filter on the router.

The `MESSAGE_SELECTOR` is programmed in an SQL WHERE syntax. Comparison, boolean and mathematical operators are supported: `+`, `-`, `*`, `/`, `=`, `>`, `<`, `<>`, `>=`, `<=`, `OR`, `AND`, `BETWEEN`, `IN`, `NOT`, `LIKE`, `IS NULL`.

Note: Property names specified in the message selector are NOT case sensitive by default (for example `JMSType` may be specified in the clause indifferently in upper or lower case). In order to force the use of case sensitivity, field names should be enclosed by quotes.

Note: The `IS NULL` clause handles properties with an empty value but does not handle nonexistent application-specific properties. For example, if the selector `COLOR IS NULL` is defined, a message with the application-specific property `COLOR` specified with an empty value is consumed correctly. Another message in the same topic/queue without this property specified would raise an exception.

Examples

Filter all messages with priority greater than 5

```
JMSPriority > 5
```

Filter all messages with priority not less than 6 and with the type `Sales_Event`.

```
NOT JMSPriority < 6 AND JMSType = 'Sales_Event'
```

Filtering on the Client

Filtering is performed after receiving the messages, and is setup by creating a standard Oracle Data Integrator interface filter which must be executed on the staging area. A filter uses pseudo-columns from the source JMS datastore. The pseudo-columns defined in the Oracle Data Integrator datastore represent the JMS properties.

Using Property Values as Source Data

It is possible to use the values of JMS properties as source data in an interface. This is carried out by specifying the pseudo-columns of the source JMS datastore in the mapping.

Setting Properties when Sending a Message

When sending messages it is possible to specify JMS properties by mapping values of the pseudo-columns in an interface targeting a JMS datastore. Certain properties may be set using KM options. See JMS Standard Properties for more information.

JMS XML

Choosing the Right KMs for JMS XML

The choice of KMs in an interface determines the interface's abilities and performance. The recommendations below help choose the right KMs for different situations concerning the transport of XML messages with Oracle Data Integrator..

For generic information about the KM, see Knowledge Modules

Note: Only knowledge modules imported into a project are available for use in the project's interfaces. To import a KM, see Import a KM.

Loading XML Messages from a MOM

In order to load XML messages from a MOM (Message Oriented Middleware), the following steps must be followed:

- The first interface reading the XML message from the MOM must use the **LKM JMS XML to SQL** with the `SYNCHRO_JMS_TO_XML` **LKM option** set to Yes. This option creates and loads the XML schema from the message retrieved from the queue or topic.
- The last interface should commit the message consumption by setting the `JMS_COMMIT` to Yes.

Choosing a LKM for JMS XML

Recommendations for loading data from a JMS XML source to any ISO SQL staging area

Target or Staging Area Technology	Recommended KM	Notes
All	LKM JMS XML to SQL	Generic KM.

Checking XML Data in a MOM

It is not possible to perform data validation on a JMS XML Topic or Queue data server.

Integrating XML Messages into a MOM

In order to integrate XML data into a MOM, the following steps must be followed:

- The first interface loading the XML schema must provide a value for the `ROOT_TABLE` (it is the model's table that corresponds to the root element of the XML file), and also set the `INITIALIZE_XML_SCHEMA` option to Yes.

Note: The root table of the XML schema usually corresponds to the datastore at the top of the **hierarchy** tree view of the JMS XML model. Therefore the `ROOT_TABLE` parameter should take the value of the **resource name** for this datastore.

- The interfaces should load the datastores in the hierarchy order, starting by the top of the hierarchy and going down. The interfaces loading subsequent levels of the XML schema hierarchy should load the foreign key column linking the current hierarchy level to a higher one.
For example, when loading the second level of the hierarchy (the one under the root table), the foreign key column should be set to '0' (Zero), as it is the value that is set by the IKM in the root table primary key when the root table is initialized.
- The last interface should send the XML schema to the MOM by setting the `SYNCHRO_JMS_TO_XML` parameter to Yes.

Example

An XML file format generates a schema with the following hierarchy of datastores:

```
+ GEOGRAPHY_DIM (GEO_DIMPK, ...)
|
+--- COUNTRY (GEO_DIMFK, COUNTRYPK, COUNTRY_NAME, ...)
|
+--- REGION (COUNTRYFK, REGIONPK, REGION_NAME, ...)
```

In this hierarchy, GEOGRAPHY_DIM is the root table, and its GEOGRAPHY_DIMPK column is set to '0' at initialization time. The tables should be loaded in the GEOGRAPHY_DIM, COUNTRY, REGION sequence.

- When loading the second level of the XML hierarchy (COUNTRY) make sure that the FK field linking this level to the root table level is set to '0'. In the model above, when loading COUNTRY, we must load the COUNTRY.GEOGRAPHY_DIMFK set to '0'.
- You must also link the records of REGION to the COUNTRY level by loading the REGION.COUNTRYFK column with values that correspond to a parent record in COUNTRY (having REGION.COUNTRYFK = COUNTRY.COUNTRYPK).

For more information on loading data to XML schemas, see *Oracle Data Integrator XML Driver (JDBC edition) documentation*.

Choosing a IKM for JMS XML

Recommendations for integrating data to a JMS XML source from any ISO SQL staging area

Mode	Target	Recommended KM	Notes
Append	JMS	IKM SQL to JMS Append	No flow control

JMS XML Specific Options

The JMS specific KM options are detailed below. Options that are specific to XML message processing are indicated in bold.

Option	Used to	Description
CLIENTID	Read	Subscriber identification string. Not used for publication.
DURABLE	Read	D: Session is durable. Indicates that the subscriber definition remains on the router after disconnection.
MESSAGEMAXNUMBER	Read	Maximum number of messages retrieved [0 .. 4 000 000 000]. 0: All messages are retrieved.
MESSAGETIMEOUT	Read	Time to wait for the first message in milliseconds [0 .. 4 000 000 000]. if MESSAGETIMEOUT is equal to 0, then there is no timeout. MESSAGETIMEOUT and MESSAGEMAXNUMBER cannot be both equal to zero. if

		<p>MESSAGETIMEOUT= 0 and MESSAGEMAXNUMBER =0, then MESSAGETIMEOUT takes the value 1.</p> <p>Warning! An interface may retrieve no message if this timeout value is too small.</p>
NEXTMESSAGETIMEOUT	Read	<p>Time to wait for each subsequent message in milliseconds [0 .. 4 000 000 000]. The default value is 1000.</p> <p>Warning! An interface may retrieve only part of the messages available in the topic or the queue if this value is too small.</p>
MESSAGESELECTOR	Read	<p>Message selector in ISO SQL syntax. See Using JMS Properties for more information on message selectors.</p>
SENDMESSAGETYPE	Write	<p>Type of message to send (1 -> BytesMessage, 2 ->TextMessage).</p>
INITIALIZE_XML_SCHEMA	Write	<p>Initializes an empty XML schema. This option must be set to YES for the first interface loading the schema.</p>
JMSTYPE	Write	<p>Optional name of the message.</p>
JMSPRIORITY	Write	<p>Relative Priority of the message: 0 (lowest) to 9 (highest).</p>
JMSEXPIRATION	Write	<p>Expiration delay in milliseconds for the message on the server [0..4 000 000 000]. 0 signifies that the message never expires.</p> <p>Warning! After this delay, a message is considered as expired, and is no longer available in the topic or queue. It is advised to set this parameter to zero when developing interfaces.</p>
JMSDELIVERYMODE	Write	<p>JMS delivery mode (1: Non Persistent, 2: Persistent). A persistent message remains on the server and is recovered on server crash.</p>
ROOT_TABLE	Write	<p>Resource name of the datastore that is the root of the XML model hierarchy.</p>
SYNCHRO_XML_TO_JMS	Write	<p>Generates the XML message from the XML schema, and sends this message. This option must be set to YES for the last interface that writes to the schema XML.</p>

Creating a JMS XML Data Server

An JMS XML **data server** corresponds to one JMS provider/router that is accessible through your local network.

It exists two types of JMS XML data servers: **JMS Queue XML** and **JMS Topic XML**.

- The **JMS Queue XML** data server establishes a connection to a **Queue** in the JMS router that contains the XML content.
- The **JMS Topic XML** data server establishes a connection to a **Topic** in the JMS router that contains the XML content.

The XML content is accessed through the **JMS router** (via the **JNDI directory service**).

The **Oracle Data Integrator JMS driver** loads the messages that contain the XML content into the XML hierarchical structure in a relational structure in a **schema** stored in memory to enable SQL queries through JDBC. It is also able to unload the relational structure back to the JMS messages.

Pre-requisites

JNDI Configuration

The JNDI configuration depends on the chosen JMS router. Please refer to the JMS router specific documentation for details on the JNDI configuration.

XML Configuration

XML content is accessed through the **Oracle Data Integrator JDBC for XML driver**. The driver is installed with Oracle Data Integrator.

Connection Information

Ask your system administrator for the location of the **DTD file** associated with your XML content.

Creation of the Data Server

The creation process for a **JMS XML Queue** or **JMS Topic XML** data server is identical to the creation process for an XML data server except that you need to define a JNDI connection with JMS XML specific information in the JNDI URL. The details for creating a JMS Queue XML or JMS Topic XML data server are given in the following procedure.

To create a JMS Queue XML or JMS Topic XML Data Server:

1. Connect to **Topology Manager**
2. Select **Topology > Physical Architecture > Technologies > JMS Queue XML** or **JMS Topic XML** in the tree view
3. Right click and select **Insert Data Server**
4. Fill in the following fields in the **Definition** tab:
 - **Name:** Name of the Data Server as it will appear in Oracle Data Integrator.
 - **User/Password:** Not used here. Leave these fields empty.
5. Fill in the following fields in the **JNDI** tab, according to the driver used:
 - **JNDI Authentication:** Specify or select from the list the authentication mode.
 - **JNDI User:** Enter the username to connect to the JNDI directory (not mandatory).

- **Password:** This user's password (not mandatory).
- **JNDI Protocol:** Select from the list the JNDI protocol (not mandatory).
- **JNDI Driver:** Name of the Java class to connect to the JNDI resource, for example `com.sun.jndi.ldap.LdapCtxFactory` for LDAP.
- **JNDI URL:**
`<JMS_RESOURCE>?d=<DTD_FILE>&s=<SCHEMA>&JMS_DESTINATION=<JMS_DESTINATION_NAME>`. The JNDI URL properties are detailed in the table below.
- **JNDI Resource:** Logical name of the JNDI resource corresponding to your JMS Queue (or Topic) connection.
Specify `QueueConnectionFactory` if you want to access a message queue and `TopicConnectionFactory` if you want to access a topic. Note that these parameters are specific to the JNDI directory.

JNDI URL Properties:

Parameter	Value	Notes
d	<DTD File location>	DTD File location (relative or absolute) in UNC format. Use slash "/" in the path name and not backslash "\ in the file path. If this parameter is missing, the driver will build the name of the DTD file from the XML file, replacing the ".xml" extension with ".dtd". This parameter is mandatory.
re	<Root element>	Name of the element to take as the root table of the schema. This value is case sensitive. This parameter can be used for reverse-engineering a specific message definition from a WSDL file, or when several possible root elements exist in a XSD file.
ro	true false	If true, the XML file is opened in read only mode.
s	<schema name>	Name of the relational schema where the XML file will be loaded. If this parameter is missing, a schema name is automatically generated from the file name. This value must match the one set for the physical schema attached to this data server. This parameter is mandatory.
cs	true false	Load the XML file in case sensitive or insensitive mode. For case insensitive mode, all element names in the DTD file should be distinct (Ex: Abc and abc in the same file are banned). The case sensitive parameter is a permanent parameter for the schema. It CANNOT be changed after schema creation. Please note that when opening the XML file in insensitive mode, case will be preserved for the XML file.
JMSXML_ROWSEPARATOR	5B23245D	Hexadecimal code of the string used as a line separator (line break) for different XML contents. Default value is 5B23245D which corresponds to

		the string [#\$] .
JMS_DESTINATION	JNDI Queue name or Topic name	JNDI Name of the JMS Queue or Topic. This parameter is mandatory.

Example:

If using an **LDAP** directory as the JNDI provider, you should use the following parameters:

- **JNDI Driver:** `com.sun.jndi.ldap.LdapCtxFactory`
- **JNDI URL:**
`ldap://<ldap_host>:<port>/<dn>?d=<DTD_FILE>&f=<XML_FILE>&s=<SCHEMA>&JMS_DESTINATION=<JMS_DESTINATION_NAME>`
- **JNDI Resource:** `<Name of the connection factory>`

6. Click **Test**.
7. Click **Test** in the **Test Connection** window.
8. A windows indicating a **Successful Connection** should appear. Click **OK**.
9. Click **OK** to validate the creation of the data server


The creation window for the data server's first **Physical Schema** appears. See [Creating a JMS XML Physical Schema](#).

Creating a Physical Schema for JMS XML

The **Physical Schema** will be a storage location for the tables associated with the XML content.

To Create a Physical Schema for XML:

Note: If you have just created a data server, ignore step 1. The **Physical Schema** window should already be opened.

1. In **Topology Manager**, select the appropriate JMS Queue XML or JMS Topic XML **Data Server** for which you want to create a physical schema, right-click and select **Insert Physical Schema**. The **Physical Schema** window will appear.
2. Name the **Schema** and **Work Schema**. Use the **schema name** defined in the `s=<schema name>` property of the JNDI URL of the JMS Queue XML or JMS Topic XML **data server**.
3. Check the **Default** box if you want this schema to be the default for this data server (The first physical schema is always the default). For more information, see [Physical Schema](#). Note that for JMS Queue or Topic XML files, you do not need any other Physical Schemas.
4. Go to the Context tab
5. Select a Context and an existing Logical Schema for this new Physical Schema, then go to step 8.
If no JMS Queue or Topic XML Logical Schema exists yet, proceed to step 6.
6. Click the  button
7. Select an existing **Context** from the left column, then type the name of a **Logical Schema** into the right column. This JMS Queue or Topic XML Logical Schema is automatically created and associated with this physical schema in this context

Warning ! A **Logical Schema** can be associated with only one **Physical Schema** in a given **Context**

8. Click **OK**

Creating and Reverse-Engineering a JMS XML Model

Creating a JMS XML Model

A JMS Queue XML or JMS Topic XML **Model** will be seen as a set of datastores, with each datastore representing an entry level in the XML file. The models contain datastores describing the structure of the JMS messages. A model contains the message structure of one topic or one queue. This model's structure is reverse-engineered from the DTD or the XML file specified in the data server definition, using standard reverse-engineering.

A model is always based on a **Logical Schema**. In a given **Context**, the **Logical Schema** corresponds to one **Physical Schema**. This Physical Schema's corresponding Data Schema contains the tables for this JMS Queue or Topic XML model.

To create a Model for a JMS Queue or Topic XML file:

1. Connect to **Designer**
2. Select **Models** in the tree
3. Right click and select **Insert Model** from the menu
4. In the **Definition** tab, fill in the **Name** field
5. In the **Technology** field, select **JMS Queue XML** or **JMS Topic XML**.
6. In the **Logical Schema** field, select the Logical Schema on which your model will be based.
7. Go to the **Reverse** tab and select the **Context** that will be used when reverse-engineering the model. Click **Apply**.

The model is created but does not contain any datastores yet.

Reverse-Engineering a JMS Queue or Topic XML Model

A model is created with no datastores. The **Reverse-Engineering** operation gathers the structure of the model's tables to create the appropriate datastore definitions. It is recommended to use **standard reverse-engineering** for JMS Queue or Topic XML models, which uses the abilities of the XML driver.

Standard Reverse-Engineering

To perform a Standard Reverse-Engineering on a JMS Queue or Topic XML file:

1. Go to your JMS Queue or Topic XML **Model's Reverse** tab
2. Fill in the following fields:
 - **Standard**
 - **Context**: Context used for the reverse-engineering
 - **Types of objects to reverse-engineer**: List of object types that should be taken into account by the reverse-engineering process.
3. Go to the **Selective Reverse** tab.
 - Check the **Selective Reverse**, **New Datastores**, and **Objects to Reverse** boxes.

4. A list of datastores to be reverse-engineered appears. Uncheck those that you do not wish to reverse-engineer.
5. Click **Reverse**, then **Yes** to validate the changes.
6. A progress bar appears as Oracle Data Integrator starts a reverse-engineering of the selected datastores

The reverse-engineered datastores appear under the model.

It is possible to refine the reverse-engineering using the options found in the **Reverse** and **Selective Reverse** tabs. Refer to Model for more information on this.

Result of the Reverse-Engineering

Oracle Data Integrator will automatically add the following columns to the tables generated from the XML file:

- Primary keys (**PK** columns) for parent-child relationships
- Foreign keys (**FK** columns) for parent-child relationships
- Order identifier (**ORDER** columns) to enable the retrieval of the order in which the data appear in the XML file.

These extra columns enable the hierarchical XML structure's mapping in a relational structure stored in the schema.

XML

Creating an XML Data Server

An XML **Data Server** corresponds to one XML file that is accessible through your local network.

XML files are accessed through the **Oracle Data Integrator Driver for XML**. This JDBC driver loads the XML file hierarchical structure in a relational structure in a **schema** stored in memory to enable SQL queries through JDBC. It is also able to unload the relational structure back in the XML File.

Pre-requisites

JDBC Configuration

XML files are accessed through the **Oracle Data Integrator Driver for XML**. The driver is installed with Oracle Data Integrator.

Connection Information

You must ask the system administrator for the following information:

- The location of the **DTD file** associated with your XML file
- The location of the **XML file**
- The name of the **Root** element of your XML file

Creation of the Data Server

To create an XML Data Server:

1. Connect to **Topology Manager**
2. Select **Topology > Physical Architecture > Technologies > XML** in the tree
3. Right click and select **Insert Data Server**
4. Fill in the following fields in the **Definition** tab:
 - **Name:** Name of the Data Server as it will appear in Oracle Data Integrator.
 - **User/Password:** Not used here.
5. Fill in the following fields in the **JDBC** tab:
 - **JDBC Driver:** `com.sunopsis.jdbc.driver.xml.SnpsXmlDriver`
 - **JDBC URL:** `jdbc:snps:xml?[property=value&property=value...]`

JDBC Driver Properties:

Parameter	Value	Notes
f	<XML File location>	XML File location (relative or absolute) in UNC format. Use slash "/" in the path name and not backslash "\ in the file path.
d	<DTD File location>	DTD File location (relative or absolute) in UNC format. Use slash "/" in the path name and not backslash "\ in the file path. If this parameter is missing, the driver will build the name of the DTD file from the XML file, replacing the ".xml" extension with ".dtd".
re	<Root element>	Name of the element to take as the root table of the schema. This value is case sensitive. This parameter can be used for reverse-engineering a specific message definition from a WSDL file, or when several possible root elements exist in a XSD file.
ro	true false	If true, the XML file is opened in read only mode.
s	<schema name>	Name of the relational schema where the XML file will be loaded. If this parameter is missing, a schema name is automatically generated from the file name.
cs	true false	Load the XML file in case sensitive or insensitive mode. For case insensitive mode, all element names in the DTD file should be distinct (Ex: Abc and abc in the same file are banned). The case sensitive parameter is a permanent parameter for the schema. It CANNOT be changed after schema creation. Please note that when opening the XML file in insensitive mode, case will be preserved for the XML file.

Example:

```
jdbc:snps:xml?f=../demo/xml/GEO_DIM.xml&re=GEOGRAPHY_DIM&ro=false&case_sens=true&s=GEO
```

Warning ! To access an XML file, you must first ensure that the file is not locked. To check if a file is locked, look for a .lck file in the directory of your XML file.

6. Click **Test**.
7. Click **Test** in the **Test Connection** window.

8. A windows indicating a **Successful Connection** should appear. Click **OK**. If the connection is not successful, then refer to the Common Errors with XML.
9. Click **OK** to validate the creation of the data server


The creation window for the data server's first **Physical Schema** appears.
See Creating an XML Physical Schema

Creating a Physical Schema for XML

The **Physical Schema** will be a storage location for the tables associated with the XML file.

To Create a Physical Schema for XML:

Note: If you have just created a Data Server, ignore step 1. The **Physical Schema** window should already be opened.

1. Select the appropriate XML **Data Server** then right-click and select **Insert Physical Schema**. The **Physical Schema** window will appear.
2. Name the **Schema** and **Work Schema**. Note that if you have named the **schema** with the `s=<schema name>` property of the JDBC URL of the XML **Data Server**, you must use the same schema name here.
3. Check the **Default** box if you want this schema to be the default for this data server (The first physical schema is always the default). For more information, see Physical Schema. Note that for XML files, you do not need any other Physical Schemas
4. Go to the **Context** tab
5. Select a **Context** and an existing **Logical Schema** for this new **Physical Schema**, then go to step 8.
If no XML Logical Schema exists yet, proceed to step 6
6. Click the  button
7. Select an existing **Context** from the left column, then type the name of a **Logical Schema** into the right column. This XML Logical Schema is automatically created and associated with this physical schema in this context

Warning! A **Logical Schema** can be associated with only one **Physical Schema** in a given **Context**

8. Click **OK**

Creating and Reverse-Engineering a Model for an XML file

Creating a Model for an XML file

An XML file **Model** will be seen as a set of datastores, with each datastore representing an entry level in the XML file. A model is always based on a **Logical Schema**. In a given **Context**, the **Logical Schema** corresponds to one **Physical Schema**. This Physical Schema's corresponding Data Schema contains the tables for this XML model.

To create a Model for an XML file:

1. Connect to **Designer**

2. Select **Models** in the tree
3. Right click and select **Insert Model** from the menu
4. In the **Definition** tab, fill in the **Name** field
5. In the **Technology** field, select **XML**
6. In the **Logical Schema** field, select the Logical Schema on which your model will be based
7. Go to the **Reverse** tab and select the **Context** that will be used when reverse-engineering the model. Click **Apply**

The model is created but does not contain any datastores yet.

Reverse-Engineering an XML Model

A model is created with no datastore. The **Reverse-Engineering** operation gathers the structure of the model's tables to create the appropriate datastore definitions. There are two types of reverse-engineering: **Standard** reverse-engineering, which uses only the abilities of the driver, and **Customized** reverse-engineering, which uses a RKM (Reverse Knowledge Module) to get the structure of the objects.

Note: There is no out of the box RKM for XML files. It is recommended that you use standard reverse-engineering for XML files.

Standard Reverse-Engineering

To perform a Standard Reverse-Engineering on an XML file:

1. Go to your XML **Model's Reverse** tab
2. Fill in the following fields:
 - **Standard**
 - **Context:** Context used for the reverse-engineering
 - **Types of objects to reverse-engineer:** List of object types that should be taken into account by the reverse-engineering process.
3. Go to the **Selective Reverse** tab
 - Check the **Selective Reverse**, **New Datastores**, and **Objects to Reverse** boxes
4. A list of datastores to be reverse-engineered appears. Uncheck those that you do not wish to reverse-engineer
5. Click **Reverse**, then **Yes** to validate the changes
6. A progress bar appears as Oracle Data Integrator starts a reverse-engineering of the selected datastores

The reverse-engineered datastores appear under the model.

It is possible to refine the reverse-engineering using the options found in the **Reverse** and **Selective Reverse** tabs. Refer to Model for more information on this.

Result of the Reverse-Engineering

Oracle Data Integrator will automatically add the following columns to the tables generated from the XML file:

- Primary keys (**PK** columns) for parent-child relationships
- Foreign keys (**FK** columns) for parent-child relationships

- Order identifier (`ORDER` columns) to enable the retrieval of the order in which the data appear in the XML file.

These extra columns enable the hierarchical XML structure's mapping in a relational structure stored in the schema.

Choosing the Right KMs for XML files

The choice of KMs in an interface determines the interface's abilities and performance. The recommendations below help choose the right KM for an XML file according to specific situations.

For generic information about the KM, see Knowledge Modules

Note: Only knowledge modules imported into a project are available for use in the project's interfaces. To import a KM, see Import a KM.

Synchronize file and memory

To ensure a perfect synchronization of the data in an XML file and the data in memory, certain commands have to be called:

- Before using the tables of an XML model, either to read or update data, it is recommended you use the `SYNCHRONIZE FROM FILE` command on the XML Logical Schema. This operation reloads the XML hierarchical data in the relational memory schema. The schema being loaded in memory when first accessed, the data in memory may not be synchronized the content of the XML file.
- After performing changes in the relational memory schema, you must unload this schema into the XML hierarchical data by calling the `SYNCHRONIZE ALL` or `SYNCHRONIZE FROM DATABASE` commands on the XML Logical Schema.

These commands must be executed in procedures in the packages before (and after) the interfaces and procedures manipulating the XML schema.

Refer to Oracle Data Integrator XML driver documentation for more information on these commands.

Warning! If the schema for the XML file is not set in the JDBC URL, the command `SET SCHEMA` must be called before processing any data.

Interfaces to XML

When using a table of an XML model as a target of an interface, you must insert data in the extra columns mapping the XML hierarchical structure in the relational structure. For instance, if filling `region` records in an XML structure like :

```
<country COUNTRY_ID="6" COUNTRY_NAME="Australia">
  <region REGION_ID="72" REGION_NAME="Queensland">
</country>
```

You must fill in the columns `REGION_ID` and `REGION_NAME` of the `region` table, but also fill:

- `REGIONPK`: This column enables you to identify each `<region>` tag.
- `REGIONORDER`: This column enables you to order the `<region>` tags in the XML file (records are not ordered in a relational schema, whereas XML tags are ordered)

- **COUNTRYFK:** This column enables you to put the <region> element in relation with the <country> parent element. This value is equal to the COUNTRYPK value for the "Australia" record in the country table.

To ensure data integrity, Oracle Data Integrator inserts specific **references** and **primary key** constraints in the XML model.

Loading data to and from an XML file

An XML file can be used as an interface's source or target. The choice of the LKM (in the Interface Flow tab) used to load data between XML files and other types of data servers is essential for the execution of the interfaces.

LKM Choice from an XML Schema

LKM recommendations in the following cases:

- XML Source to a Staging Area

If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Staging Area	Recommended KM	Notes
Microsoft SQL Server	LKM ISO SQL to MSSQL (bulk)	Uses SQL Server's bulk loader.
Oracle	LKM ISO SQL to Oracle	Faster than the Generic LKM (Uses Statistics)
Sybase	LKM ISO SQL to Sybase (bcp)	Uses Sybase's bulk loader.
All	LKM ISO SQL to SQL	Generic KM

LKM Choice to an XML Schema

It is not advised to use an XML Schema as a staging area, except if XML is the target of the interface and you wish to use the Target as a Staging Area. In this case, it might be required to load data to an XML Schema.

LKM recommendations in the following cases:

- Source to an XML Staging Area

If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Source	Recommended KM	Notes
JMS	LKM JMS to SQL	
File	LKM ISO File to SQL	
All	LKM ISO SQL to SQL	Generic KM

Integrating data into an XML file

The data integration strategies in an XML file are numerous and cover several modes. The IKM choice in the Interface Flow tab determines the performance and possibilities for integration.

IKM choice for an XML file

LKM recommendations in the following cases:

- Staging Area to an XML Target.
- XML Staging Area to XML Target. In this case, The Staging area is on the XML Target.

If several solutions are possible, they are indicated in the order of preference and performance. Generic KM are in bold.

Mode	Staging Area	Recommended KM	Notes
Update	XML	IKM ISO SQL Incremental Update	Generic KM
Append	XML	IKM ISO SQL Control Append	Generic KM
Append	All RDBMS	IKM ISO SQL to SQL Append	Generic KM

Common Errors with XML

Detect the errors coming from XML

Errors appear often in Oracle Data Integrator in the following way:

```
java.sql.SQLException: No suitable driver
at ...
at ...
...
```

the `java.sql.SQLException` code simply indicates that a query was made through the JDBC driver, which has returned an error. This error is frequently a database or driver error, and must be interpreted in this direction.

Only the part of text in bold must first be taken in account. It must be searched in the XML driver documentation. If its contains a specific error code, like here, the error can be immediately identified.

If such an error is identified in the execution log, it is necessary to analyze the SQL code send to the database to find the source of the error. The code is displayed in the **description** tab of the **task** in error.

The most common errors with XML are detailed below, with their principal causes.

Common Errors

Connection Errors

No suitable driver

The Oracle Data Integrator Driver for XML is not correctly installed.

Errors in the interfaces

File <XML file> is already locked by another instance of the XML driver.

The XML file is locked by another user/application. Close all application that might be using the XML file. If such an application has crashed, then remove the .lck file remaining in the XML file's directory.

The DTD file "xxxxxxx.dtd" doesn't exist

This exception may occur when trying to load an XML file by the command LOAD FILE. The error message can have two causes:

- The path of the DTD file is incorrect.
- The corresponding XML file was already opened by another schema (during connection for instance).

Table not found: S0002 Table not found: <table name> in statement [<SQL statement>]

The table you are trying to access does not exist in the schema.

Column not found: S0022 Column not found: <column name> in statement [<SQL statement>]

The column you are trying to access does not exist in the tables specified in the statement.

Glossary

A

Action: Actions are templates for Data Definition Language (DDL) commands.

Agent: Java software components that allow Oracle Data Integrator jobs to be executed on a remote machine.

C

Common Format Designer: Common Format Designer (CFM) is used to quickly design a data model from the Designer user interface.

Connection: A Connection is how Oracle Data Integrator connects to a data server. It requires (in most cases) a user name (login) and a password. A connection can be managed through a LDAP directory. A single connection enables access to several schemas stored in the same data server.

Context: A context is a set of resources allowing the operation or simulation of one or more data processing applications.

D

Data Server: A data server is a data processing resource which stores and reproduces data in the form of tables. It can be a database, a MOM, a connector or a file server.

Data Type: Category or nature of the data. The technologies storing the formatted data allocate to each of them a type that defines their nature.

Datastore: A datastore is a structure that allows data to be stored. It can be a table, a file, a message queue or any other data structure accessible by middleware compatible with Oracle Data Integrator (JDBC/ODBC, JMS or JNDI).

Diagram: A diagram is a graphical view of a subset of the datastores contained in a sub-model(or data model).

Driver: Software component provided as a set of Java classes, enabling access to external data.

F

Flow: Oracle Data Integrator Entity enabling a Datastore to load from several source datastores.

Folder: A folder is a group of packages, interfaces and specific procedures. Folders and sub-folders allow these objects to be grouped and organized according to specific criteria. Sub-folders can be created to an unlimited number of levels.

G

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

I

Integrity Constraints: Rule defining the validity limits for information. The integrity constraints are attached to datastores. There are several types of constraint: References, primary keys, alternative keys, Oracle Data Integrator Controls.

Interface: An interface consists of a set of rules that define the loading of a Datastore or a temporary target structure from one or more source Datastores. The Designer module allows interfaces to be defined and tested.

Interface IN: These generated integration interfaces are used to load the model's datastores assembled from other datastores/columns. They are the integration process merging data from the original datastores into the composite datastores.

Interface OUT: These integration interfaces are used to extract data from the model's datastores. They are generated using the interfaces (including the interfaces IN) already loading the model's datastore. They reverse the integration process to propagate the data from the composite datastore to the original datastores

J

Java Virtual Machine: Environment enabling compilation and execution of Java programs. All Oracle Data Integrator components require a JVM to run.

JDBC: JDBC (Java Database Connectivity) a standard Java API providing access to the data contained in an RDBMS. It requires the installation of a JDBC driver if the latter is not provided with Oracle Data Integrator.

JMS: Java Message Service a standard Java API created by Sun Microsystems to provide access to the MOM.

JVM: Java Virtual Machine

L

LDAP: "Lightweight Directory Access Protocol" is an access protocol for a directory of corporate resources. These resources are organized in hierarchies and represent employees, departments, machines, databases, sites. Access to an LDAP directory is secured by user names and passwords.

M

Meta-data: Set of data describing other data. It is a description of the structure of the tables and columns in a database contained in a data server.

Middleware: Software component enabling communication between two programs in client/server mode. This type of software is generally based on a standard (JDBC, ODBC, JMS) linked to the server's technology type.

Model: Data Physical Model.

Module: Software provided by Oracle Data Integrator connecting the Repositories and providing a set of features useful for a group of people. Example: Designer, Agent, Security Manager ...

MOM: Message Oriented MiddleWare: Tools enabling events information transport in a structured format (text, XML, objects) or not between remote heterogeneous systems. Information pools managed by the MOM are QUEUES and TOPICS.

O

ODBC: ODBC (Open Database Connectivity) is an API providing access to the data contained in a RDBMS. It requires the installation of an ODBC driver, specific to the RDBMS accessed and the client operating system.

P

Package: A package is a set of sequenced jobs, also called steps, designed to be executed in a pre-defined order.

Physical Schema: The physical schema is a decomposition of the data server, allowing the Datastores (tables, files, etc) to be classified.

Pooling: Action consisting of repetitively interrogating an application to retrieve the new events.

Project: A project is a group of objects developed using Oracle Data Integrator.

Q

Queue: Information pool managed by a MOM enabling publishing of a type of event by an application and consumption of this same event. The queue is used for point to point asynchronous communication between applications.

R

RDBMS: DataBase Management System. It is a data server such as Oracle, Sybase, DB2 ...

Reference: Functional Link between two datastores.

Repository: Repository contains all information required for developing and operating interfaces. The Repository is stored in a database accessible by several users simultaneously.

Reverse: Reverse engineering consists of retrieving the meta-data from a data server repository to store them in the Repository.

S

Sequence: A sequence is a variable that increments each time it is used.

Session: A session is an execution (of a scenario, an interface, a package or a procedure, ...) undertaken by a Oracle Data Integrator execution agent. A session is made up of steps which are made up of tasks.

Solution: A solution is a comprehensive and consistent set of interdependent versions of Oracle Data Integrator objects. It can be checked in at a given time in a version and that may be restored at a later date. Solutions are saved into Oracle Data Integrator Master Repository. A solution assembles a group of versions called the solution's elements.

Sub-Model: A sub-model is a group of functionally homogeneous datastores within a model.

T

Technology: In Oracle Data Integrator terminology, this is any type of technology accessible by JDBC, ODBC, JMS, JNDI, JCA, or any operating system.

Topic: Information pool managed by a MOM enabling communication in "Publish and Subscribe" mode. An application that wishes to consume a type of event must subscribe to this type of event. Each event is consumed by all subscribers for the type of event.

U

URL: "Uniform Resource Locator". Syntax Element enabling to locate a resource accessible by TCP/IP. Each URL begins with the name of the protocol used, the most common is "http".

V

Version: A version is a backup copy of a Oracle Data Integrator object. It is checked in at a given time and may be restored later. Versions are saved in the Oracle Data Integrator Master Repository.